Towards Combined Functional and Non-functional Semantic Service Discovery

Kyriakos Kritikos, Dimitris Plexousakis

Towards Combined Functional and Non-Functional Semantic Service Discovery

Kyriakos Kritikos and Dimitris Plexousakis

ICS-FORTH
Heraklion GR-70013, Greece
{kritikos, dp}@ics.forth.gr

Abstract. Service-orientation is increasingly adopted by application and service developers, leading to a plethora of services becoming increasingly available. To enable constructing applications from such services, respective service description and discovery must be supported by considering both functional and non-functional aspects as they play a significant role in the service management lifecycle. However, research in service discovery has mainly focused on one aspect and not both of them. As such, this paper investigates the issues involved in considering both functional and non-functional aspects in service discovery. In particular, it proposes different ways via which aspect-specific algorithms can be combined to generate a complete service discovery system. It also proposes a specific unified service discovery architecture. Finally, it evaluates the proposed algorithms' performance to give valuable insights to the reader.

Keywords: service, discovery, matchmaking, semantics, ontology, performance, evaluation, functional, non-functional, QoS, architecture

1 Introduction

Nowadays, modern applications and business processes adopt service-orientation due to the many advantages it delivers, including loose coupling, re-usability, increased performance and cost reduction. To construct such applications, the services from which they are built need to be described appropriately, discovered and finally composed. Concerning service discovery, the state-of-the-art can be split into approaches that either focus on functional or non-functional aspects.

Functional service discovery work [8] matches user's functional requirements by exploiting various types of techniques from information retrieval and the semantic web [9,16]. Functional requirements and capabilities are mainly described via service IO while some work [7] covers behavioral aspects via service preconditions and effects which are however not available in real service advertisements. Most work nowadays exploits semantic techniques to exhibit better accuracy levels and other techniques to achieve performance speedup. However, the discovery accuracy is imperfect due to non-consideration of behavioural aspects.

Non-functional service discovery work [10] can be split into three main categories. Ontology-based approaches [19] employ subsumption techniques to infer

the matching between ontology-based non-functional service descriptions but are suitable for unary-constrained specifications. Constraint-based approaches [5] exploit n-ary specifications as models, including quality terms drawn from common vocabularies, and particular metrics which involve solving one or more constraint (combined) models to infer the matchmaking. Mixed approaches [10] combine the best of both worlds by exploiting ontology-based specifications to cover the non-functional semantics and align the quality terms involved as well as the metrics in the previous approach type to perform the service matchmaking.

While each aspect is more or less well covered in literature, very few approaches [2, 6] deal with both aspects concurrently. Such approaches, however, do not adopt the best possible algorithm for each aspect, do not capture service semantics, and do not have a suitable performance and accuracy level. Moreover, they have not explored the possible ways the two different types of matching can be best combined to infer the best possible one. In fact, most of these approaches employ a simplistic approach to account for non-functional user requirements and preferences which will never be adopted by respective practitioners.

As such, this paper first proposes a unified architecture explicating how different-aspect algorithms can be integrated to support a complete service discovery process by also accommodating for semantics capturing. Then, the paper proposes different combinations of aspect-specific algorithms attempting to accelerate the overall matching performance by not compromising discovery accuracy. Some combinations might be naturally applied and easily realised while others attempt to intelligently organise the service advertisement space to reduce the matchmaking time. These combined algorithms are finally evaluated by a semi-randomised framework according to their performance so as to provide particular insights on which is the preferred one in different circumstances.

The rest of the paper is structured as follows. Section 2 reviews the related work. Section 3 presents the unified architecture. Section 4 analyses the proposed combined algorithms. Section 5 presents the performance evaluation results. Finally, Section 6 concludes the paper and draws directions for further research.

2 Related Work

As we focus on combined service discovery, we only consider combined approaches. For aspect-based discovery analysis, the reader can refer to [8] and [10].

QoS Ranking. By reviewing related work, it seems that most approaches [12–14] first functionally match the service request and then rank the respective matches based on the user's non-functional preferences. Non-functional ranking usually relies on considering utility functions that depend on the respective quality term monotonicity while the overall rank is produced via a weighted sum of the application of the utility functions over the match's promised quality term values. Based on the above, it seems that all such approaches neglect the fact that a user may pose non-functional requirements which must be respected such that the functional matches are further filtered before they are ranked.

The approach in [14] caters for ontology encoding and fast reasoning issues. It attempts to smartly organise the functional advertisement space by exploiting two advertisement relations that seem to map to well-known functional degrees of match. However, the second relation seems not to be suitable based on the formal notion of subsumption. After cleverly matching a request, the functional matches produced are just ranked based on their non-functional degree of match.

