An Automated Approach for Selecting Web Services

Alysson Alves, Gledson Elias
Informatics Center
Federal University of Paraíba
João Pessoa, Brazil
e-mail: a.alvesdl@gmail.com, gledson@ci.ufpb.br

Abstract—**The task of selecting web services is one of the main challenges for successfully exploring the Service-Oriented Architecture (SOA) approach in software development processes. Whereas the availability of web services tends to increase rapidly in the software industry, it is impractical to adopt ad-hoc manual approaches for selecting web services. Thus, considering a very large and complex search space, it is required an automated approach for selecting web services. In such a direction, exploring Search Based Software Engineering (SBSE) techniques, this paper proposes an automated approach for selecting web services, whose optimization strategy is based on functional and structural metrics that evaluate the functionalities provided by candidate web services, as well as their dependencies in the architectural level. As main contribution, experimental results show that the proposed approach represents an extremely complex problem in a systematic and structured way, discovering good-enough or even optimal solutions among the candidate web services.**

Keywords-Web Services; Service-Oriented Architecture; Search Based Software Engineering.

I. INTRODUCTION

The advancements in software engineering approaches have contributed for increasing productivity in software development processes [1]. As a promising approach, software reuse has the potential to reduce development time, cost and risk during the development of a software product [2]. In such a context, Service-Oriented Architecture (SOA) has emerged as one of the main software reuse approaches, in which software systems can be developed reusing services available in the internet. Note that, SOA is an architectural style for building software systems, while Web Services (WS) are the preferred standards-based way to realize SOA [3].

Ideally, web services are perfectly connected and integrated without additional adaptation efforts for composing a software system or even a new web service. However, in practice, web services can be developed by different software providers, and, generally, such services can only be integrated with additional adaptation efforts for resolving incompatibilities among their required and provided functionalities [4]. As a consequence, such incompatibility issues must be already considered during the selection of the candidate web services, trying to choose more compatible candidates as a mean to reduce adaptation efforts, and consequently integration time and cost.

The selection of web services has proven to be a phase of major complexity in SOA-based development processes. Most processes for selecting web services take into account only quality attributes or non-functional requirements of the candidate web services, such as availability, reliability, response time and price. However, functional requirements also have significant impact in the quality of a SOA-based software product. Indeed, functional requirements make possible to assess the effectiveness of the integration of all candidate web services, minimizing integration mismatch issues. The higher the integration effectiveness, the lower the amount of incompatibilities that arise from the integration, and consequently the lower the adaptation efforts for integrating candidate web services.

Therefore, the selection of web services for a given architectural specification is a pivotal task that is more complex than traditional products selection [5]. Besides, taking into account that the availability of web services tends to rapidly increase in software industry, it is impracticable the adoption of ad-hoc manual approaches for selecting web services. In fact, considering a SOA-based architectural specification, several candidate implementations can exist for each web service specification included in the architectural specification. The amount of possible solutions creates a very large search space with exponential complexity, in which the base is the average number of candidate implementations and the exponent is the number of web service specifications included in the architectural specification.

As a consequence, considering a very large and complex search space, it is required an automated approach for selecting web services. Even adopting an automated approach, the search space is typically too large to be explored exhaustively, suggesting the adoption of metaheuristic search techniques explored in Search Based Software Engineering (SBSE), in which software engineering problems are reformulated as optimization problems that can be tackled with metaheuristics, such as Genetic Algorithms [6].

In such a direction, exploring SBSE techniques, this paper proposes an automated approach for selecting web services, in which from a SOA-based architectural specification, web service specifications are contrasted against their correspondent candidate implementations, which are selected by evaluating the effectiveness of their integration, minimizing integration mismatch issues, and consequently, reducing adaptation efforts for integrating them. In the proposed approach, the optimization strategy is based on functional and structural metrics that evaluate the functionalities provided by candidate web services, as well as their dependencies in the architectural level. As main contribution, experimental results show that the proposed approach represents an extremely complex problem in a systematic and structured way, discovering good-enough or even optimal solutions among the candidate web services.

