Composite service adaptation: a QoS-driven approach

Mario Henrique Cruz Torres
MarioHenrique.CruzTorres@cs.kuleuven.be

Department of
Computer Science
Katholieke Universiteit Leuven
3001 Heverlee, Belgium

Tom Holvoet
Tom.Holvoet@cs.kuleuven.be

Department of
Computer Science
Katholieke Universiteit Leuven
3001 Heverlee, Belgium

ABSTRACT

Nowadays it is possible to design and execute complex service orchestrations using standardized languages, such as Business Process Execution Language (BPEL). Even more, there are (open and closed source) tools capable of executing and monitoring service orchestrations using those languages. What is still lacking, however, is proper software support to maintain the desired quality of service (QoS) of the composite services, that are created using such orchestrations.
We propose a middleware solution that is able to maintain the required non-functional attributes, expressed as QoS, using a decentralized coordination mechanism. We evaluate how our approach can enhance current composite services regarding the availability, performance, robustness, and scalability.

1. INTRODUCTION

Nowadays, complex business processes can be modeled as composite services using BPEL and other service composition languages. These composite services describe the control and data flow needed to properly perform business processes. The composite services orchestrate component services, which in fact provide the functionality required by the composite service [3]. However, the binding between the composite and component services is rather static. If there are changes in the environment such as changes in the QoS of component services, the composite service instance is stuck with pre-defined component services and there is no simple way to change the component services at runtime.

Component services can be scattered around the internet and offer specific quality attributes to their clients. It is the responsibility of the composite service to bind to the best component services available at a certain time [5]. There are middleware capable of offering, at runtime, candidate component services to participate in a composition, however current solutions do not take advantage of the aggregated

COMSWARE 2011, July 04-07, Verona, Italy

DOI 10.4108/comsware.2011.5

We borrowed ideas from the Multi-Agent System (MAS) research community, more specifically from MAS coordination mechanisms [6]. We assume that each component service can be represented by an agent. The agent that represents a component service is called Resource Agent. Resource Agents are responsible for monitoring and maintaining the QoS information of their services. OrgAgents represent composite service's instances and are responsible for finding component services which satisfy the required end-to-end QoS of the composite service.

Our middleware creates an overlay network constituted of agents which represent each available service. The agents use the overlay network to disseminate their services QoS information, making it possible for OrgAgents to make use of this information. We used Ant Colony Optimization (ACO) algorithms to organize the agent's dissemination of information in the overlay network. We did that to benefit from ACO solution's properties such as resilience to failures and adaptation to dynamic changes in the environment.

We show, in this work, how the techniques used in our middleware contribute to create performant, efficient, and robust QoS-driven composite services in dynamic environments.

This paper is structured as follows. Section 2 describes the problem we are solving and the related work. We present our solution's technique in Section 3, then we evaluate our work in the following section, Section 4. Finally, in Section 5, we present our conclusions and directions for future research.

2. BACKGROUND & RELATED WORK

Service Oriented Computing (SOC) is a computing paradigm that has services as its core concepts. A service provides well defined functionality to its clients, can be remotely invoked and bound [8]. Another key characteristic of services is that they can be used to integrate functionality of different services, they can be composed, what is called a service composition.

Service composition is the premise of create large scale complex systems, that are needed in nowadays business environments. Composite services can be created from component ones allowing the creation of hierarchies of services, from infrastructure to business level services.

However, there is no simple way to select the component

services to participate in the service compositions. This problem, of selecting component services, is discussed in the following section.

2.1 Quality-driven Service Composition

The internet environment is very dynamic due to its distributed nature. Services can become available, unavailable, or present changing operational conditions while the composite service instances are being executed. In order to enhance composite service resilience to failures, performance, and scalability it is necessary to give proper adaptation support to the composite service engines.

A crucial aspect to the proper composite service execution is the selection of proper component services. The selection of component services must be done in order to fulfill the required end-to-end QoS from the composite service.

We assume that several services share the same interfaces to offer their operations. Services sharing the same communication interface are grouped in equivalence classes, called concrete target sets CTSs [5]. Any of the concrete services in a CTS implement the same interface and operations. A composite service is created and bound to the interfaces of services.

The problem then is how to select the best component services from a specific CTS that will participate in a composition. This problem is illustrated in Figure 1.