The sequential approach in [13] starts by discovering services that functionally match the request and have an appropriate distance from the user to minimise network latency. Then, the expected execution time of each matched service is computed based on performance ratings which is finally exploited to rank the matched services and select the top one for dynamic adaptation reasons.

QoS Threshold-Based Filtering. A small improvement over the previous category comes via threshold-based filtering of functional matches [6,15]. However, it is questionnable whether a simple threshold can be enough to respect the semantics of all non-functional requirements posed. It rather seems as a trial-and-error approach towards attempting not to overwhelm users with irrelevant results not satisfying their non-functional requirements. What makes the approach in [6] more interesting with respect to the rest in this category is that it attempts to enable a unified semantic service description and considers various types of information to infer the ranking, including QoS, business policies and context.

Combined. [2] proposes a sequential combined algorithm coupled with service ranking based on non-functional preferences. This algorithm actually resembles *SeqOnTheFly* (see Section 4) as it attempts to match on the fly each functional result with the user's non-functional requirements. However, this work is not assorted with specification validation mechanisms and does not employ specification alignment, thus relying on a common quality term vocabulary. It does not also check how sequential matching can be enhanced for better performance.

In [3], a three step discovery process is proposed. First, functional matching is performed by exploiting subsumption hierarchy and predicate-based inferencing. Second, functional matches are clustered based on QoS via the average linkage clustering and squared Euclidian distance metrics. Third, the best cluster's matches are ranked based on each match's distance from the cluster centroid. This approach is regarded as combined as it performs a kind of filtering on the functional match space. However, it is questionnable whether such a filtering is suitable if we also consider performance aspects. In addition, semantics for QoS terms are neglected. Functional matching seems also to be wrongly performed.

3 Architecture

Figure 1 depicts the architecture of a complete and unified service discovery system by also showing the interactions between the respective components and their ordering in terms of basic discovery system operations. The architecture comprises 10 main components: 2 constitute entry points, 4 map to the main discovery logic and 4 relate to the respective individual and combined registries.

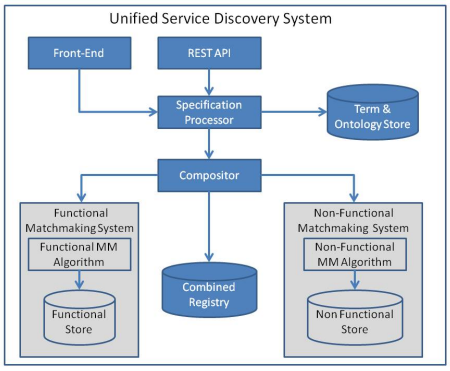


Fig. 1: The architecture of the unified service discovery system

Front-end is the first entry point, a web-based UI interacting with a human user, either a service provider or requester. It visualises information assisting the respective interaction, such as forms to specify functional and non-functional requests or to enable providers to upload, register or update service descriptions.

The *REST-API* is a service-based component enabling programmatic interaction with automated agents acting on behalf of users. It exposes the functionality expected in a service discovery system, like registering, updating and matching service specifications. It also offers utilities that validate the specifications before being registered. Service provider-related functionalities are only available to registered users, while discovery ones are publicly available with additional features only offered to registered users, like the ability to retrieve all matchmaking results and provide customised algorithms for service selection.

Once one operation starts execution, the respective specifications are passed to *Specification Processor* which loads and processes them to check their syntactic and semantic validity as well as to align them if they refer to equivalent but differently described terms. Constraint-based consistency is also checked for non-functional specifications. In case of a validation error, an error message is relayed to the user. Otherwise, operation passes to the core discovery component, the *Compositor*. Specification alignment is performed by consulting a *Term and Ontology Store* which includes a common set of basic terms via which alignment can be rapidly performed (see [10]) as well as the domain ontologies encountered.

Compositor realises the composition logic with respect to the individual aspect-specific algorithms exploited. It also guarantees the consistency of the information being registered in these algorithms. This is achieved via the *Combined Registry* storing the mappings between functional and non-functional specifications of service providers. This integration approach enables decoupling the aspect-specific discovery functionality and independence from any service specification language. Language adoption is coupled with selecting an aspect-specific algorithm. As such, we cater for using either unified or aspect-specific service description languages. In case the latter language type is used, consistency is

maintained via the entries of the *Combined Registry*. We aim at semantic languages for which any service profile kind is identified via a *URI*. This enables using only semantic algorithms but leads to increased discovery accuracy.