The remainder of this paper is structured as follows. Section II introduces the main concepts and fundamentals related to the approach proposed herein. Then, Section III briefly discusses related work, evincing the contribution of the proposed approach. In Section IV, the proposed approach is presented, defining the metrics adopted in the optimization strategy. Thereafter, Section V presents an experimental evaluation in three case studies. Finally, concluding remarks and future work are discussed in Section VI.

II. Concepts and Fundamentals

According the OASIS consortium, SOA is a paradigm for organizing and utilizing distributed services that may be under the control of different ownership domains [2]. SOA has emerged as a means to promote software reuse, in which software systems can be developed reusing services available in the internet. On the one hand, SOA is an architectural style for building software systems, which can be implemented using different strategies or technologies. On the other hand, Web Services are the preferred standards-based way to realize SOA. Thus, while SOA is conceptual and abstract, WS-based architectures and technologies are specific and concrete.

Web Services technologies are built on top of XML based open standards, which abstract details related to network protocols, operating systems and programming languages. Among such standards, Web Services Description Language (WSDL) has a fundamental role in the context of the approach proposed herein. WSDL is an interface definition language that is used for describing the functionality offered by a web service, including the provided operations and their input and output parameters. Thus, its purpose is roughly similar to that of a method signature in a programming language.

SOA concepts and WS-based architectures and technologies support intra and inter-provider service integration. However, as already discussed, integration mismatch issues can arise and must be treated adopting automated approaches during the selection of the candidate web services. In such a context, considering a very large and complex search space, automated approaches for selecting web services have been proposed in the literature adopting metaheuristic search techniques explored in the SBSE field.

According Harman and Jones [6], in SBSE, software engineering problems are reformulated as optimization problems that can be tackled with metaheuristics, such as Genetic Algorithms and Simulated Annealing, facilitating automated and semi-automated solutions in situations typified by large complex problem spaces with multiple competing and conflicting objectives. Complementarily, in [7], Harman argues that software engineering provides the ideal set of application problems for which SBSE techniques are supremely well suited, once the virtual nature of software makes it ideal for search-based optimization.

In order to reformulate a given software engineering problem as an optimization problem, SBSE-based approaches ought to define: *(i)* a representation of the problem, which must be amenable to symbolic manipulation; *(ii)* a fitness function defined in terms of the adopted representation; and *(iii)* a set of manipulation operators, which are applied in the search algorithm for transforming candidate solutions.

The fitness function is the characterization of what is considered to be a good solution, imposing an ordinal scale of measurement upon candidate solutions. By contrasting the value of the fitness function for each candidate solution, metaheuristic search techniques can find good-enough or even optimal solutions. Although eventually possible, search techniques do not guarantee to find the optimal solution. Besides, due to their non-determinist aspects, they can find different solutions in different executions.

In the proposed approach, the adopted search technique is genetic algorithms, which is a class of evolutionary algorithm that mimics the biological natural evolution process as a problem-solving strategy, including operators such as crossover, mutation and selection [8]. In summary, a set of candidate solutions, represented as chromosomes, are quantitatively evaluated using the fitness function. Then, promising candidates are kept and allowed to reproduce using genetic operators, creating the next generation of candidates. The process repeats during several generations, making them into better, more complete or more efficient solutions.

III. Related Work

Selection of web services is a key research field in SOA-based development processes. As a consequence, it is possible to find several proposals in the literature [5][9][10][11][12][13], proving different strategies for selecting web services in more effective ways in order to reduce development time and cost. Despite their pivotal contributions, in general, such available proposals deal with criteria related to non-functional requirements only, more specifically those related to Quality of Service (QoS), including availability, reliability, execution cost and time, reputation, location and price. Few proposals can be found that deal with criteria directly related to functional requirements and structural properties, which clearly is the main contribution of the approach proposed herein, as will become clear in the following.