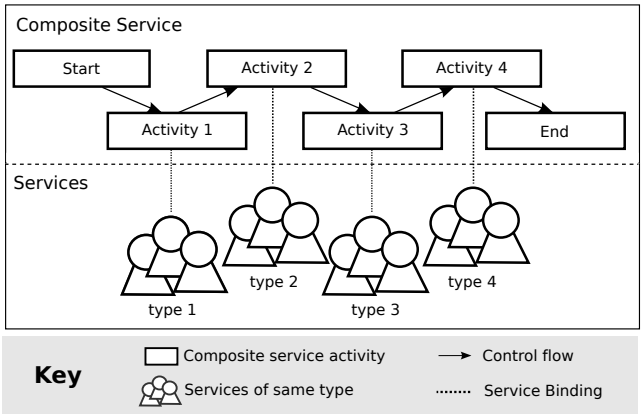

Figure 1: Service selection and binding in the many.

The computational cost of selecting component services regarding global constraints is know to be a NP-hard problem [7]. It is mandatory then, to have algorithms capable of doing a selection of component services in a short period of time.

The component services selection needs to be done according to the component service functional and non-functional characteristics and according to the required composite service end-to-end QoS. The composite service designer needs to specify a selection criteria using a service quality model.

2.2 Service Quality Model

A service quality model encompasses expected quality attributes from a component service. It needs to take into account quality attributes that allow the service clients do a proper service selection. We used the quality model proposed by [13] as a starting point to model the relevant service quality attributes.

In our quality model, there are mainly five service qualities, which are describe below:

- **Execution price.** Represents the cost of invoking a operation from a service
- **Execution duration.** Represents the time it will take for the service to execute the requested operation regarding the real world effect of the operation.
- **Successful execution rate.** Represents the probability that a operation is successfully completed and a message is sent to the requestor of the operation.
- **Availability.** Refers to the technical aspect of the success rate of contacting the service, having any kind of response.
- **Response time.** Represents the time it takes to contact a service through the network.

Possessing this quality model it is possible to evaluate the quality of each service, regarding its domain and technical qualities.

2.3 Related Work

A form of providing service adaptation is through late-biding. Component services are chosen and bind at runtime, also called bind-in-the-many [5], according to certain quality criteria. This selection and binding can be directly incorporated by service adaptation middleware or can be wrapped as a service that can be consumed through the network.

In this section we discuss a number of approaches and tools which cope with the problem of service adaptation and composition, specifically based on QoS properties of component services.

There are many platforms which offer support for composite service adaptation. Zeng et. al. show a middleware platform called AgFlow [13], which allows the deployment and execution of composite services.

In order to compare one service to another AgFlow defines a quality model to represent the component services non-functional properties. Different non-functional parameters are represented in this model, such as: the execution duration, which reflects the time it takes between sending a request and receiving its results; the execution price, which is the fee charged by a service provider to execute each offered operation; reputation, which is a measure of trustworthiness that mainly depends on the user experience of using the service; successful execution rate that is the probability that a request is correctly responded within a fixed time; and availability, which is the probability that a service is accessible. All these quality parameters are taken into account by the selection approaches of AgFlow. Different weights can be given to each parameter to show different concerns of the

composite service's designer, influencing how the selection algorithms will select the partner web-services.

The middleware provides two distinct approaches to adapt the composite service. The first approach is based on local optimization. The local optimization approach tries to maximize the utility function for each task to be performed. Using this approach the selection algorithm evaluates each partner's QoS according to the defined quality model and selects the best candidate for each activity. The local based approach is evaluated in a set of experiments and the results show that it consumes less time than the global optimization counterpart, however the results obtained for the service selection doesn't lead to a global optimal solution.

The second approach is based on global optimization. This approach considers the composite service's QoS constraints as a whole and selects partner services focusing on the entire composition. The author's experiments using the global planning approach indicate that the computational cost of the global planning algorithm is greater than the local optimization one, however the obtained component service selection are more satisfactory in terms of the whole composition, leading to lower costs, in terms of money, lower time to execute the composition and increased reliability. The experiments show, as expected, that global optimization leads to better results than local optimization, but also has greater computational costs. Both approaches provided by AgFlow use integer linear programming to implement the selection algorithms, what is best suited for small-scale problems as the computational costs increases exponentially with the problem size.