The *Compositor* implements the combined service matching logic with respect to the main algorithms proposed (see Section 4). It also guarantees the transactionality of service (de-)registration and update operations. This means that if an operation fails when executed via a specific aspect-specific sub-system, and is successful with respect to the other sub-system, we have to roll-back to the previous state before operation execution. The latter maps, e.g., to de-registering a functional service specification if its non-functional counterpart fails to be registered. This transactionality type is enabled by realistically assuming that each respective aspect-specific sub-system provides aspect-specific operations that return either boolean values indicating the operation outcome (e.g., non-registration as functional service specification already exists) or exceptions when errors occur. The respective suite of functionally-equivalent service operations is obviously already available in each aspect-specific system.

Each aspect-specific sub-system should provide an entry point via which aspect-specific interactions can be performed (either a programmatic REST-API or a specific component). For the two main aspects, we name the respective components as *Functional Service Discovery* and *Non-Functional Service Discovery*. We do not unveil their structure as it can be specific to the sub-system selected. We just unveil that each sub-system logically has a (functional / non-functional) store in which aspect-specific service specifications are stored.

Concerning implementation details, all system code was realised in Java as it is the main implementation language for almost all matchmakers. *Front-End* implementation is on-going while the *REST-API* was realised via Jersey. The *Specification Processor* exploits different loading and validation techniques. The Pellet reasoner [17] is exploited for ontology-based loading and consistency checking. The Ibex (`www.ibex-lib.org`) finite constraint and Choco (`http://www.choco-solver.org/`) constraint programming solvers are exploited for constraint-based consistency checking. The *Combined Registry* is a serialisable Java object which exposes different methods related to manipulating aspect-specific specifications (e.g., registering a functional and non-functional service URI pair or an additional non-functional profile for an existing service).

Alive Matchmaker [4] was selected as a state-of-the-art functional matching sub-system which exhibits high performance levels due to applying smart structures via which the matching of semantic I/O concepts can be performed while also catering for domain ontology evolution. The *Unary* algorithm of the discovery system in [10] was selected for hybrid non-functional service matching as it is scalable, exhibits high performance levels and has perfect accuracy.

4 Algorithm Analysis

Combining aspect-oriented matching algorithms was explored under different criteria. The first one concerns the expected way to combine two functionalities,

where two possible ways exist: (a) each functionality is performed in sequence or (b) both functionalities are executed in parallel and their results are integrated.

Concerning (a), we chose to execute first the functional discovery algorithm as this more naturally depicts the process executed by humans who first seek to satisfy their functional requirements and then the non-functional ones. Functional service discovery can also be considered more restrictive than non-functional one. This can be due to the fact that when performing non-functional matching, domain-independent metrics are usually considered leading to obtaining many functionally irrelevant results out of the respective domain scope. As such, when no results are discovered in the first discovery form, there is no reason to continue with the non-functional one spending unnecessary resources.

Solution (b) attempts to execute the aspect-specific discovery algorithms in parallel to save time. Compared to the first approach, it may spend more resources but it will be surely faster, provided that when one aspect-based discovery algorithm ends earlier with not result, we can stop the other one.

The second criterion explored exploiting different structures to smartly organise the service advertisement space to speed-up service matching. By relying on the approach in [10], we have considered combined subsumes relations between different service discovery offers (where a discovery offer maps to one functional and non-functional offer pair for one service) enabling us to not browse the whole advertisement space but stop at certain places without requiring to go down in respective subsumes branches. Figure 2 depicts the notion of a subsumes service advertisement hierarchy via a small forest of 4 offers. Offer O_1 subsumes offers O_{11} and O_{12} for different reasons. Offer O_{11} is subsumed due to its functional part (its non-functional part is equivalent) while offer O_{12} is subsumed due to its non-functional part (its functional part is equivalent). Offer O_2 is not related to the tree's offers as it possesses an unrelated functional part. This also highlights the definition of the combined subsumes relation: $comb\_subsumes\,(S_1, S_2) \equiv func\_subsumes\,(S_1, S_2) \wedge nonfunc\_subsumes\,(S_1, S_2)$

By exploiting this subsumption relation, we speed up the service matching process. For example, assume that request R is issued. We will first compare it with offer O_1. If R subsumes O_1, it also subsumes its descendants. As such, we do not have to go deeper into the respective tree. However, we need to also check other trees in the forest which could be partially related to each other. There are two relation kinds to be exploited: *subsumes* and the opposite one, *subsumedBy*. Based on the empirical evaluation in [10], the first relation is more suited when more than 30 percent of the offers match a request. Otherwise, the second relation is more suitable. By combining functional and non-functional service matching, we expect that the percentage of matched offers can be lower than in the case of aspect-specific matching. In this way, it might be preferrable to exploit the *subsumedBy* relation in the end, especially for a highly populated service registry spanning multiple domains as each request is expected to be specific to just one domain and thus lead to matching of a quite low offer percentage.

