***Ranking Strategies for Quality-Aware Service Selection***

Fábio N. Souza, David J. M. Cavalcanti, Tarcisio C. Silva, Nelson S. Rosa

Centre of Informatics

Federal University of Pernambuco

Pernambuco, Brazil

{fns, djmc, tcs5, nsr}@cin.ufpe.br

*Abstract*—Service oriented computing enables the development of applications as compositions of basic entities called services. These services offer business functions, which are used as primary criteria in the service selection algorithms. In distributed scenarios, a large number of services can offer similar functionalities, motivating the embracement of quality attributes as fundamental selection elements. Different strategies can be used to classify services based on these attributes and it is quite difficult to elect one that seems adequate in every scenario. In this context, we present DSOA platform, which is an environment that supports dynamic service compositions based on quality attributes. Particularly, our focus in this paper is in its Service Selection component which supports the definition of new selection strategies at runtime. In order to validate our solution, we perform an experimental evaluation that shows the utilization of different service selection strategies and their performance impacts and effectiveness in selecting services.

Keywords—service selection; quality attributes; ranking algorithms;

# Introduction

Service-oriented computing (SOC) enables us to develop applications by composing entities called services. Such services are selected and integrated into the application prior to or during application execution. In this context, a key aspect is how to select the desired services. In general, service selection is based on functionality. The service consumer expresses her/his functional needs and accesses a registry requesting one or more services that implement them. The service registry returns a collection of services satisfying the requirements. Finally, the consumer is able to select, bind to and invoke one of them.

In distributed scenarios, it is common to see a large number of services implementing the same functionality. In these cases, consumers should use additional criteria to select one of them. To differentiate these services (and corresponding providers) consumers usually adopt the so called non-functional requirements (or quality attributes), such as availability, reliability, response time and cost. Still, selecting an adequate service is not an easy task; it usually involves using some mechanism to monitor the quality attributes announced by the service providers and implementing a raking algorithm to classify them. Usually, two key factors usually impact the way the services are ranked: the use of multiple quality attributes and the need of distinguishing mandatory and optional ones.

Several strategies have been used to rank services based on their functional and non-functional characteristics [3]. Each strategy usually adopts a specific ranking algorithm to determine the most appropriate service to the consumer. However, the use of a single algorithm may not always be interesting. In fact, while some algorithms are more precise, others are faster. Some algorithms differentiate between mandatory and optional quality attributes, others do not. Some algorithms allow the service consumer to specify weights to rank the candidates, while in others the consumer establishes a goal function. Hence, the possibility choosing the ranking algorithm and eventually changing it at runtime to better fit the application needs is certainly an interesting feature.

In this context, our main contribution is a new service selection mechanism, which supports various ranking algorithms and enables dynamically replacing them without interrupting running applications. Our solution is dynamic and can be activated/deactivated anytime without affecting the execution of other services. It is also extensible in the sense that it enables including new ranking algorithms at runtime. We also say that our solution is automatic since it does not require human intervention during the service selection process. Finally, our solution allows consumers to express their preferences, which are used during the selection process. Our selection mechanism is part of DSOA (Dynamic Service Oriented Architecture), an open source platform that is completely defined around the service concepts.

This paper extends our previous one by presenting more details concerning the selection process and implementing two new ranking algorithms. Besides that, we improved our experimental evaluation.

The rest of the paper is organized as follows: Section 2 introduces the basic concepts that are required for a better understanding of the proposed work. In Section 3, we present our quality meta-model. Then, in Section 4 we detail our service selection mechanism presenting its workflow and internal components. Next, Section 5 presents a performance evaluation. Related works are shown in Section 6. Finally, in Section 7 the conclusion of this paper is presented and some lines of future work are discussed.

# basic concepts

Prior to presenting the proposed solution, this section introduces some basic concepts related to quality attributes, service selection and the OSGi platform.

## Quality Attributes

While functional requirements define what the software is expected to do, non-functional requirements (quality attributes) specify global constraints on how it operates or how its functionality is exhibited. Quality attributes have a very distinctive nature, comprising a wide variety of aspects such as performance, cost and fault-tolerance.

Considering the broad scope of that deﬁnition, different schemes have been proposed to group and classify these attributes. A natural classiﬁcation that can be easily considered is grouping these attributes into two categories. The ﬁrst one comprises the deterministic attributes which are those whose values are known before application execution (e.g. price and transactional properties). The other category, referred to as non-deterministic attributes, comprises those attributes whose values observed at runtime include some uncertain degree. This category clearly includes performance related characteristics, such as availability and response time. It is important to mention that this research paper focuses on performance related quality attributes.

