Optimizing Service Ecosystems in the Cloud

Usman Wajid, César A. Marín, and Nikolay Mehandjiev

The University of Manchester

{firstname.lastname}@manchester.ac.uk

Abstract. A service ecosystem is a virtual space ideally distributed across networks and geographical areas where vast numbers of services and other digital entities can coexist and converge to form *ad-hoc* solutions. In this paper we present experimental results showing the performance of two optimization models in service ecosystem. We describe the two models and how they operate under service ecosystem conditions. We emulate service ecosystem conditions in a multi-site federated Cloud and test the two models under different scenarios. The experimental results help us to determine strengths and weaknesses of the two optimization models enabling us to make recommendations for their use in different application domains.

1 Introduction

The proliferation of web-based services and rapid adoption of SOA have not only changed the perception of services but in effect also changed the way in which services are offered and consumed. An emerging development in this area is the notion of service ecosystems that can be seen as a virtual space where services and other digital entities interact and form (service-based) solutions at higher levels of granularity to achieve specific objectives. Dynamism is one of the fundamental characteristics of service ecosystem that allows services to appear and disappear at any time thus enabling the creation of ad-hoc solutions composed of various services.

Developing composite service-based solutions involves combining the functionality of multiple services in a unified solution on the basis of several factors, e.g., cost, performance, SLAs, etc. However, an automated approach for service composition may end up with sub-optimal solution (or composition) within service ecosystem because better services may become available soon after a composition was developed or some services in the composition may disappear with key functionality. These and other similar scenarios (e.g. choosing a particular service when the same functionality is offered by a number of services) represent a multi-criteria optimization problem creating the need for composition optimization to determine the best mix of services that can contribute towards the overall goal of the composition.

In this paper we describe two existing optimization models and test suitability of these models for optimizing service compositions in a service ecosystem while taking into account the dynamism of the domain. Building on our earlier research in service-based systems [1, 11] and complex adaptive systems [3, 4], our two service optimization models are characterized as global optimization and local optimization. The first

one, developed in the European Commission funded SOA4All project (<http://www.soa4all.eu>), computes the optimization of a service composition from a holistic point of view by analyzing end-to-end connections between composite services together with their non-functional properties [1, 2]. In contrast, the second optimization model called DAEM (Dynamic Agent-based Ecosystem Model) is able to compute local optimizations of service compositions by allowing the one-to-one interactions between service providers and consumers to create emergent service chains providing composite services that are resilient to changes [3, 4].

Both optimization models are computationally intensive, and this impedes their ability to optimize compositions involving a large number of services. To resolve this obstacle and to create necessary conditions for a service ecosystem, we need a suitable infrastructure that allows practical use of the optimizers and supports the dynamism and open nature of our target service ecosystems. The summary requirements to the supporting infrastructure are as follows:

- Multiple Cloud facilities to emulate the conditions for geographically distributing resources and services (i.e. candidate services that can be used in a composition) both manually and dynamically
- Seamless communication across inter- and across distributed sites
- Availability of on-demand computational resources; and
- Monitoring capabilities of computational resources both at runtime and after execution for monitoring data collection.

Fig. 1. Multi-site federated Cloud facilities offered by BonFIRE (<http://www.bonfire-project.eu/infrastructure>)

Based on the above requirements, we have selected the BonFIRE multi-site Cloud infrastructure presented in Fig. 1 (<http://www.bonfire-project.eu/>) as a suitable facility for running our optimization models.

In this respect, the contribution of this paper is twofold (i) testing the maturity of existing technology represented by two optimization model and its suitability for use in Future Internet scenarios such as service ecosystem, and (ii) investigating the

existing infrastructure offerings such as BonFIRE and evaluate its support for Future Internet scenarios.

The rest of the paper is organized as follows: Section 2 presents the details about the two optimization models and the objectives of our tests, Section 3 outlines the design aspects of our experiments, and Section 4 presents a findings of the experiments, Section 5 concludes the paper with recommendations for using optimization models in different application domains and directions for future work.

2 Overview of Optimization Models

Our optimization models were designed to solve optimization problems using different techniques and we characterize the models according to the perspective they use to tackle the optimization problem i.e. globally and locally.