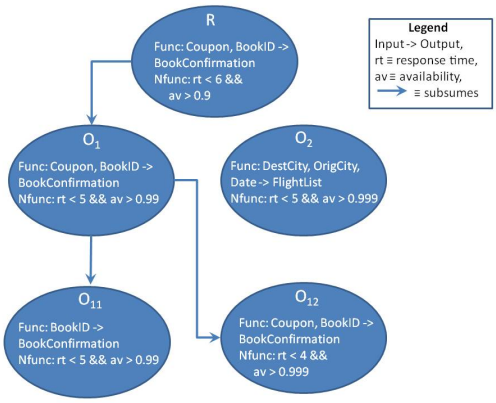


Fig. 2: The subsumption forest with request R subsuming a root tree node

In the sequel, a small section is first provided explaining the main symbols and assumptions made. The next four subsections shortly analyse the proposed algorithms' functionality for service registration and discovery by also providing a respective complexity analysis. Other operation types (e.g., deletion and updating) are not considered as they tend to map either to similar actions as in case of registration (i.e., deletion) or to a two actions sequence (i.e., updating by combining deletion and insertion). Finally, the last section discusses the expected performance of the proposed algorithms based on the complexity analysis.

4.1 Symbols & Assumptions

We assume that one functional and non-functional part comprise the service request. We also assume that S service offers have been registered. This means that around $\frac{S}{3}$ functional offers are to be registered in a functional matchmaker and S non-functional offers in the non-functional one as each functional offer is expected to be accompanied by 3 non-functional offers for the same service, mapping to gold, silver and bronze classes of customers. Functional offer pre-processing takes $O(1)$ while for a non-functional offer takes $O(2M + M * T_z)$, where the first part maps to the offer's consistency checking and the second to its alignment; M represents the number of the offer's quality terms. The latter can be reduced to $O(M * T_z)$ constituting the total pre-processing time. Finally, we assume that each non-functional offer comprises at most $2M$ constraints mapping to the upper and lower bounds given for each quality term.

4.2 Sequential Algorithm

Analysis *Matchmaking.* We propose two main algorithm variations named as: *Sequential* and *SeqOnTheFly*. The first version first performs functional service discovery; then it fetches the respective non-functional offers from the functional

results, registers them to the non-functional matchmaker and matches them with the non-functional request part. Finally, it returns back the ending results to the user and clears the non-functional matchmaker. On the other hand, the second version checks on the fly whether the non-functional offers mapping to the matched functional ones are subsumed by the request's non-functional part. We expect that the first version is more suitable when non-functional offers are great in number as the total registration time will be compensated by the fast matching via the matchmaker's smart structures. Otherwise, the second version should be preferred. Section 5 will explore which is the amount of non-functional offers that represents the break point between choosing one over the other version.

Registration. Both algorithm versions follow the same registration process by registering the functional offer in the functional matchmaker and inserting the respective combined entry in the *Combined Registry*. Non-functional-based registration is not needed due to the way the non-functional matchmaker is exploited. The pre-requisite processing step for specifications controls their validity and transactionally rolls back the combined registration, if needed.

Complexity analysis *Matchmaking.* We rely on the complexity analysis of the aspect-specific matchmakers and on the fact that the request validity has to be checked. Pre-processing as already stated takes $O(M * T_Z)$.

The functional matching part [4] requires $O\left(R_O * \frac{S}{3} * R_{AO} + M_{FO} * I_S^{Mean}\right)$, where R_O, R_{AO} represent the number of input and acceptable inputs of the request, respectively, $\frac{S}{3}$ represents the number of functional service offers registered, M_{FO} the number of matched services based on their output and $I_{M_{FO}}^{mean}$ the mean number of inputs of each matched service. The latter can be reduced to $O(R_O * S * R_{AO})$ as $\frac{S}{3}$ is expected to be much bigger than M_{FO} and I_S^{Mean} could be at most 3. Non-functional matching time depends on the combined algorithm version. For each version, we assume that $O(k * M_{FO})$ non-functional offers must be matched, k non-functional offers for each functional match, based on our assumption for the mapping between functional and non-functional offers.

For normal non-functional matching, in the worst case, covering both registration and matching of non-functional offers, the time will be $O(M * (M_{FO} + 2 * log_{M_{FO}}))$. For on the fly non-functional matching, $O(2M * M_{FO})$ time is needed as we must check each constraint of a non-functional offer with the request non-functional part's respective constraint. Thus, in the end, the matchmaking time for *Sequential* will be $O(R_O * S * R_{AO} + (M * (M_{FO} + 2 * log_{M_{FO}})) + M * T_Z)$. If we consider that M is small and cannot go beyond 10 quality terms and that S is much bigger than M_{FO}, the complexity formula can be reduced to: $O(R_O * S * R_{AO} + T_Z)$.