Services operate in a dynamic environment, problems can happen with the partner services, after the parter selection has been done, leading to sub-optimal solutions. AgFlow global optimization approach needs to be adapted in order to work in dynamic events, re-assigning the partner services to each task, but even this solution, has at least two drawbacks: i) can lead to sub-optimal solutions; ii) has a expensive computational cost. So the problem of dynamic environments is not tackled entirely by AgFlow [13].

There are many differences between AgFlow and our approach. The main difference, however, is that our selection strategy is totally decentralized and uses the shared knowledge created by all clients of the same service, through the pheromone deposition. Another key difference is that we consider the environment dynamics as a natural part of the problem explicitly tackling. We also show that our solution works in high scale environments.

Tao Yu et.al . [12] discuss a broker based architecture called QBroker. QBroker can be seen as a platform which helps to select the best services for composite service processes. It does not act as a middleware which seats between a composition engine and component services.

QBroker is a centralized solution which receives user requests to execute services and create concrete composite service instances, based on previously defined abstract process, called Process plan. The concrete service is created by selecting the best component services that can participate in the Process plan. The component service selection is done using QoS properties such as exectuion time, price and availability.

Tao Yu et.al. show two approaches to select component services. The first approach is a combinatorial model which defines the problem as a multidimension multichoice knapsack problem (MMKP). The second approach uses a graph model and solves the multiconstrained optimal path (MCOP) problem. Both approaches select the component services trying to maximize a user defined utility function based on the quality attributes. An interesting consideration is that QBroker guarantees end-to-end QoS properties what is a desired property of composition systems.

Our approach to service adaptation differs both in the technique and in the solution architecture. Our approach assumes that a service environment changes too much to be possible to execute on composition without reassessing the quality of component services. QBroker algorithms may create optimal compositions, but does not provide a solution to cope with dynamic changes in the environment as our solution does. Another difference is that we do not require a central broker which knows about the quality of service of all the participating services. A similarity is that QBroker also takes QoS attributes into account when performing the service selection operations.

Another work which tackles the service composition based on QoS attributes problem is from Mabrouk et. al. [7]. Their selection algorithm is integrated in the SemEUsE middleware architecture and is centered on dynamic binding of services. Many components are described in the SemEUse middleware architecture, which has three distinct layers: Service Layer, with components responsible for message routing over the network, communication protocol mapping and certain security aspects; Semantic Layer, which includes a semantic registry a composition framework and a monitoring service.

The SemEUse selection algorithm aims at determining a set of compositions that: respect global QoS constraints and maximize a QoS utility function, defined by the user. They model QoS attributes, such as response time, availability, reliability, throughput between others. These attributes are gathered by the monitoring component and are used for dynamic evaluation of services.

SemEUse selection algorithm uses a clustering technique approach to produce a set of possible compositions that respect the defined global QoS constraints. Due to the NP-hard nature of the selection problem, Nebil et. al. use a centralized heuristic approach to find the best near-optimal compositions which are then monitored at runtime to check if the selected composition is still good enough, according to the QoS criteria.

The approach discussed in Mabrouk et. al. [7] differs from our approach in the sense that they do not use the aggregated information created by different compositions. In our approach we use the distributed QoS information in order to select the best available services. Another key difference is that we do not have a separate monitoring component, since

in our approach the monitoring and composition are part of the same component, which is the coordination mechanism.

Shen and Yuan [11] provide a solution to the selection of component services based on QoS attributes that uses an Ant Colony Optimization (ACO) technique. Their selection approach is based on collecting QoS information from a number of component services, called peers in their approach, and executing a ACO algorithm to select the best services for one specific composition.

In their approach, Shen and Yuan [11] create an artificial graph with the available services represented as nodes in the graph. Each service is grouped, according to the operations it offers, what is called an atomic service. Virtual ants can then walk in this graph, from node to node, searching for the best available path which contains all the activities defined in the composite service. The ants follow concentrations of pheromones, which are indications of the quality of the path, to find the best solutions for the composition.