## Service Discovery and Selection

Service discovery and selection are closely related activities. In fact, service discovery comprises the process of finding services that meet consumers' requirements. Meanwhile, service selection is the process that deals with the choice of a service instance that has been previously discovered. In this context, service discovery is a prerequisite to the selection process [4]. In general, any service selection solution has some basic input parameters: the consumers’ requirements (e.g., functional characteristics and desired quality attributes); the set of available services (that were previously discovered); and a ranking algorithm, that is used in order to classify the discovered services.

## OSGi Platform

Open Services Gateway Initiative (OSGi) consists of a platform that provides support to the development of modular Java applications. A module in OSGi is named bundle and each bundle is a JAR (Java Archive) file that contains executable code and other resources such as native libraries. In practice, the OSGi specification defines a framework that leverages Java’s dynamic class loading feature to enhance modularization and also introduces a runtime where bundles can be installed, started, stopped, updated or uninstalled without stopping the application . The platform contains an execution environment where bundles can provide and request services which are maintained in a Service Registry. Therefore, the OSGi platform was denominated a form of Service Oriented Architecture (SOA) .

In fact, in its core specification, the OSGi platform defines a local service registry, where the bundles publish or consume services hosted in the same JVM. Nevertheless, the OSGi specification introduces a distribution mechanism, which is an optional component that allows the construction of a distributed environment composed of several OSGi runtime environments, where components can provide and request remote services.

# DSOA Quality Model

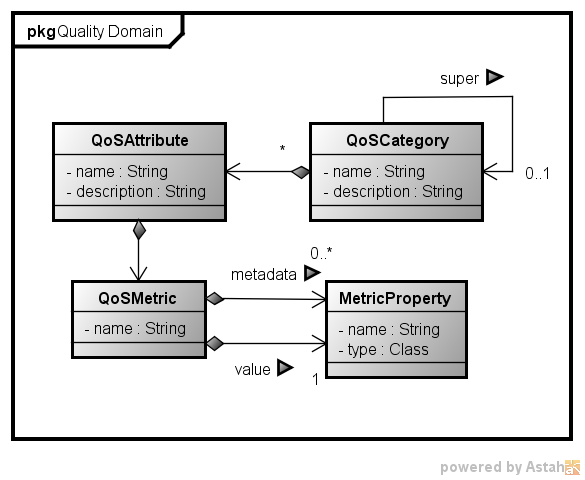
Despite the relevance of quality attributes to the service-oriented computing area and the attention that this topic has received from the academy and industry, there is not an universally accepted QoS-model [9]. In fact, QoS is a broad area, and it is not possible to establish an exhaustive list describing all relevant QoS attributes [14]. Moreover, each attribute can be evaluated through different metrics, each one summarizing a different point of view, such as the average and minimal values.

Since it is not feasible to propose a uniﬁed quality model, it is necessary to enable each application to deﬁne its own model, representing the quality attributes and metrics that it considers relevant. In SOA platforms, quality attributes are usually associated to services and their operations. As already mentioned, these attributes play a very relevant role and are frequently used by the underlying infrastructure to manage services and SBAs lifecycle.

This section details how quality and service domains are represented in DSOA platform using

## Quality Metamodel

In DSOA platform, a quality model is defined using its quality meta-model, which includes constructors enabling the specification of new quality attributes and metrics. Fig. 1 presents a simplified overview of this meta-model.