For *SeqOnTheFly*, the matching time will be $O(R_O * S * R_{AO} + (2M * M_{FO}) + M * T_Z)$ which can be similarly reduced to $O(R_O * S * R_{AO} + T_Z)$. Thus, in the end, the matching complexity for both algorithm versions coincides.

Registration. One functional and its respective 3 non-functional offers are to be registered. Thus, we need to pre-process 4 specifications and only register via the functional matchmaker the functional offer. Pre-processing takes again

$O(M * T_Z)$ as in matchmaking. Functional registration takes $O(S_C)$ as in the worst case is dominated by the time needed to infer the subsumption hierarchy of the domain ontology mapping to the service I/O, where S_C represents this time for an ontology of C concepts. Thus, registration time for both algorithm versions will be $O(T_Z + S_C)$.

4.3 Parallel Algorithm

Analysis Matchmaking is performed, after user request is pre-processed and aligned, by exploiting in parallel the functional and non-functional matchmakers and then concatenating their results. The exploitation of the *Combined Registry* to map functional matches to their non-functional counter-parts (which can be 3 times their number) guarantees the concatenation of the same type of objects.

For registration, we register in parallel the functional offer and non-functional part in the functional and non-functional matchmakers, respectively. The respective consistency is achieved by informing the *Combined Registry*.

Time Complexity Analysis *Matchmaking.* Pre-processing as in the previous algorithm takes $O(M * T_Z)$. Functional matching, as already stated, takes $O(R_O * S * R_{AO})$. Non-functional matching takes $O(M * (S + log_S))$. Concatenating the different result types takes: $O(M_{FO})$. In the end, the total matchmaking time will be: $O(\max(R_O * S * R_{AO}, M * (S + log_S)) + M_{FO} + M * T_Z)$ which can be further reduced to: $O(S + T_Z)$.

Registration. Functional registration takes $O(S_C)$ as indicated in the previous algorithm. Non-functional registration takes $O(M * log_S)$. Thus, total registration time will be $O(\max(S_C, M * (log_S + T_Z)))$ which can be reduced to $O(\max(S_C, log_S + T_Z))$.

4.4 Subsumes Algorithm

Analysis *Matchmaking.* It is recursively performed [10]. The offer is matched with each root node. If it subsumes the node, it also subsumes its children. As such, we just consider this node and its descendants as matches. Otherwise, we must descend this hierarchy recursively to find respective matching nodes. All matching results for each root node search (including recursive calls) are finally collected to be returned to the user.

Registration. It is recursively performed [10]. First, each root node is checked with the offer to be matched. If the offer is equivalent to the node, it is entered into the node's represented offers and registration ends. If the offer subsumes the node, it becomes its parent. If the root node subsumes the offer, the same checking is performed recursively at the node's children. When at the root node's subsumption hierarchy the offer is subsumed by one node but does not subsume its children, the offer is entered as a child of this parent node.

Complexity Analysis *Matchmaking.* 3 cases hold [10]. In the worst case, we perform functional and non-functional subsumption checking for all forest nodes. In functional subsumption checking, we expand each output concept of the functional offer with respect to its ancestor concepts in $O(1)$ step and check if each output concept of the functional request is inside one of the expanded lists. This takes $O(R_O * S_O)$ for each functional offer. For non-functional subsumption checking, we chech if each offer constraint is more restrictive that the respective demand constraint. This takes $O(2 * M)$ as for each offer constraint, we immediately find the request counter part. So, individual subsumption checking takes $O(R_O * S_O + 2M)$. By visiting all S nodes and accounting processing time, total matching time is: $O(S * (R_O * S_O + 2M) + M * T_Z)$. As both R_O and R_O tend to be small and M is less than 10, this reduces to just $O(S + T_Z)$.

In the best case, only the sole root node is matched (forest reduced to a tree) which takes $O(S + T_Z)$. In the average case, we expect that the tree is more or less balanced, a percentage of P nodes is subsumed and there is always a pair of parent-child offers. In this case, total matching will taje: $O\left(S * \left(1 - \frac{P}{2}\right) + T_Z\right)$.

Registration. Three cases are also considered. For all cases, we need to do preprocessing but also account for ontology subsumption-based structure updating. This maps to $O(S_C + T_Z)$. In the best case, the first root node compared with the new offer is equivalent to it. This ends up doing two comparisons (one for node-to-offer subsumption and one for offer-to-node subsumption) and translates to $O(2 * (S * (R_O * S_O) + 2M))$. Thus, in total, the time will be $O(S_C + T_Z)$.

In the worst case, we have a single tree and the offer has to be inserted in the rightmost leaf. This maps to performing twice the subsumption checking over all tree nodes which will map to $O(S + S_C + T_Z)$ in the end.