2.1 SOA4All Optimizer

The global model or *SOA4All optimizer* was designed to mainly work with semantic web services. It uses semantic link-based approach [2] for service composition. A semantic link is established between the output and input parameter of two services as a partial link indicating matching functionality. The matching functionality is determined by a matching function [1] that enables the estimation of semantic compatibilities among the descriptions of different services. This matching function is then used to define the quality of data flow in service composition at semantic level. This functional quality estimator is extended by considering non-functional properties of services also known as QoS parameters in the composition. These QoS parameters include Execution Price (i.e. fee requested by the service provider) and Response Time (i.e. expected delay between request and response time). Thus extended, the quality estimator allows the selection of the ‘best’ pairs of services in a composition. Further details about the optimization model, semantic links and the computation of functional and non-functional quality constraints can be found in [1, 2, 9]

The global optimization model uses this extended quality estimator and comprises the following three components.

- *Genetic Algorithm-based optimizer* a robust combinatorial optimization tool for generating alternative compositions based on different quality weights/criteria.
- *Reasoning engine*: a reasoner for computing the semantic similarities (and matching quality) between composite services.
- *Service repository* for hosting candidate services to be used in the composition.

The above components communicate extensively with each other and can be deployed in a distributed manner.

2.2 DAEM Optimizer

On the other hand, the local optimization model, *DAEM* [3], was developed to test the emergence of service value chains while dealing with unexpected disturbances. It is inspired by ideas from natural ecosystems where local, dynamic interactions are fundamental to the creation of ecosystems [8]. In *DAEM* links between services are created according to individual input and output matching of two services regardless of how that would affect the overall composition. Thus optimization occurs locally when a service selects another service (from a set of available services) that offers best match for its output. Likewise the other service also tries to select the service offering best input. As a result a composition emerges from several locally optimized input-output matchings.

Fig. 2. The DAEM environment sets the rules and necessary conditions for agent interactions

The optimization model contains two types of components.

- *Environment* as a fundamental virtually observable surface where inhabitants (i.e. agents representing services) wander across and encounter others in order to interact. The environment sets the rules for agent interactions as well as mediates agent communication, as shown in Fig. 2.
- *Agent(s)* represent service providers and consumers in the service ecosystem.

Like SOA4All, the above components in *DAEM* require extensive communication and can be deployed in a distributed manner.

In this respect, the distributed nature of the two optimization models makes them suitable for testing in distributed computational environment.

3 Experimental Design

The experiments concerning both optimization models are designed to solve the problem archetype shown in Fig. 3.

The composition problem shown in Fig. 3 is represented as a service composition process template where various activities (shown by rectangular boxes) are interlinked with connectors (arrows). We choose this template because it contains the basic connectors namely XOR/AND joins and splits and sequence operators.

In the experiments the two optimizers start with a desired process template and select suitable services or agents to perform each activity in the template. The selection is made by matching the description of services (or capabilities of agents in case of

DAEM) with the description of the activities constituting the template. The services or agents are selected from a repository of services or agent clusters, which depending on the deployment configuration can be deployed at the same site as other optimizer components or it can be distributed across different Cloud sites.

Fig. 3. Composition problem - service composition process template

Once the optimizers associate at least one service with each activity in the template the result is categorized as a non-optimal composition where there is at least one service associated with each activity. The optimizers then try to optimize the composition by replacing already associated services with better candidates using their inherent optimizing techniques. E.g., global optimization generates several alternative compositions and eventually selects the optimal one based on certain factors e.g. functional and non-functional quality of links/connections between composite services.

Whereas, in local optimization agents interact with each other in order to establish preferred provider-consumer chains (for each activity in the process template) that can change when agents find better partners.

Configurations	Scenarios	
	Controlled Environment	Dynamic Environment
One site	Exp1	Exp2
Fixed distribution	Exp3	Exp4

Fig. 4. Configuration and Scenarios used for testing optimization models

In this respect, we designed the experiments concerning the two optimization models to pursue the following objectives.

Objective O1 – To determine the optimal manner of distributing, decoupling and parallelizing the computationally intensive properties of optimization models on a large scale distributed architecture.

Objective O2 - To determine conditions under which both models achieve optimal composition e.g. by running predefined scenarios (involving dynamic changes in deployment configuration, number and properties of services, etc.)

Objective O3 - To analyze the behavior of testbed or Cloud infrastructure (i.e. BonFIRE) by changing dynamically the population size (i.e. number of candidate services or agents) and testing different features offered by the underlying

infrastructure such as how asynchronous interactions between multiple entities are governed and managed by BonFIRE infrastructure, keeping in view the dynamism.