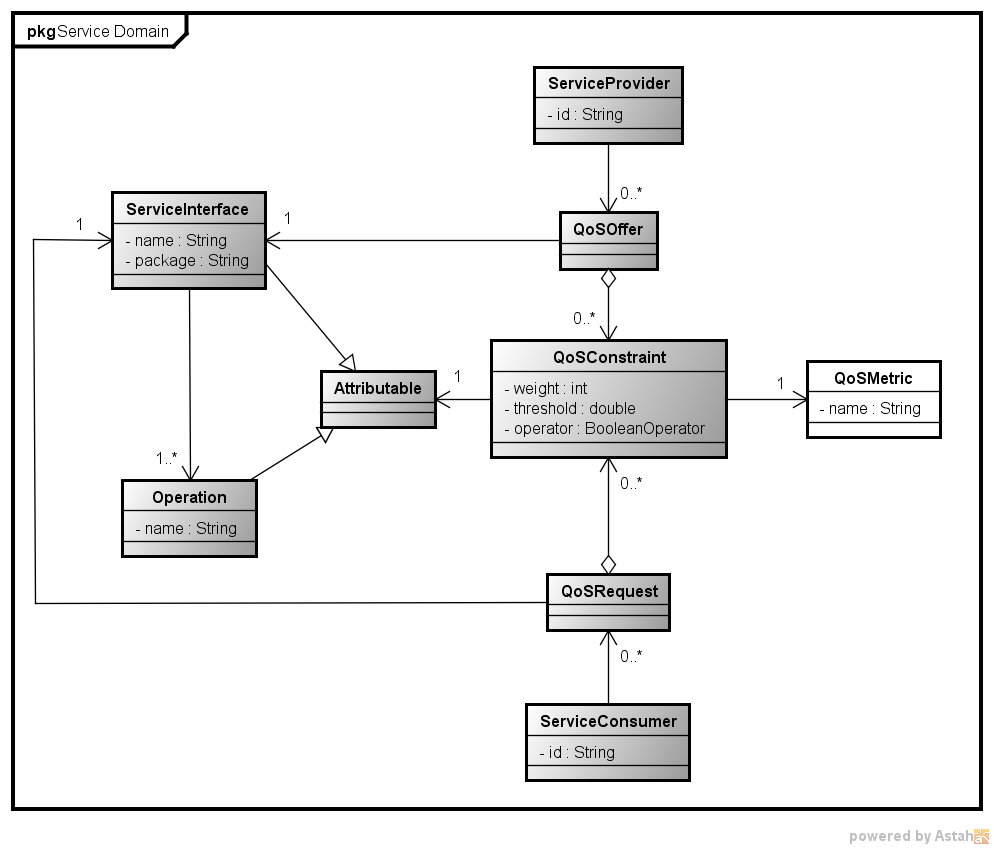
1. Overview of DSOA's QoS Meta-model

As we can observe, QoS attributes are grouped into QoS categories, which are organized in a tree like structure. Each attribute is characterized through a set of QoS metrics, which are defined in terms of a collection of metadata and data properties. In this context, metadata properties are supposed to be used to characterize the circumstances under which the QoS data was collected (e.g. timestamp), while data properties are used to contain the value computed for the metric.

Using DSOA meta-model, a quality expert can build a common quality model, which describes the QoS attributes and metrics adopted to specify the quality level provided by the services and required by the applications that consume them. An important aspect concerning this quality model is that it is agnostic about how the quality metrics are gathered or computed. In fact, metric computation is important aspect deal with by DSOA platform, however it is out of the scope of the present paper.

## Service Quality Offer and Request

In DSOA platform, in order to announce a service, a provider should build a *service quality offer* (see Fig. 2)specifying the service functional interface, and a set of quality constraints, which characterize its quality level. Each constraint defines a threshold to a quality metric that is part of the quality model.



1. Service Quality Offer and Request

On the other hand, when a consumer wants to find an adequate service, it builds a *service quality request* also stating a required service interface and a set of quality constraints. Such requests are forwarded to DSOA service selection mechanism, which uses a QoS-aware service registry in order to locate candidate services.

# Quality-Aware Service Selection Mechanism

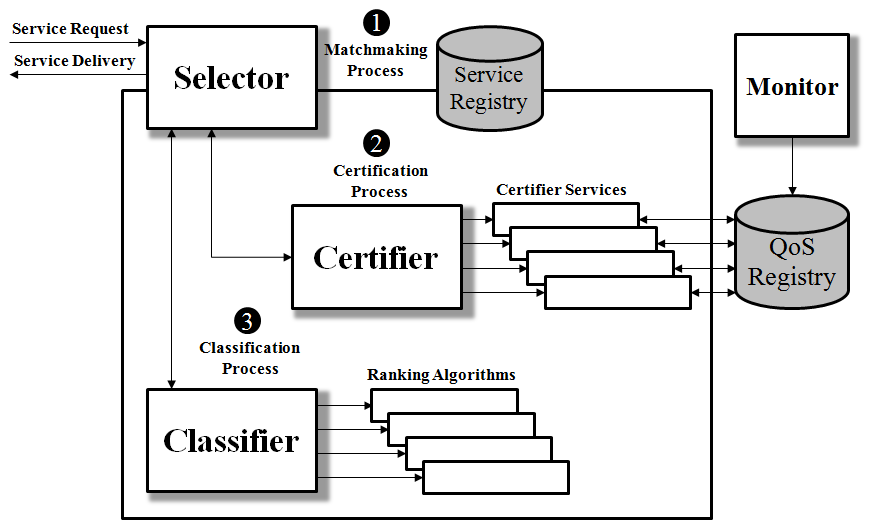
DSOA platform introduces an extensible quality-aware service selection mechanism, which can be reconfigured at runtime. In particular, the reconfiguration process allows a complete redefinition of the selection strategy, enabling us to adopt one that seems appropriate to each situation.

The selection mechanism comprises three core components: *Selector*, *Certifier*, and *Classifier*, each one responsible for a major step in the quality-aware selection process. These components collaborate using a common quality model built atop of DSOA quality meta-model. Fig. 3 presents an overview of the selection process highlighting these components.