In the average case, we will have B balanced trees and the offer has to be inserted in the middle of the median tree. This will map to $\frac{S+B^2}{2B}$ subsumption checks which will then map in the end to a total time of: $O\left(\frac{S+B^2}{2B} + S_C + T_Z\right)$

4.5 SubsumedBy Algorithm

Analysis *subsumedBy* is opposite to *subsumes*. We organise the offer hierarchy in this way to cater for the case that a very small offer number matches a request to be placed in the hierarchy's leaves (as subsumption maps to something more restricted or specific so less matches means more restricted matches).

Matchmaking. It is performed by matching the request with the root nodes based on the *subsumes* relation. If there is no match, there is no need to go down the root node's hierarchy (as either there is no relation at all or the request is *subsumedBy* the root node). Otherwise, we include the root node in the final matches and visit its children recursively. All matching results from each (recursive) root node search are finally collected and returned to the user.

Registration. It is symmetric to *Subsumes*. We check first the root nodes. An offer equivalent to a root node is included in the offers represented by the node and registration ends. If the offer is subsumed by this node, it becomes the node's parent. If it subsumes the node, we go recursively to the node children.

The offer finally becomes a child of a root node descendant or a forest root (in case it is unrelated to any root node or subsumed by one or more root nodes).

Complexity analysis *Matchmaking.* The same complexity as in *Subsumes* holds for the best and worst cases. The sole exception is the conditions mapping to these cases. In the best case, the sole root node is not subsumed by the request. In the worst case, all non-subsumed nodes by the request lie in the forest leaves and do not represent more than one offer. In average, the same assumptions hold as in *Subsumes*. This, however, maps to matching the request with $O\left(S * \frac{P+1}{2}\right)$ nodes which finally maps to the total matching time of: $O\left(S * \frac{P+1}{2}\right) + T_Z$

Registration. This process is identical to that of *Subsumes* concerning the best and worst cases, so the respective time complexity is the same. The same holds for the conditions mapping to worst, best and average cases.

4.6 Discussion

Concerning matchmaking, based on the time complexity analysis, it seems that in the worst case the *Parallel* algorithm is the best followed by *Sequential* and then the *subsumes*-based algorithms. However, in the average case, if there is some kind of subsumption hierarchy between the different nodes, it might be the case that the *subsumes*-based algorithms have the best possible performance and the best algorithm can depend on the percentage of offers being matched.

Concerning registration time, the performance order seems to be clearer as the *Parallel* algorithm must have the best performance followed by the sequential and then the *subsumes-based* algorithms. So, there is a clear winner plus a performance trade-off between matchmaking and registration for the second place where different algorithms are nominated as best for different operations.

5 Evaluation

The evaluation was performed via the experimental framework in [10], covering the non-functional aspect and being able to generate in a controlled manner both randomised and real non-functional service specifications, as well as the OWLS-TC framework, covering the functional aspect with real or realistic functional service specifications along with ways to measure functional discovery accuracy. The main goal was to evaluate the algorithms' average matchmaking and registration performance. Accuracy is neglected due to the following reasons: (a) non-functional matchmakers have perfect accuracy [10]; (b) accuracy results will be identical to those of the functional matchmaker exploited that have already been reported. The target is to discover those circumstances that the selection of a specific algorithm from those proposed is recommended. We also consider the *AliveMM* functional matchmaker so that we are enabled to compare the overall matchmaking time with respect to the functional part and unveil the degree in which the latter influences the former.

In the sequel, we explicate the way the sole experiment was conducted based on the experimental framework and then discuss the experiment results.

5.1 Experiment Set-Up and Control

Unified Framework Features. By combining aspect-specific experimental frameworks, we generate an overall framework with the capabilities to create in a controlled manner both functional and non-functional specifications. Functional specifications can either rely on the OWLSTC collection, to be as realistic as possible, or can be produced randomly [11] via a domain ontology's concepts combined to produce the service I/O. Non-functional specifications rely on the WS-Dream [18] and QWS [1] datasets or on generating randomised unary specifications that can include integer or real-valued metrics of different types and semantics. In the experiment conducted, we relied on OWLSTC and the randomised generation of non-functional specifications. Our choice for the non-functional aspect relates to the fact that the WS-Dream dataset is big but quite limited in the metric number while QWS is smaller. However, we desire to generate a much greater non-functional offer set with an increased metric number such that respective requests can also match a particular offer percentage.