A interesting aspect of this solution is that it takes the environment dynamics into account, by storing the pheromone information in the graph nodes. That way, if one component services fails, the composition engine does not need to start searching new services from the scratch because it already has information in the form of dropped pheromones in other nodes from the graph. What is not clear from this solution is how the monitoring infrastructure could work, and how it could send the event to the selection algorithm.

The proposed solution shares many concepts with our own solution in the sense that both solutions are inspired by nature, more specifically ant foraging behaviour. The main difference, however, instead of focusing on the solution from the point of view of one ant we focus on the aggregated information provided by the execution of many compositions. Another difference is that we provide first class concepts to represent the service composition, by the use of Task Agents, and our approach doesn't rely on a global pheromone update function.

Finally, Ghezzi et. al. [5] compares different rebinding approaches for service composition, called binding in-the-many. This work provides a formalization to the service selection problem and identify the main solution types to be found in the literature.

The approaches studied by Ghezzi et. al. [5] are: Minimum Strategy, which is the simplest selection approach, in which each client selects the component service, called service provider, who offers the best quality criteria for the required activity; Collaborative Strategy, clients are divided in classes according to a Near-To relation. Then the clients directly share efficiency estimators with other clients from the same service class in order to do better selection of service providers; Proxy-based approach, which can be seen as a decoupling layer between the services and the concrete services.

Ghezzi et. al. simulates possible outcomes of the different approaches for the selection strategies. From the analyses, it is possible to see that the Proxy-based approaches outperform the other selection approaches.

Our work shares properties to two approaches described by Ghezzi et. al. [5], which are: our solution is offered as a middleware which maintains a view of the overlay network of services that the composition uses, what resembles the Proxy-based approach; another property is that our approach also has properties from the Collaborative approach, but without the direct communication between the clients.

3. TECHNIQUE

As we showed in the related work section, there are several techniques to select component services by evaluating quality of service attributes of each service and also the desired end-to-end quality of service. Our approach for service selection is based on concepts from ACO [4].

3.1 Ant Colony Optimization

ACO stems inspiration from real ants, more specifically from ant colony foraging, where ants cooperate in order to find the best paths to the best sources of food [4]. Ants from the same nest need to find the best sources of food. The quality of a food source is defined by the distance to the nest and to the energy that can be obtained from a specific type of food. So ants need to to a form of evaluation in order to decide if one food source is good or not.

However there is no single ant in charge to evaluate what is the best available food source. Despite this, ant colonies are capable of finding the best food sources for them, which are at the same time close to the nest and of good quality (high sugar concentration). Recent studies revealed that ants also have good solution skills to solve problems in dynamic environments, with the inclusion and removal of paths to the food source [9].

According to research studies [10, 9] real ants do not have any form of direct communication between them, but they do have a way to communicate with other ants. They communicate by dropping pheromones, which are chemical substances, in the environment [1]. Other ants take actions based on the quality and intensity of the pheromones they smell. What is an indirect form of communication mediated by the environment [9].

In ACO algorithms a problem is modeled as a graph in which artificial ants search the best paths. The definition of the best path is domain dependent, it can be the shortest path, the cheapest path, etc. An artificial ant is a software entity that has a stochastic decision policy to move between adjacent nodes in the problem graph. The policies used by artificial ants make use only of local information available in the environment and of ant's internal state [4].

The local information in the environment is created by each artificial ant. When an ant finds a solution (a path) it drops information, called artificial pheromones, in the environment. Other ants can then use this information to find their own solutions. No ant has a complete view of all the solutions at any time. The best solution to the problem at hand emerges by the continuous exploration done by different ants.

Another crucial aspect of ACO algorithms is the idea of pheromone evaporation. Pheromone evaporation allows the ant colony to forget past solutions, otherwise the pheromone level in the environment will be to elevated to allow ants to explore new solutions. The pheromone evaporation is a mechanism in which the pheromone level is linearly decrease as the time pass by.

3.2 Service Foraging

We used ACO concepts in our solution to the problem of creating QoS oriented service compositions. In our model, the services can be seen as the available food sources for ant colonies. Each service has a certain quality, described in terms of the quality model from Section 2.2.

We model four types of entities: (i) **OrgAgents**; (ii) **ResourceAgent**, and two types of ants: (i) **ExplorationAnts**; (ii) **IntentionAnts**.