## The Selector Component

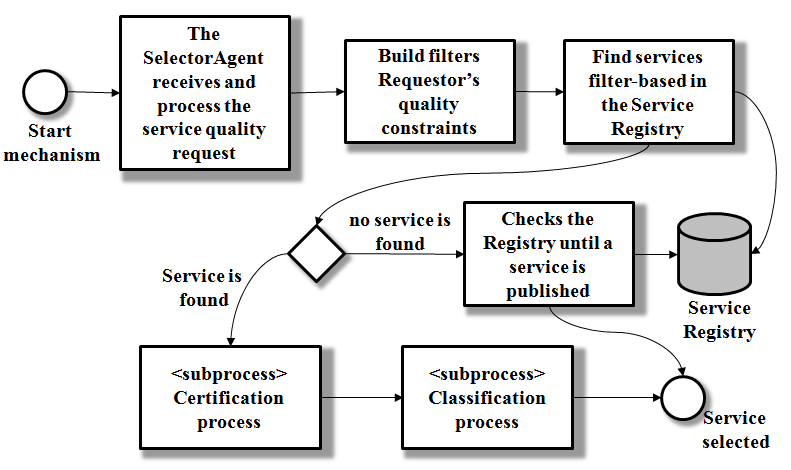
The selector component represents a Facade to the Selection mechanism embedded in DSOA Platform. Upon receiving a *service quality request*, the selector component initiates a *matchmaking process* that is represented in Fig. 4.

The *SelectorAgent* is the element of the *Selector Component* that receives and process the service quality requests, and extracts the required service interface and quality attributes and metrics.



1. Service selection mechanism architecture

The collected data are forwarded to the *Filter Builder* who’s responsible for building filters based on the specifications sent by the consumer. These filters are simple expressions with syntax similar to LDAP. They contain the service interface and the required quality attributes and metrics:



1. Selection process

(&(service=*interface.service*) (service.operation.AvgResponseTime<=*values*))

The *SelectorAgent* forwards this filter to DSOA's Quality-aware Service Registry that uses it to search for a collection of services that satisfy the request. In this context, a service is considered a proper candidate if it satisfies the functional requirements (stated through the service interface) and the quality attributes informed by the consumer.

Whether there are no suitable services, the *Service Tracker* component is activated. It saves the filter previously built and registers itself in the *Service Registry* in order to receive notifications when new services are available.

Once that the tracker is notified, it verifies whether the service satisfies its filter. If this is the case, it uses a callback mechanism to inform the consumer about it.

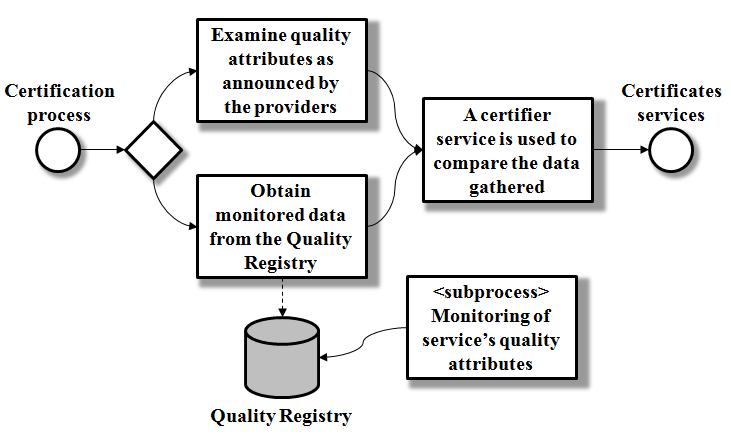
In the case that more than one service is found, the *SelectorAgent* can use the *Certifier* and *Classifier* components, to select one of them. Finally, the *SelectorAgent* is responsible for delivering the service to the Consumer.

## The Certifier Component

The SOA's distributed nature usually impacts service's quality attributes. As a consequence, the quality level effectively provided may not be in accordance with what is announced. For that reason, we established a *Certification process* (see Fig. 5), which is responsible for checking whether candidate services meet the consumer's quality requirements or not. For example, a provider may inform that the service has 95% of availability, but due to environment conditions, its actual availability could be 80%.

In these situations, using the quality data as published by the provider can lead to a less than optimal choice from the consumer's point of view. To deal with this possibility, DSOA platform monitors the quality of the services that are used from the applications that it supports. Part of our monitoring solution was already published in [5]. Monitoring components store quality data in a *quality registry*, which is used during the certification process as presented in Fig. 5.

Another important characteristic of our certifier component is that it is dynamically extensible; this means that the certification process is not hard coded. In fact, the core of the certification component is encapsulated in a certification service. The certifier component already embeds some alternative certification services, but it also supports the development of new ones, as the certification service interface is part of an API that can be used to support platform extensibility.