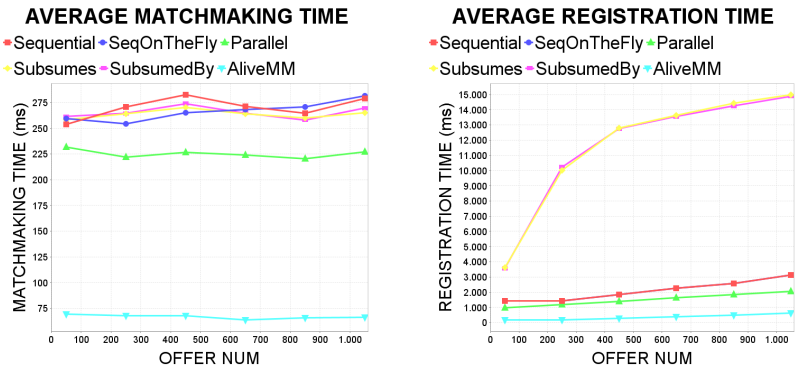
Offer Generation. For each functional service offer, three non-functional offers were generated in a controlled manner. This resulted in generating around 3150 service offers (functional and non-functional pairs) provided that OWLSTC contains around 1050 functional service specifications. This also supports our main real-world assumption that each service can be associated to three non-functional offers catering for gold, silver and bronze customer classes. The offer number considered in each experiment step depended on a specific configuration parameter given as input to the specification generator. For instance, if this number is 100, we randomly take 100 functional offers from OWLSTC and couple each to 3 non-functional offers randomly generated such that the next non-functional offer has an increased performance and price with respect to the previous one to map to the different customer classes.

Experiment Set-Up. The experiment was conducted in a Windows 7 SSD-based machine with 2GHz dual-core CPU and 6 GB of RAM. It included conducting a set of steps in which one configuration parameter varied (the offer number) according to a specific range (50 to 1050 with an increase step of 200 for the functional offers). Each step was executed 30 times to produce the respective average matchmaking and registration time values of the considered algorithms such that any possible interference at the OS level is diminished.

5.2 Experiment Results

Figure 3 depicts the experiment results for matchmaking and registration time.

Concerning matchmaking time, the best algorithm is *Parallel*. The order for the rest is not stable. Initially, the sequential algorithms are better than the subsumes; this is reversed when the functional offer number is equal or above 650. This means that beyond a specific offer number, the offer hierarchy becomes more structured such that matching time can be really saved via a subsumes-based approach. This matches exactly our expectations for the subsumes algorithms.



(a) (b)

Fig. 3: Fig. (a) shows the experiment service matchmaking results while Fig. (b) the service registration ones

The partial order between the sequential algorithms is also unstable. Initially, as the number of non-functional offers to filter is small, *SeqOnTheFly* is slightly faster. However, beyond 650 functional offers, this is reversed as it becomes better to employ a non-functional matchmaker to register and match the non-functional offers rather than employ pair-wise non-functional filtering.

The subsumes algorithms are equivalent apart from the initial step where probably the matched offer number is quite small thus favouring *SubsumedBy*. This needs further investigation as we need to discover those circumstances that one algorithm prevails over the other to assist practitioners in their selection.

The results produced validate the complexity analysis (especially with respect to the algorithm order) due also to corresponding OWLSTC features. In particular, a more or less stable algorithm performance is seen due to the small matched offer percentage and the small output parameter number per service. This leads to a more or less stable matching performance for the functional matchmaker as depicted in the results. Moreover, while the number of non-functional offers to be matched is greater in each step, the scalable non-functional matchmaker used enabled to reach almost equivalent performance levels. So, these matchmakers combination also leads to a stable matching performance.

Non-functional matching time takes more with respect to functional one. This is proven by comparing the functional matchmaker and *Parallel* algorithm performance. Thus, non-functional matching has still space for further optimisation. This also indicates that it is always proper for a sequential matchmaker to first filter based on the functional aspect and then the non-functional one. This is an interesting result to be accounted by researchers and practitioners.

Concerning registration time, the best algorithm is again *Parallel* followed by the sequential ones. The difference between these algorithms is small but is big with respect to the subsumes ones. The complexity analysis also validated this. This means that probably the subsumes algorithms should not be used in cases

when a high offer number is constantly registered or updated. However, it can also be acceptable in the rest of the cases. So, the use of these algorithms depends on the registry provider's preferences and constraints especially with respect to the main requirements of its clients, i.e., service providers and requesters.

No ordering between sequential and between subsumes algorithms can be inferred from the results. This is natural as the sequential algorithms rely on the same registration process. For the subsumes algorithms, by also relying on the matchmaking results, it seems that the structures produced are more or less similar, leading to almost the same registration time.

To conclude, we stress that the *Parallel* algorithm seems to be the best for both matchmaking and registration so it is undoubtly recommended as the ideal algorithm for service registry realisation. In case that a different algorithm is needed or preferred, then the recommendation will be towards the subsumes algorithms despite the fact that their behavior with respect to registration is the worst. However, for highly dynamic environments, it seems that the best choice will be the sequential algorithms as an alternative to *Parallel* due to their capability to also deal with the dynamicity in service updating.