To achieve the above objectives the experiments are designed to test different scenarios and also different deployment configurations, as shown in Fig. 4, and elaborated below:

Scenario I- Controlled Environment: In this scenario we introduce specific changes to the candidate services at specific times, thus changing the problem the optimisers are trying to solve one at a time.

Scenario II- Dynamic Environment: In this scenario we emulate the dynamism of a real service ecosystem where services come and go and change with less control.

Configurations I- One Site: We run each optimiser and related components in a single BonFIRE site

Configurations II- Fixed Distribution: We distribute the optimisers' components across different BonFIRE sites.

Fig. 5. Design of Exp1 (based on the scenarios and configurations in Fig. 4)

Fig. 6. Design of Exp4 (based on the scenarios and configurations in Fig. 4)

The experiments were specifically designed to allow the study of optimizers in service ecosystem conditions, such as the effect of distributing optimizer components

on the performance of optimizer, tradeoff between selecting services from one site or various sites and the effect of dynamic changes (in the number and properties of candidate services) on the optimizer outcome. In this respect, based on the combination of scenarios and configurations (from Fig. 4) the experiments range from Exp1 (as shown in Fig. 5) where all components of the optimizers reside at the same BonFIRE site, to Exp4 where the components of the optimizers are scattered over multi-site BonFIRE Cloud (as shown in Fig. 1).

4 Experimental Results

We ran 20 simulations for each experiment mentioned in Fig. 4 for each of the optimizers. For each experiment we calculate the median in fixed time intervals. The median was chosen because it does not assume any distribution in the analyzed data.

In each simulation run we first added services to the ecosystem in an incremental manner while consecutively optimizing service compositions. Once we reached an empirical maximum (10,000 service), we semi-randomly changed their service parameters while we carried on with consecutive optimizations. Finally, we reduced the number of services little by little while still optimizing using remaining services. This process was done for both optimizers in order to analyze how the optimization results changed over time by making differentiations in the problem they were optimizing. This emulates the dynamism of service ecosystem which the optimizers do not have control over but have to take into account by consecutive optimizations.

The behavior of the optimizers in service ecosystem conditions is described here.

4.1 Soa4all Is Faster on a Single Site, Whereas DAEM is Faster in a Multi-site Configuration

Our tests reveal the ability of SOA4All optimizer to demonstrate better performance (in terms of time taken to converge to an optimal composition) in a single site deployment configuration i.e., when all components of the optimization model were deployed on different VMs but on the same site.

Fig. 7. Performance in different deployment configurations (shown with the help of box plots)

Whereas, contrary to the expected results DAEM demonstrated better performance by taking less time to finish an experiment when deployed in a multi-site configuration i.e. when the environment and agent clusters were deployed on geographically distributed Cloud sites. This behavior is due to an emergent parallelization of component execution. That is, DAEM uses a fixed waiting time (30ms) in between executions in order to allow information to be exchanged between the agents and the environment. In a multi-site configuration, because of the communication delay due to geographic distribution, the environment component starts next execution before receiving all the information. Consequently, it processes the information already received at the same time while it keeps on receiving more. Then after execution it realizes it does not need to wait again because information is already there. Thus it executes again while more information is being received. As a result, there is a parallelization between information transfer and information processing resulting in shorter experiment time than when deployed all in a single site.

In a single site configuration this does not occur because the network is faster thus all the information is received before processing each time. As a result there is a always a waiting time in between executions. This is reflected in Fig. 7.

4.2 Soa4all Is More Scalable than DAEM

Our tests reveal the ability of SOA4All optimizer to be highly scalable without putting too much stress on the underlying infrastructure. As shown in Fig. 8, SOA4All optimizer was tested with up to 10,000 services (i.e. 10,000 candidate services were considered while optimizing the composition template from Fig. 3) without putting too much stress on the infrastructure i.e. in this case memory usage.

Fig. 8. Scalability comparison between the two optimizers

On the other hand, DAEM faces limitations on this front owing to the computation intensive nature of the approach. In terms of scalability, DAEM struggled to maintain memory consumption when the number of agents was increased to just 100.

This lack of scalability in DAEM can be attributed to the lack of temporary storage of partial results into disk, e.g. in a database. DAEM maintains everything in memory.

4.3 DAEM's Output Is More Stable and Resilient to Changes than SOA4All's

Let us remind the reader that optimizations are consecutive, one immediately after another while in parallel we vary the number of services and their properties to emulate the dynamism of a service ecosystem.