1. Certification process

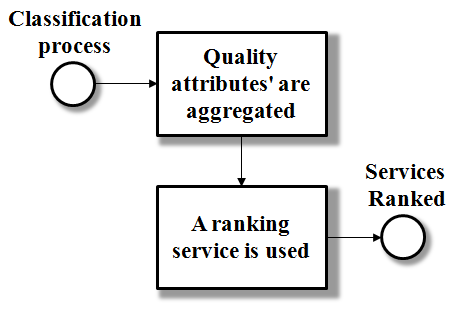
To summarize, when the *SelectorAgent* uses the certifier operation, a *certifier service* is used to perform quality certification, which encompasses the activities required to use monitored data in order to validate what is announced through *service quality offers*. For instance, while a *certifier service* might use the service's most recent monitored values, another one might assign weights to the monitored data according with their timestamps. In our platform, we have the possibility to choose different certification service to be used in order to obtain the quality values. The default *Certifier Service* considers the service's quality values offered by the provider and compares it with the quality attribute values demanded by the *Consumer*. If these data match, the service is certified as a candidate service.

After selecting the Certifier Service, it is used and the customized quality attribute values are obtained as aforementioned. The Selector uses these values to classify the services and select what it considers to be the best one.

## The Classifier Component

The aim of the classification process is to determine which service best fits the quality attributes demanded by a Consumer. A ranking is a quantitative metric that shows the relative importance of a service in the context of the current selection and defines which service must be selected. To classify different services and achieve better ranking performance many ranking algorithms have been proposed in the literature [11]. However, there is not an absolute optimal ranking algorithm capable of treating all factors and situations aforementioned. For this reason, DSOA platform enables the use of several ranking algorithms and allows changing them at runtime. For example, one algorithm may select the most similar service based on the service quality request. Fig. 6 shows the *Classification process*.

In our mechanism, each ranking algorithm is also a service, which can be started and stopped at any time without affecting the runtime environment.



1. Classification process

So, assuming that there is a collection of services that satisfies both the functional requirements and quality constraints specified by the Consumer, and that those services had they quality metrics certified by the platform (see Section B), it is necessary to determine the best service, that is, the one that should be used by the consumer.

To choose the "best" service, a ranking algorithm must be used. Each algorithm proposes a different way to compute a unique number that is associated with each service. This number allows the ranking algorithm to order the service collection and identify the best one.

Several aggregation methods can be used and each one yields different results. Our selection mechanism embeds the following ranking strategies:

### Simple Additive Weighting (SAW):

Simple Additive Weighting (SAW) is a simple multi attribute decision technique. It uses basic multiplication and addition operations to compute a weighted average (score) over the collection of quality metrics that qualifies each service candidate. The weights that are used in the algorithm are directly assigned by the service consumer. SAW aggregates quality metrics as follows:

Where,

*s* is the service,

*Qi* is the value of *i*th QoS metric,

*Wi* is the weight associated to the *i*th QoS metric that has been defined by the Consumer,

*n* is the number of QoS metrics associated to the service.

The *Score* is calculated for each candidate and the one with the highest score is selected and sent to the *Consumer.*

### Euclidean Distance:

The Euclidean Distance is a quantitative metric used to compare the similarity between two lists of objects (e.g. vectors). Therefore, we use the Euclidean Distance to evaluate the similarity between quality of services and quality required by the consumer. The Distance Euclidean aggregates data as follows:



Where,

*s* is the service,

*ϕi* is the value of *i*th quality metric,

*Ri* is the value associated to the *i*th quality metric that has been defined by the Consumer,

*n* is the number of quality metric associated to the service.

The Distance is calculated for each candidate and the one with the lower distance is selected and sent to the *Consumer.*

### Utility Function:

Our Utility Function is very similar to SAW. It aims to select individual services that meet quality constraints and contains the best values defined. However, it differs from SAW because uses the statistical concepts of mean and standard deviation. The Utility Function aggregates data as follows:



Where,

*s* is the service,

*ϕi* is the value of *i*th quality metric,

*µi* is the mean of *i*th quality metric,

*σi* is the standard deviation of *i*th quality metric,

*Wi* is the weight associated to the *i*th quality metric that has been defined by the Consumer,

*α* + *β* is the number of quality metrics associated to the service.

*α* is quality metrics to be maximized (e.g. Availability),

*β* is quality metrics to be minimized (e.g. Response Time),

The Score is calculated for each candidate and the one with the highest score is selected and sent to the Consumer.

### Entropy Function:

Entropy is a method of determining the weights that are assigned to the criteria used in a decision process. In service selection context, each quality attribute is viewed as selection criterion. In this scenario, the idea is to use an entropy function in order to determine the weight of each attribute in the service selection process. After obtaining the weights for each quality attribute, they are added together to get the service to be selected. The Entropy Function aggregates data as follows:









Where,

*s* is the service,

*j* is the quality metric,

*ϕij* is the value of *i*th quality metric,

*k* is the constant value,

*Ej* is the entropy associated to the *i*th quality metric that has been defined by the Consumer,

*Dj* is the dispension associated to the *i*th quality metrics,

*Wj* is the weight calculated for each quality metrics.