Briefly, this section presents and discuss six approaches identified in the literature which are related to our work to some extent. In [9], Fetthallah and coworkers propose a QoS aware service selection approach based on genetic algorithm. The Fetthallah's proposal has the aim of optimizing the composition of web services based on criteria, such as response time, availability, reliability, price and reputation. Lifeng and colleagues [10] define a penalty-based genetic algorithm for QoS-aware web service composition with service dependencies and conflicts. The Lifeng's proposal also considers QoS criteria only, such as response time, price, reputation, availability and reliability.

Vescan [11] presents an evolutionary approach for component selection. Based on genetic algorithms, it adopts QoS-aware metrics such as cost and reusability, but also includes a functional metric. Although adopts a functional metric, unlike the proposed approach, it does not try to identify mismatch issues among dependent components, but tries only measuring the ratio of functionalities provided by each component in relation to functionalities required in the whole system. Clearly, the functional and structural metrics adopted in the proposed approach are much more precise in evaluating mismatch issues.

Adopting similar QoS-aware criteria, Maamar and his fellows [12] have discussed the selection of web services for composition based on the criteria of execution cost, execution time and location of provider hosts. Besides, Tang and Cheng [13] analyzed the optimal location and pricing of web services from the view of web services intermediary, whose criteria can contribute to companies for making selection decisions.

Lastly, Feng and associates [5] examine an approach for web service selection based in six criteria (functional, price, location, integration and reputation). The functional criterion takes a rough-grain keyword-based search in a service repository considering required functionalities that the web services must fulfill. In contrast, instead of evaluating keywords related to functional requirements, the proposed approach evaluates the signature of operations provided and required by candidate web services, which represents a much more precise strategy than simple keyword-based search.

IV. Proposed Approach

By exploring SBSE techniques, the proposed approach has the goal of automating the web service selection process. In the proposed approach, the metaheuristic search algorithm is based on functional and structural metrics that evaluate the functionalities provided by candidate web services, as well as their dependencies in the architectural level. Together, both metrics evaluates the integration effectiveness among candidate web services. As a result, it is expected to find a near optimal architectural configuration, which minimizes integration mismatch issues, and consequently, reduces adaptation efforts for integrating its constituting web services. Figure 1 illustrates the stages of the proposed approach.

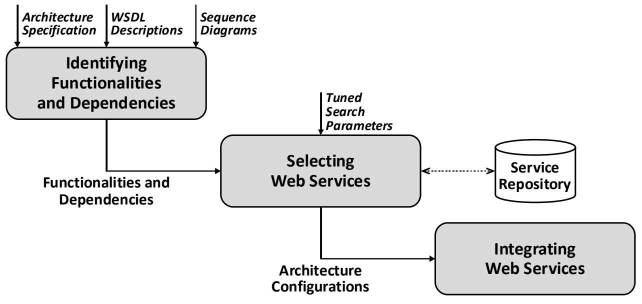
Architecture Specification
WSDL Descriptions
Sequence Diagrams
Identifying Functionalities and Dependencies
Tuned Search Parameters
Selecting Web Services
Service Repository
Functionalities and Dependencies
Integrating Web Services
Architecture Configurations

Figure 1. Stages of the Proposed Approach

The first stage, called *Identifying Functionalities and Dependencies*, has the purpose of identifying provided and required functionalities, as well as services dependencies. To do that, the first stage adopts as inputs three types of artifacts: the architecture specification, WSDL descriptions and sequence diagrams. As explained later, all of them are produced during the architectural design phase.

Upon identifying functionalities and dependencies, the second stage, called *Selecting Web Services*, represents the core of the proposed approach in which candidate web services are evaluated and then selected for composing near optimal architectural configurations that reduce adaptation efforts for integrating constituting web services. Note that several architectural configurations can be recommended, allowing the software development team to choose one that best meets the needs of the project and organization.