6 Conclusions and Future Work

This paper has presented four algorithms which attempt to combine in a different way the facilities of functional and non-functional state-of-the-art service discovery algorithms. We believe that this investigation is genuine and really assists practitioners in choosing the algorithm that best matches their current situation. The respective algorithm evaluation has unveiled the circumstances in which each algorithm prevails based on performance aspects. Apart from these 4 novel algorithms proposed, we have also implemented an unified service discovery architecture covering both the functional and non-functional aspects. Such an architecture comprises components that not only perform core service discovery tasks but also specification validation and alignment. It also includes components that enable both a visual and a programmatic interaction with a human or software agents, respectively. The algorithm combination is performed such that transactionality of offer registration and updating is achieved.

Concerning future work, the following directions will be pursued. First, more thorough validation of the proposed algorithms to produce even more interesting performance insights. Second, completing the development of the service discovery architecture. Third, extending functional matching towards covering the service functional behaviour to further increase discovery accuracy in case respective formal service descriptions are in place. Finally, integrating the service discovery system in an existing service composition framework to enable a faster and more accurate service composition process.

Acknowledgments This research has received funding from the European Community's Framework Programme for Research and Innovation HORIZON 2020 (ICT-07-2014) under grant agreement number 644690 (CloudSocket).

References

1. Al-Masri, E., Mahmoud, Q.H.: Investigating web services on the world wide web. In: WWW. pp. 795–804. ACM, Beijing, China (2008)
2. Benaboud, R., Maamri, R., Sahnoun, Z.: Agents and owl-s based semantic web service discovery with user preference support. International Journal of Web & Semantic Technology 4(2), 57–75 (2013)
3. Charrad, M., Ayadi, N.Y., Ahmed, M.B.: A Semantic and QoS-aware Broker for Service Discovery. Journal of Research and Practice in Information Technology 44(4), 387–399 (2012)
4. Cliffe, O., Andreou, D.: Service Matchmaking Framework. Public Deliverable D5.2a, Alive EU Project Consortium (10 September 2009), available at: `http://www.ist-alive.eu/index.php?option=com_docman&task=doc_download&gid=28&Itemid=49`
5. Cortés, A.R., Martín-Díaz, O., Toro, A.D., Toro, M.: Improving the Automatic Procurement of Web Services Using Constraint Programming. Int. J. Cooperative Inf. Syst. 14(4), 439–468 (2005)
6. Jiang, S., Aagesen, F.A.: An Approach to Integrated Semantic Service Discovery. In: First International IFIP TC6 Conference. pp. 159–171. Springer Berlin Heidelberg (2006)
7. Klein, M., Konig-Ries, B.: Coupled Signature and Specification Matching for Automatic Service Binding. Springer Berlin Heidelberg (183-197)
8. Klusch, M.: Semantic Web Service Coordination. In: CASCOM: Intelligent Service Coordination in the Semantic Web. pp. 59–104. Springer (2008)
9. Klusch, M., Fries, B., Sycara, K.: OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. Web Semantics: Science, Services and Agents on the World Wide Web 7(2), 121 – 133 (2009)
10. Kritikos, K., Plexousakis, D.: Novel Optimal and Scalable Nonfunctional Service Matchmaking Techniques. IEEE T. Services Computing 7(4), 614–627 (2014)
11. Kritikos, K., Plexousakis, D., Paternò, F.: Task model-driven realization of interactive application functionality through services. TiiS 3(4), 25 (2014)
12. Lemos, F., Grigori, D., Bouzeghoub, M.: Adding Non-functional Preferences to Service Discovery. In: ICWE. pp. 299–306. Springer Berlin Heidelberg (2012)
13. Makris, C., Panagis, Y., Sakkopoulos, E., Tsakalidis, A.: Efficient and Adaptive Discovery Techniques of Web Services Handling Large Data Sets. J. Syst. Softw. 79(4), 480–495 (Apr 2006)
14. Mokhtar, S.B., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y.: EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. J. Syst. Softw. 81(5), 785–808 (2008)
15. Pathak, J., Koul, N., Caragea, D., Honavar, V.G.:
16. Plebani, P., Pernici, B.: URBE: Web Service Retrieval Based on Similarity Evaluation. IEEE Transactions on Knowledge and Data Engineering 21(11), 1629–1642 (2009)
17. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. Web Sem. 5(2), 51–53 (2007)
18. Zhang, Y., Zheng, Z., Lyu, M.R.: WSPred: A Time-Aware Personalized QoS Prediction Framework for Web Services. In: ISSRE (2011)
19. Zhou, C., Chia, L.T., Lee, B.S.: DAML-QoS Ontology for Web Services. In: ICWS. p. 472. IEEE Computer Society (2004)