The Score is calculated for each candidate and the one with the highest score is selected and sent to the Consumer.

In summary, the architecture of DSOA service selection mechanism is quite flexible and adaptable, since it allows deploying and using several selection strategies. Moreover, it enables dynamically changing the current strategy, through a reconfiguration process that occurs at runtime. New strategies may also be implemented by defining alternative ways of evaluating the quality attributes monitored. Our solution was implemented in Java, atop of OSGi/DOSGi technologies. In fact, DSOA is built as collection of components that require and provide services to each other. In this context, all dynamic components, such as classification strategies, are implemented as OSGi services that can be started and stopped indivudually at any time.

# Experimental Evaluation and Results

To evaluate the proposed solution and the ranking strategies, we made some experiments in the same scenario presented in [1]; which consists of a *Consumer* that is responsible for purchasing shares at lower prices and a set of services responsible for obtaining information about share prices. The consumer is build atop of DSOA platform, which is responsible for selecting a price share service based on its expected quality attributes.

The evaluation was performed on a PC Intel Core i7 at 2.10 GHz with 8GB of RAM running Windows 7 Professional of 64 Bits with Service Pack 1. Java with JRE version 1.7.0\_07 was used to deploy the Apache Felix OSGi 4.0.3 environment.

The experiment has two goals: evaluating the effectiveness of the proposed ranking strategies and evaluating the selection mechanism as a whole. To do that, we compare the execution of the consumer application supported by DSOA platform with and without its dynamic components enabled.

Our first study focused on evaluating the effectiveness of the proposed ranking strategies. In this case, it is necessary to analyze which services are selected by the ranking strategies. For such assessment, the workload parameters that were used are the number of candidate services and number of quality attributes. We used ten candidate services and the number of quality metrics varied from 1 to 5 as adopted in [3][13][16]. Besides of the two ranking strategies (Simple Additive Weighting [3][6][8] and Euclidean Distance [7]) already presented in [1], and that are widely adopted, we used two other ranking strategies, the Utility Function presented in [16][17] and Entropy Function that we adapted for service selection (see Section IV.C). Additionally, the OSGi default selection algorithm has also been considered, in which a service is selected without considering quality aspects.

TABLE I. presents the quality attributes as required by the *Consumer*, whilst TABLE II. depicts the quality attributes of the candidate services. The quality attributes used are response time, availability, throughput, accuracy and cost. The results of the effectiveness of the selection using different ranking strategies are shown in TABLE III. When the ranking strategies are enabled in the platform, the following ones (*SimpleAdditiveWeight*, *EuclideanDistance*, *UtilityFunction* and *EntropyFunction*) are considered. When they are disabled, we use the default OSGi selection algorithm.

1. Quality attributes specified by the Consumer

|  | QoS Attributes | | | | |
| --- | --- | --- | --- | --- | --- |
| Response Time (ms) | Throughput (req/s) | Accuracy (%) | Availability (%) | Cost ($) |
| Resquest | 999 | 10 | 85 | 80 | 5.0 |

1. Candidate services and their respective quality attributes

| Services | QoS Attributes | | | | |
| --- | --- | --- | --- | --- | --- |
| Response Time (ms) | Throughput (req/s) | Accuracy (%) | Availability (%) | Cost ($) |
| Service 1 | 700 | 20 | 89 | 85 | 1 |
| Service 2 | 900 | 10 | 92 | 95 | 1.2 |
| Service 3 | 915 | 28 | 90 | 80 | 0.5 |
| Service 4 | 910 | 12 | 96 | 98 | 2.5 |
| Service 5 | 390 | 28 | 99 | 99 | 5.0 |
| Service 6 | 950 | 10 | 95 | 100 | 0.9 |
| Service 7 | 600 | 25 | 90 | 95 | 2.0 |
| Service 8 | 500 | 24 | 98 | 93 | 1.75 |
| Service 9 | 998 | 15 | 85 | 83 | 0.0 |
| Service 10 | 545 | 22 | 100 | 90 | 0.3 |

With the proposed solution disabled, a service is selected randomly and may not be the best one. This can be checked in the TABLE III. (see *Default* line). The service 3 was selected, even though has a higher response time and very low availability.

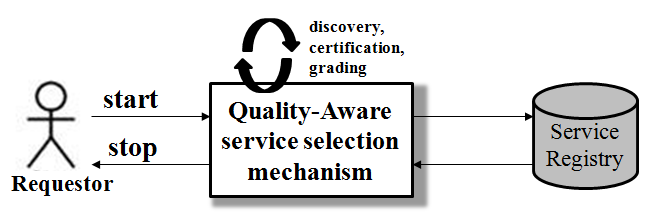
Considering the use of the service selection mechanism, we see that the *SimpleAdditiveWeight* strategy is always the best option, because it is more accurate when selecting the best service (Service 5). The Euclidean Distance strategy also proved to be efficient; it selects the most similar service based on the quality attributes required by the *Consumer* (Service 9).