After selecting web services, in the third stage, called *Integrating Software System*, the software development team can integrate and adapt the set of web services included in the selected architectural configuration.

In this paper, the focus is on the first two stages of the proposed approach. Due to that, the next subsection introduces some notes about the identification of functionalities and dependencies. Then, in a succeeding subsection, the mathematical representations of the functional and structural metrics are presented in details.

A. Functionalities and Dependencies

Considering a SOA-based software development process, the architectural design phase must come before the service selection phase. In the architectural design phase, the software architect ought to identify the functionalities provided and required by each specified service, together with their dependencies. Such functionalities are specified as interfaces. When adopting Web Services technologies, interface specifications are explicitly described using WSDL, allowing to indicate the set of operations provided by each interface for each specified web service. Thus, in the proposed approach, provided functionalities are effortlessly extracted from WSDL descriptions evaluating a set of XML elements, including *portType*, *operation*, *input*, *output* and *message*.

Differently, required functionalities and dependencies cannot be explicitly represented in WSDL specifications. Instead, required functionalities and dependencies can be implicitly modeled using sequence diagrams associated with each operation provided by each specified web service. Thus, in the proposed approach, required functionalities and consequently service dependencies are extracted in a more elaborated way, evaluating sequence diagrams that show how web services collaborate and work together, revealing the set of operations required by one web service but provided by other ones. For instance, in Figure 2, it is possible to note that the *getPackage* operation, provided by the *TravelSrv* service, requires the *getFlight* operation, provided by the *FlightSrv* service. As a conclusion, the *TravelSrv* service requires the *getFlight* operation. Besides, the *TravelSrv* service depends on the *FlightSrv* service.

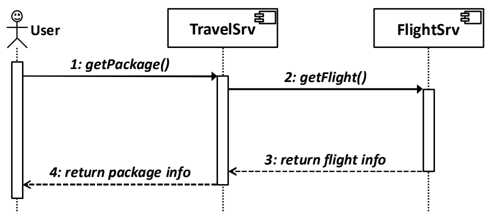

Figure 2. Service Dependency in a Sequence Diagram

After identifying provided and required operations, it is possible to generate an architecture specification that shows all constituting web services together with their dependencies (Figure 3). To do that, provided and required operations are respectively organized in provided and required interfaces, making the architecture specification to appear like those adopted in component-based development processes [14]. As can be noted, each service dependency is characterized by connecting the related services through their provided and required interfaces.

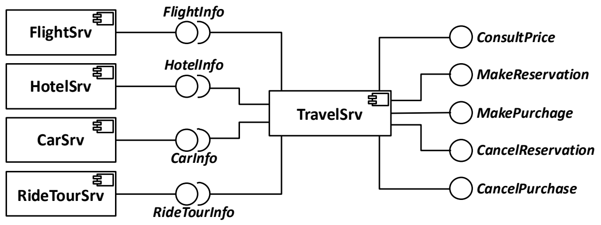

Figure 3. Architectural View for Service Dependencies

B. Functional and Structural Metrics

As already discussed, the proposed approach selects candidate web services by evaluating integration effectiveness through functional and structural metrics that evaluate the functionalities associated with candidate web services, as well as their dependencies. On the one hand, the structural metric evaluates how effective is the link between each pair of dependent services. On the other hand, the functional metric evaluates how similar are the specification and the implementation of web services.

In order to measure the structural metric, it is necessary to evaluate how effective is the integration between the required interface of the requester service and the provided interface of the provider service. Figure 4 characterizes a link, including associated services and interfaces, which together define all entities to be considered in measuring the structural metric.