Fig. 9. Comparison of optimizers' output in service ecosystem conditions

The results reveal that SOA4All optimizer's output oscillates frequently without any hints of stabilization. On the other hand, DAEM produces a more stable output regardless of the dynamism of service ecosystem. Fig. 9 depicts these results. In this respect, we can say that DAEM is resilient to changes because it overcomes each consecutive change in the service ecosystem at local level (only changing aspects of the composition that are affected by the change) thus not negatively affecting the final or overall output.

This behavior of DAEM is attributed to the ecosystem properties embedded in it such as agents' short memory which allows the system to absorb any change (new services, removal of services, and change of parameters) and propagate only those that have a positive impact on the agent who detected them. Consequently, a resilient service composition emerges.

4.4 BonFIRE Support for Future Internet Scenarios

Testing the optimization models on BonFIRE also allowed us to test and evaluate various aspects of BonFIRE infrastructure. The evaluation of BonFIRE revealed the following characteristics that contribute towards creating and supporting service ecosystem conditions.

Dynamism: A simple way to create service ecosystem conditions for our experiments was to enable dynamism at experiment as well as infrastructure level. That is, dynamism not only referred to the ability of services and other software entities to appear, disappear and change their properties at any time but also to the dynamic fluctuations in the infrastructure offerings such as appearance and disappearance of computational and storage resources. Our experimentation reveal that BonFIRE supports both types of dynamism by i) allowing users to have full control over their experiments, and ii) providing APIs to manage infrastructure resources not only at design time but also at runtime.

Deployment and experiment management: In BonFIRE the initial deployment and management of experiments can be done by using BonFIRE portal that offers a simplistic visual interface for creating experiment, initializing VM instances, allocating computation and storage resources to VMs and deployment of experiments on user specified locations within BonFIRE infrastructure. The portal allow users to manage their experiments at run time e.g. add-remove infrastructure resources, stop-start experiment and eventually terminate the experiment. Having an easy to use graphical interface allow users to effectively manage their experiment and efficiently interact with underlying BonFIRE infrastructure.

Monitoring: Reliable monitoring of experiments was vital to track the performance of optimization models under dynamic conditions. In this respect, the BonFIRE monitoring mechanism allowed effective management of running experiments by means of an intuitive GUI. The visual interface of the monitoring mechanism further simplifies the monitoring of experiments and related activities such as load balancing, service monitoring, monitoring of infrastructure or resource utilization and scalability benchmarking.

Authentication, authorization and cross site communication: Making use of geographically distributed computational and storage resources require certain mechanisms to be in place for seamless authentication, authorization and communication. In BonFIRE the *Identity Manager* offers a single sign-on service for users accessing resources from multiple (geographically distributed) Cloud facilities. The service is also responsible for authentication of software/application components thus allowing different components (residing on different Cloud facilities) to interact without any problem. All Cloud facilities in BonFIRE are connected via a BonFIRE WAN that ensures seamless communication and data exchange (using unique IP addresses) between geographically distributed infrastructure resources.

Overall BonFIRE proved to be a suitable platform for supporting future Internet scenarios that may involve experiments using large number of services, geographically distributed components and requiring on-demand computational and storage resources.

5 Discussion about Related Work

While existing works (e.g. [5, 6]) investigate the QoS and network latency issues in dynamic service composition, they do not particularly focus on issues associated with designing and testing service composition mechanisms in service ecosystem conditions. Further, in [7] a distributed Cloud service composition framework is proposed together with protocols to dynamically select candidate services and form compositions. The service composition protocols support incomplete information about Cloud participants when applying dynamic service selection mechanisms. The main limitations of [7] stem from the scale of the composition scenarios evaluated taking into account small numbers of services and compositions and also the use of desktop based environment instead of a Cloud facility.

In terms of experiments on service ecosystems or similar technologies, [11] proposes an architecture inspired by ecosystems portraying self-adaptation and self-organization to support dynamic scenarios, such as service ecosystems. The approach considers four types of ecosystem entities such as flora and consumers, and niches represented by tuple spaces which function as the interaction interfaces. The entities (agents) have needs and a “happiness” status they try to maximize by fulfilling their needs. Experiments show how the “happiness” levels reach a balanced state at the end of their simulations throughout a set of niches, however this only demonstrate its capacity to converge once to a solution rather than its capacity to deal with continuous changes and dynamism, as we do in this paper.