The other strategies have some faults in the ranking of services. The *UtilityFunction* selected a bad candidate (Service 3). While the *EntropyFunction* cannot be applied in scenarios service selection based on just one quality attribute. But we need a more elaborated study in order to better analyze these strategies.

1. Service selected by different strategies

| Ranking Strategies | Number of Quality Attributes | | | | |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Default | Service 4 | Service 1 | Service 3 | Service 5 | Service 1 |
| SimpleAdditiveWeight | Service 5 | Service 5 | Service 5 | Service 5 | Service 5 |
| EuclideanDistance | Service 9 | Service 3 | Service 9 | Service 9 | Service 9 |
| UtilityFunction | Service 9 | Service 6 | Service 3 | Service 5 | Service 5 |
| EntropyFunction | - | Service 5 | Service 5 | Service 5 | Service 5 |

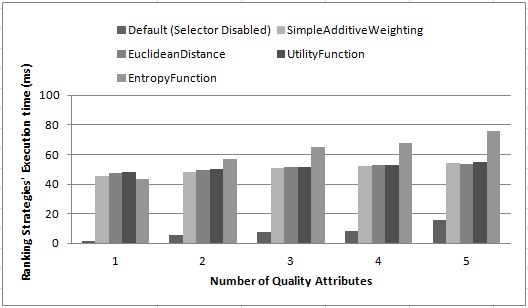
The second objective of our experiments is to evaluate the performance of the service selection mechanism. In this context, the metric considered is "*service selection time*", which represents the time elapsed between sending a service quality request and receiving a reference to the most adequate service. Fig. 7 shows when the selection time is started (when a service quality request is sent by the consumer), and when it is stopped (upon receiving a service reference from DSOA platform).



1. Overview of the Evaluation Experimental Scenário

In this performance experiment, our parameters were: ranking algorithms enabled/disabled, number of quality attributes and number of candidate services. The number of quality attributes varies from 1 to 5, whilst the number of candidate services is fixed at 500. These numbers have been defined in order to extrapolate the scenarios usually used to evaluate other solutions, which usually consider up to 3 quality attributes and 10 candidate services.

Fig. 8 shows the selection time when the service selection mechanism is enabled and disabled (default), and a comparison among different ranking strategies when the number of quality attributes range from 1 to 5. As we can see, the selection time of the default strategy is lower than other strategies. This occurs because it does not consider the quality of the candidates in the selection process, and so, a service is randomly selected. Consequently, there is no guarantee that the service with best quality attributes is selected as shown in



1. Service Selection time x Number of quality attributes results with and without service selection mechanism

Despite the minimal differences between the ranking algorithms proposed, we can observe that the introduction of the ranking process does not have a significant impact on the service selection performance (the selection time is delayed by approximately 40 ms). This means that introducing and using a good ranking strategy is very interesting, once that the platform will be able to select an adequate service candidate, without heavily impacting the application performance. For instance, spending some time in order to select the service that offers the shortest execution time will certainly be better than using one service that has been randomly selected.

Based on our experiments' results, we conclude that our platform embeds a powerful service selection mechanism enabling the use of different ranking strategies, maintaining an acceptable performance.

# Related Work

The selection of one service among thousands with similar or identical functionality is a complex task and several approaches have been proposed to solve this problem. Furthermore, the majority of solutions adopt quality attributes as key criteria for the success of the selection.

Serhani et al. present a QoS broker-based architecture that helps clients to select a Web services considering quality attributes. This solution has two-phases: syntactic and semantic verification of the service interface, including the quality attributes; and the use of test cases to obtain the actual quality attributes and compares their values with ones required by the consumer. This solution concentrates on the verification and certification process through the measurement of quality attributes using test cases. Unlike this related work, our measurement is carried at runtime considering the actual functioning of the service. Furthermore, we also allow to dynamically changing the way the services are ranked.

Reiff-Marganiec [13] proposed a method for automatic selection of the most relevant service for composition based on quality attributes and the user’s context. The proposed method divides the services into categories with similar functionality, filters them using the functionality and then ranks the services using the quality attributes. However, this work does not check whether the service actually meets the user’s quality attributes or not, i.e., it relies on the quality attributes set by the service provider. There is not a monitoring at runtime.

Rajendran [15] proposed a complete mechanism for dynamic services’ selection that makes easier for consumers to specify their quality attributes. The mechanism has a technique to verify and certify the quality attribute to determine whether the services actually meet them at runtime or not. However, they do not take into in account dynamic consumer preferences.