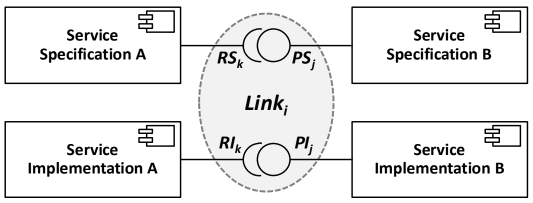

Figure 4. Characterization of a Link

As can be observed, each link is characterized in terms of two interfaces in the architecture specification and two interfaces in the candidate architecture configuration: RS_i - required interface of the requester service specification; PS_i - provided interface of the provider service specification; RI_i - required interface of the requester service implementation; and PI_i - provided interface of the provider service implementation.

Taking into account such interfaces, it is important to note that the greater the number of operations in common in such interfaces the better the integration effectiveness. Consequently, as indicated in (1), the value of the structural metric for a link can be defined by the relation between the number of operations in common in the related interfaces and the total number of operations in the required interfaces of both the requester service specification and implementation. As defined, the value of the structural metric for a link is in the interval [0, 1], where the closer to 1 is the value, the better is the integration effectiveness, and so, the lower is the adaptation effort.

$$L_i = \frac{|(RS_i \cap PS_i) \cap (RI_i \cap PI_i)|}{|(RS_i \cup RI_i)|} \qquad (1)$$

As can be observed in (1), the denominator includes operations in required interfaces only. The reason for that is the premise adopted in the proposed approach which states the following: *superfluous operations in provided interfaces do not represent extra adaptation effort*. In other words, non-used provided operations in the provider service do not impose adaptation effort in the requester service.

Now, considering all links in the architecture specification, as indicated in (2), the value of the structural metric for the whole architecture is defined by the relation between the total sum of the structural metric for each link and the total number of links in the architecture (*L*). Thus, the value of the structural metric for the architecture is also between [0, 1], where the closer to 1 the value, the better the candidate architectural configuration.

$$A_x = \sum_{i=1}^{L} \frac{L_i}{L} \qquad (2)$$

Unlike the structural metric that evaluates dependencies among services, the functional metric contrasts web service specifications against their correspondent candidate implementations, evaluating their similarity in terms of provided and required interfaces. In other words, a candidate service implementation imposes a lesser amount of adaptation effort when its provided and required interfaces are more similar in relation to the corresponding interfaces in the service specification.

In order to measure the functional metric for a given service, as illustrated in Figure 5, it is necessary to evaluate the functional metric for each provided and required interface of the service.

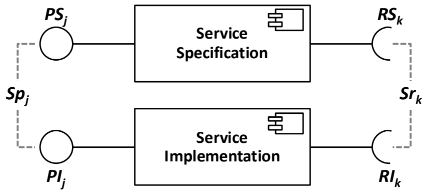

Figure 5. Characterization of Similarity

Considering correspondent provided interfaces in the service specification and implementation, it is important to note that the greater the number of operations in common in such interfaces the better the integration effectiveness. Thus, as indicated in (3), the value of the functional metric for a given provided interface can be calculated by the relation between the number of operations in common in the provided interfaces in the service specification and implementation divided by the number of operations in the provided interface in the service specification. Here, once again, the proposed approach assumes that superfluous operations in provided interfaces do not represent extra adaptation effort, and so, the denominator in (3) does not consider operations in the provided interface in the service implementation.

$$Sp_j = \frac{|PS_j \cap PI_j|}{|PS_j|} \quad (3)$$

Now, considering correspondent required interfaces in the service specification and implementation, the greater the number of operations in common in such interfaces the better the integration effectiveness. Thus, as indicated in (4), the value of the functional metric for a given required interface can be calculated by the relation between the number of operations in common in the required interfaces divided by the total number of operations in such interfaces conjointly.

$$Sr_k = \frac{|RS_k \cap RI_k|}{|RS_k \cup RI_k|} \quad (4)$$

Equations (3) and (4) evaluate individually each provided and requited interface in a given service. As defined, the values of the functional metrics Sp_j and Sr_k are also in the interval [0, 1], where the closer to 1 the value, the better the provided or required interface.