We create an overlay network based on the relations between services. A ResourceAgent represents its service and has references to other ResourceAgents in the overlay network. ResourceAgents are also responsible for storing the QoS information of the services they represent.

OrgAgents represent the composite service in the overlay network. They are responsible for given the best set of services to their composite service. OrgAgents are responsible for selecting the services that will participate in a composition and are also responsible for maintaining the end-to-end QoS of the composite service they represent.

OrgAgents send a number of ExplorationAnts to search for services in the environment. These ants crawl the environment, overlay network, looking for services needed by the OrgAgent. We call this search **Service Foraging**. The ExplorationAnts check the component services QoS, asking this information to the ResourceAgents, and evaluate the quality of the component service using a heuristic defined by the OrgAgent.

Each ExplorationAnt has a survival time. If the ExplorationAnt is not capable of finding a solution, the ExplorationAnt simply stops its own execution after the survival time.

The ExplorationAnts return the gathered information to the OrgAgent, which, in turn, evaluates the solutions brought by the ExplorationAnts selecting one of them. Once the OrgAgent decides to engage in a particular composition path, it sends out IntentionAnts to walk along this path.

The IntentionAnts drop pheromones along the composition path, indicating the OrgAgent intention to use the services along the path. So, in our model, the pheromone levels of a composition path are increased just when **Intention Ants** drop pheromones in that path.

We also modeled the pheromone evaporation, through a pheromone evaporation mechanism. This mechanism modifies the pheromone levels in the trails allowing the OrgAgents to slowly forget past solutions, so that it can search for newer solutions that can be even better than the current ones. Thus we model the pheromone evaporation according to the passing of time.

The entities interact through the deposition of pheromones in the virtual environment, provided by the middleware, and never by direct communication.

Listing 1: PheromoneInfrastrucuture Interface

```
public interface PheromoneInfrastructure {
 void drop(Pheromone p);
 List<PheromoneLevels> smell(Pheromone p);
 ResourceAgent getAgent();
}
```

3.3 CASAS Middleware

In order to implement the selection algorithms described in the above section, we extended our previous work on the CASAS middleware [2]. This middleware provides tools to abstract the coordination layer from the message inspection and service binding.

For each composite service instance created in the composition engine, an analogous OrgAgent is created in the coordination layer. The OrgAgent is loaded with the set of activities described in the composite service. That way, the OrgAgent knows with types of services it needs to find in the network.

The CASAS loader creates the virtual graph where the virtual ants do their searches based on configuration given by the application developer. This configuration is basically the set of contracts between application user and the service providers that will participate in the service compositions. We follow this approach since the overlay network creation is out of the scope of this paper.

Our middleware is located between the composition engine and the services middleware that does the invocation of the component services. As illustrated in Figure: 2.

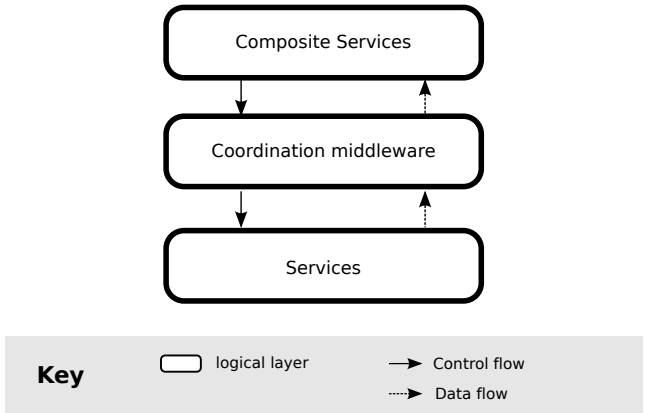

Figure 2: Middleware layers.

With the middleware approach we can use the aggregated information created by all the composite services that are executing concurrently in the composition engines. Because, as explained above, for each composition instance, there is one OrgAgent acting in the overlay network. The aggregated

information is disseminated by the Intention Ants sent by each OrgAgent, as explained in Subsection 3.2.

4. EVALUATION

We performed a number of simulations to evaluate the service selection capabilities of our approach. Each simulation was performed 20 times and the average of each simulation result was used in the evaluation, in order to diminish any simulation bias.