Finally, the service selection solutions have different ranking algorithms to determine which services should be selected. Al-Masri and Mahmound , Xu et al. and Taher et al. propose solutions that use Euclidean Distance for ranking service’s quality attributes. Meanwhile, Zeng et al. adopts the Multi Criteria Decision Making (MCDM) technique, called Simple Additive Weighting, for yielding the service ranking. However, all these works set their ranking algorithms in a static way, i.e., after defined, the ranking algorithm may not be altered at runtime.

# Conclusion and Future Work

This paper presents our service selection mechanism, which takes into account quality attributes in order to select the best candidate when functionally similar service are published in a service registry. The proposed mechanism is extensible, dynamically adaptable, and may be reconfigured at runtime without affecting the execution of other services. The mechanism was implemented in Java/OSGi, and we carried out an experimental evaluation to verify its impact on the execution of service-based applications.

The main contribution of the proposed service selection mechanism is allowing the dynamic replacement (at runtime) of the algorithms used to rank services using quality attributes. The kind of changing may be motivated by a variety of factors, including: the diversity of quality attributes, the necessity of giving priority to a particular one, changes in the consumer’s preferences, and so on.

As our next steps, we are working on an implementation of additional ranking algorithms including ideas related to context-awareness. Finally, we are planning to extend an open source orchestration engine (e.g., Apache ODE) with the proposed selection strategy.

##### References

1. D. J. M. Cavalcanti, F. N. Souza, and N. S. Rosa, “Adaptive and Dynamic Quality-Aware Service Selection.” in 21st Euromicro International Conference on Parallel, Distributed, and Network-based Processing, pp. 323-327, 2013.
2. D. Marples, and P. Kriens, “The open services gateway initiative: An introductory overview.” in Communications Magazine, IEEE, 39(12), pp. 110-114, 2001.
3. E. Al-Masri, Q. H. Mahmoud, “QoS-based Discovery and Ranking of Web Services.” IEEE 16th International Conference on Computer Communications and Networks, pp. 529-534, 2007.
4. E. M. Maximilien, and M. P. Singh, “Toward Autonomic Web Services Trust and Selection.” in Proceeding og the 2nd International Conference on Service Oriented Computing, pp. 212-221, 2004.
5. F. N. Souza, D. Lopes, K. Gama, N. S. Rosa, and R. Lima, “Dynamic Event-based Monitoring in a SOA Environment.” in 13º International Symposium on Distributed Objects and Applications, 2011.
6. H. Q. Yu, and S. Reiff-Marganiec, “Non-functional property based service selection: A survey and classification of approaches.” in Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop co-located with The 6th IEEE European Conference on Web Services, 2008.
7. L. Taher, H. El Khatib, and R. Basha, “A Framework and QoS Matchmaking Algorithm for Dynamic Web Services Selection.” In Proceedings of the 2nd International Conference on Innovations in Information Technology, 2005.
8. L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, “Quality Driven Web Services Composition.” in Proc. 12th Int’l Conf. World Wide Web, 2003.
9. L. Zeng, H. Lei, and H. Chang, “Monitoring the QoS for Web Services.” in ICSOC, ser. Lecture Notes in Computer Science, B.J.Kramer,K.-J. Lin, and P. Narasimhan, Eds., vol. 4749. Springer, 2007, pp. 132–144.
10. M. A. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui, “A QoS Broker Based Architecture for Efficient Web Services Selection.” in Proc. Of the IEEE Int’1 Conf. on Web Services, pp. 113-120, 2005.
11. M. Sathya, M. Swarnamugi, P. Dhavachelvanm, and G. Sureshkumar, “Evaluation of QoS Based Web-Service Selection Techiniques for Service Composition.” in International Journal of Software Engineering, pp. 73-90, 2011.
12. S. Neelavathi, and K. Vivekanandan, “An Innovative Quality of Service (QOS) based Service Selection for Service Orchrestration in SOA.” in International Journal of Scientific & Engineering Research, 2011.
13. S. Reiff-Marganiec, H. Q. Yu, and M. Tilly, “Service Selection based on Non-Functional Properties.” in Proceeding of Non-Functional Properties and Service Level Agreements in Service Oriented Computing Workshop, 2007.
14. S. Sentilles, “Managing Extra-Functional Properties in Component-Based Development of Embedded Systems,” Ph.D.dissertation, Malardalen University, Vasteras, Sweden, June 2012.
15. T. Rajendran, P. Balasubramanie, and R. Cherian, “An Efficient WS-QoS Based Architecture for Web Services Selection.” in International Journal of Computer Applications, 2010.
16. T. Yu, K. Lin, “Service selection algorithms for Web services with end-to-end QoS constraints.” in IEEE International Conference on e-Commerce Technology, 2004.
17. T. Yu, Y. Zhang, and K. Lin, “Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints.” in ACM Transactions on the Web, 2007.
18. Z. Xu, P. Martin, W. Powley, and F. Zulkernine, “Reputation-enhanced qos-based web services discovery.” in IEEE International Conference on Web Services, pp. 249-256, 2007.