6 Conclusion

Our experiments provide an opportunity to test the optimization models as well as the Cloud infrastructure in a Future Internet scenario. The results of our experiments reveal interesting characteristics of the optimizers under different scenarios and deployment configurations.

In particular the experiments on BonFIRE helped us to identify the strengths, weaknesses and improvement opportunities of the optimization models, which allow us to make recommendations about the use of optimization models in different application domains.

SOA4All Optimizer: The experimental results reveal that SOA4All optimization is suitable for one-off optimization and/or when unprecedented numbers of services are involved in the composition. This can be case in travel booking applications or financial applications that consider huge number of services before producing a composition that is best suited for the user.

For example, in a travel booking application a user can request a composition of services to represent and automatically perform activities (based on user input/requirements) such as booking airline tickets, travel insurance, hotel reservation, hire car and taxis to and from airport. Up on receiving such a request the SOA4All optimizer can perform the following steps:

- Search the service repository to find suitable services that can perform the user specified activities.
- Select at least one service to perform each of the user specified activity based on the matching the descriptions of the activity and the service. The end result of this step will be a non-optimal composition.
- Starting from a non-optimal composition the optimizer will replace the already associated services with better alternatives while considering the quality of semantic links between the services and other non-functional parameters that affect the overall quality of the composition. This step may involve going through and considering thousands of candidate services for each activity. The end results of this step will be an optimal composition that best suits user requirements.

DAEM: Contrary to the one-off nature of SOA4All optimization, the type of optimization offered by DAEM is suitable for continuous processes as in the case of optimizing traffic infrastructure or complex manufacturing processes.

In the traffic domain the agents in DAEM can represent different entities such as cars, roads and traffic sensors that communicate with each other in order to optimize the overall utilization of the available infrastructure resources.

In future work, we intend to analyze more data that was gathered during the experiments and use the analysis to study further details of our optimizers. The complete analysis will also allow us to make further technological improvements in the implementation of our optimizers and investigate their use in real work applications.

The current implementations of both optimization models are available as open-source resource. BonFIRE is available as an open access multi-site federated Cloud facility (<http://www.bonfire-project.eu/involved>) for researchers and experimenters to test their applications and technologies.

Open Access. This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

References

1. Lécué, F., Mehandjiev, N.: Seeking quality of web service composition in a semantic dimension. *IEEE Transactions on Knowledge and Data Engineering* (2010)
2. Lécué, F.: Optimizing QoS-Aware Semantic Web Service Composition. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) *ISWC 2009*. LNCS, vol. 5823, pp. 375–391. Springer, Heidelberg (2009)
3. Marín, C.A.: Adaptation to unexpected changes: where ecosystems and multiagent systems meet. PhD Thesis. Manchester Business School. The University of Manchester, UK (2010)
4. Marín, C.A., Stalker, I.D., Mehandjiev, N.: Engineering business ecosystems using environment-mediated interactions. In: Weyns, D., Brueckner, S.A., Demazeau, Y. (eds.) *EEMMAS 2007*. LNCS (LNAI), vol. 5049, pp. 240–258. Springer, Heidelberg (2008)
5. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: *GECCO*, pp. 1069–1075 (2005)
6. Calinescu, R.: Dynamic QoS Management and Optimization in Service Based Systems. *IEEE Transactions on Software Eng.* (2011)
7. Gutierrez-Garcia, J.O., Sim, K.-M.: Self-Organizing Agents for Service Composition in Cloud Computing. In: *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 59–66 (2010)
8. Green, D.G., Sadedin, S.: Interactions matter-complexity in landscapes and ecosystems. *Ecological Complexity* 2(2), 117–130 (2005)
9. Lécué, F., Léger, A.: A formal model for semantic web service composition. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) *ISWC 2006*. LNCS, vol. 4273, pp. 385–398. Springer, Heidelberg (2006)
10. Villalba, C., Mamei, M., Zambonelli, F.: A Self-organizing Architecture for Pervasive Ecosystems. In: Weyns, D., Malek, S., de Lemos, R., Andersson, J. (eds.) *SOAR 2009*. LNCS, vol. 6090, pp. 275–300. Springer, Heidelberg (2010)
11. Lécué, F., Wajid, U., Mehandjiev, N.: Negotiating Robustness in Semantic Web Service Composition. In: *Proceedings of IEEE European Conference on Web Services*, pp. 75–84. IEEE Computer Society (2009)