In order to evaluate our ACO based algorithms offered by our middleware we created a scenario based on commonly used scenarios found in the literature [13][5][11]. The first scenario has one instance of a composite service with five sequential activities. The quality criteria to evaluate in this scenario is the end-to-end QoS of the composite service, specifically focusing on the response time.

We simulated an overlay network constituted by 302 services, 100 services services capable of performing operation 1, 100 services capable of performing operation 2, 100 services capable of performing operation 3, and 2 services capable of performing operation 4. Each service has a particular mean response time, which was created according to the Normal distribution, with values in the range of (55.00, 165.00) for the services of type 1, (100.00, 400.00) for the services of type 2, (40.00, 120.00) for services of type 3, and finally (100.00,400.00) for services of type 4.

We used the Poisson probability density function to model the response time of each component service. The poisson probability was used because of the discrete nature of the events generated. We used the libraries from the Apache commons-Math project [1], specifically the Mersenne twister as the random number generator, to avoid any errors in the generation of the simulation events.

For this experiment we created just one instance of the composite service, what means that there were just one instance of the OrgAgent. The OrgAgent goal function was to minimize the response time of the composite service, that is, to minimize the response time of the sum of all activities needed by the composite service. The OrgAgent was configured to change the current composition, just if the new service path, brought by the exploration ants was 40 % better then the current service path. So, the OrgAgent could avoid instability in the composition.

Figure 3 indicates that the most part of the created compositions had a response time between (450.00, 500.00) ms. The mean response time of the composite service was 475.5 ms with standard deviation of 22.50 ms. These results indicated that the OrgAgent could indeed find good solutions, good component services, to participate in the composition.

We added a failure probability of 0.10 to services of type 1, 2, 4 to investigate our solution's resilience to failure. So, at every execution each service can fail with a probability of 0.10, but instead of simply failing we assume the services will have a worse average response times 2 times greater then the mean response time for that service.

[1]http://commons.apache.org/math/

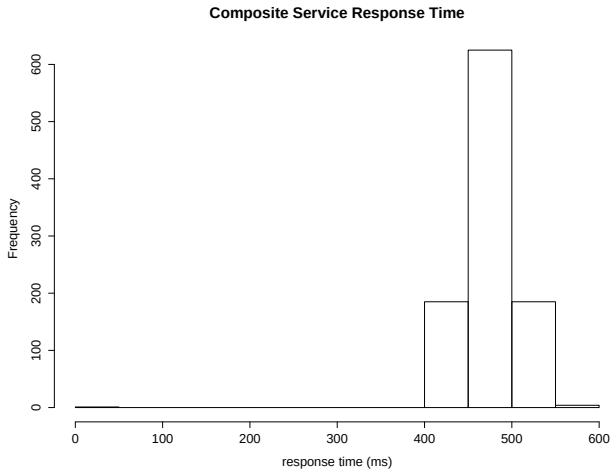

Figure 3: Mean response time of created compositions in a static scenario.

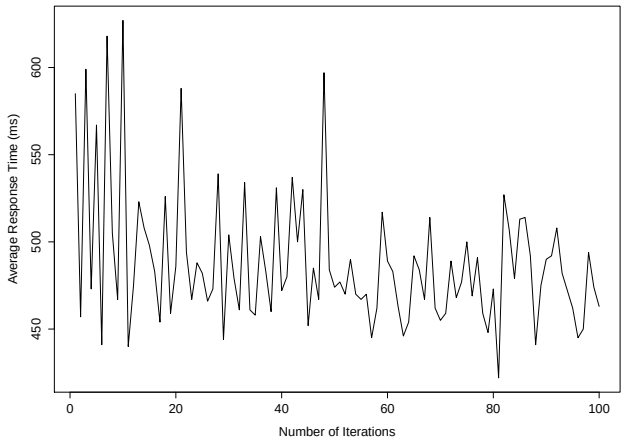

Figure 4: 0.10 failure probability for each component service.

The obtained results are depicted in Figure 4, which shows that the composition times were greater in the first iterations but tended to lower response times after a few iterations of the selection algorithm. It is possible to see that the OrgAgent could create compositions, even in the presence of dynamic changes in the environment. The mean response time was 477.03 ms and the standard deviation 25.45 ms.

In order to evaluate the benefits of using the aggregated information from more then one composition, we added 3 OrgAgents in the overlay network.

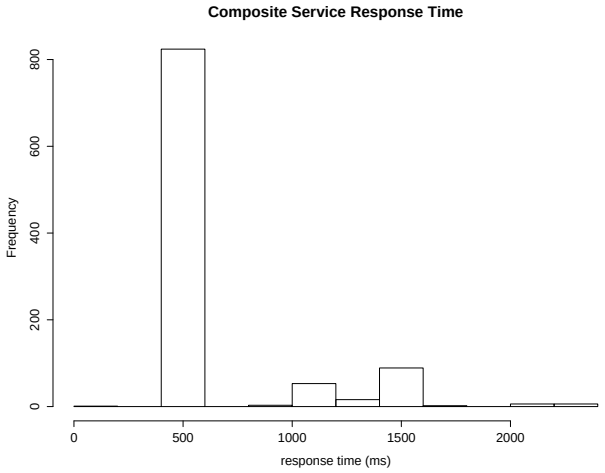

Figure 5: 3 OrgAgents working on the overlay network with 0.10 failure probability for each component service.

It is possible to see, in Figure 5, that most part of the created compositions had a mean response time between (400.00, 600.00) ms. It is possible to see that more different service compositions were explored due the having more OrgAgents in the network, what can be seen by the compositions with response time greater then 1000.00 ms.

5. CONCLUSION AND FUTURE WORK

The main contribution of this work is to show a service selection mechanism that is decentralized, scalable, and robust. We evaluated our solution in a number of test scenarios and showed that it is possible to have interesting results by using the aggregated information created by the different composite services that use the middleware.

Our test scenarios were created corresponding to the scenarios found in the related literature but we still want to assess the validity of our approach in real life scenarios, what will be done in future research.

Another future research is on the creation of the overlay network used by the selection algorithms. We believe it is possible to create this overlay network using a peer-to-peer approach to bootstrap the network and then use the pheromone metaphor and ACO algorithms to find and select services in this overlay network.

6. ACKNOWLEDGMENTS

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven.

7. REFERENCES

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

[2] M. H. Cruz Torres, V. Noël, T. Holvoet, and J.-P. Arcangeli. Mas organisations to adapt your composite service. In *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond*, MONA '10, pages 33–39, New York, NY, USA, 2010. ACM.

[3] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15(3-4):313–341, Dec. 2008.

[4] M. Dorigo, G. D. Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999.

[5] C. Ghezzi, A. Motta, V. Panzica La Manna, and G. Tamburrelli. Qos driven dynamic binding in-the-many. In G. Heineman, J. Kofron, and F. Plasil, editors, *Research into Practice – Reality and Gaps*, volume 6093 of *Lecture Notes in Computer Science*, pages 68–83. Springer Berlin / Heidelberg, 2010.

[6] T. Holvoet, D. Weyns, and P. Valckenaers. Patterns of delegate mas. *Self-Adaptive and Self-Organizing Systems, International Conference on*, 0:1–9, 2009.

[7] N. B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny. QoS-Aware service composition in dynamic service oriented environments. In *Middleware 2009*, volume 5896 of *Lecture Notes in Computer Science*, pages 123–142. Springer Berlin / Heidelberg, 2009.

[8] M. P. Papazoglou. *Web Services: Principles and Technology*. Pearson, Prentice Hall, 2008.

[9] C. R. Reid, D. J. T. Sumpter, and M. Beekman. Optimisation in a natural system: Argentine ants solve the towers of hanoi. *J Exp Biol*, 214(1):50–58, 2011.

[10] S. K. Robson and J. F. A. Traniello. Resource assessment, recruitment behavior, and organization of cooperative prey retrieval in the ant formica schaufussi (hymenoptera: Formicidae). *Journal of Insect Behavior*, 11:1–22, 1998. 10.1023/A:1020859531179.

[11] J. Shen and S. Yuan. Qos-aware peer services selection using ant colony optimisation. In W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, W. Abramowicz, and D. Flejter, editors, *Business Information Systems Workshops*, volume 37 of *Lecture Notes in Business Information Processing*, pages 362–374. Springer Berlin Heidelberg, 2009.

[12] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. Web*, 1(1):6, 2007.

[13] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.