Now, it is needed to derive the functional metric for the service as a whole, revealing how similar are provided and required operations in the service specification and implementation. Thus, considering all provided and required interfaces of a given service specification, as indicated in (5), the value of the functional metric for the service is defined by the relation in which the numerator is the total sum of the functional metric for each required and provided interface of the service, while the denominator is the total number of required and provided interfaces of the service. As defined, the value of the functional metric for a given service is also in the interval [0, 1], where the closer to 1 the value, the better the candidate web service.

$$S_i = \frac{\sum_{k=1}^{|RS \cup RI|} Sr_i + \sum_{j=1}^{|PS|} Sp_j}{|RS \cup RI| + |PS|} \quad (5)$$

As can be seen in (5), in terms of required interfaces, the functional metric comprises the number of required interfaces in both the service specification and implementation conjointly ($|RS \cup RI|$). However, in terms of provided interfaces, the functional metric for the service comprises the number of provided interfaces in the service specification only ($|PS|$). Note that, once more, it is supposed that superfluous provided interfaces in the service implementation do not represent extra adaptation effort, and so, the terms in (5) do not account for provided interfaces in the service implementation (PI).

At this point, considering all candidate services in the architecture configuration, as indicated in (6), the value of the functional metric for the whole architecture is defined by the relation between the total sum of the functional metric for each service and the total number of services in the architecture (S). Thus, the value of the structural metric for the architecture is also between [0, 1], where the closer to 1 the value, the better the candidate architectural configuration.

$$C_x = \sum_{i=1}^{S} \frac{S_i}{S} \quad (6)$$

Finally, functional and structural metrics should be combined together in order to derive the fitness function adopted in the metaheuristic search technique, more specifically a genetic algorithm. In such a direction, the fitness function is defined in (7) as a normalized weighted mean of the functional and structural metrics, in which the terms w_c and w_a represent their respective normalized weights. As can be noticed, the value of the fitness function is in the interval [0, 1], where the closer to 1 the value, the better the candidate architectural configuration in terms of adaptation effort.

$$F_x = w_c.C_x + w_a.A_x \begin{cases} 0 \leq w_c \leq 1 \\ 0 \leq w_a \leq 1 \\ w_c + w_a = 1 \end{cases} \quad (7)$$

V. Experimental Evaluation

In order to conduct an experimental evaluation, the proposed approach was implemented in the Java platform. In such experiments, the genetic algorithm is parametrized as follows. For each generation, the population is equal to 300 candidate architecture configurations. The stopping criterion is reached when the highest ranking solution's fitness becomes stable in a plateau during 25 successive iterations and no longer produce better results. The selection of candidate solutions to breed a new generation is based on the tournament method. For breeding a next generation, the uniform crossover method is adopted, together with a mutation ratio of 20%. Finally, the normalized weights w_c and w_s included in the fitness function adopt both the value 0,5, representing an equal contribution for the functional and structural metrics.

The experimental evaluation was performed using a typical architecture specification composed of five web service specifications and six dependencies among them. Each web service specification has 30 candidate implementations, generating a search space size equal to 30^5. The evaluation takes place in three different scenarios, varying the number of specifications that have perfect implementations: *(i)* all specifications with perfect candidates; *(ii)* three specifications with perfect candidates; and *(iii)* absence of perfect candidates. Such scenarios make possible to evaluate the proposed approach in the presence or absence of perfect candidates, including something in the middle.

For each scenario, the proposed approach was compared against the exhaustive search and the random search. In such a comparison, the proposed approach and the random search have been executed 1000 times, and the mean value of the highest ranking solution's fitness is computed. Besides, the exhaustive search has been executed just one time for discovering the optimal solution. As show in Figure 6, experimental results reveal that, in all scenarios, the proposed approach has always found the optimal solution, which is confirmed by the exhaustive search. Due to that, in such experiments, standards deviations are equal to zero, and so, confidence intervals are not estimated.

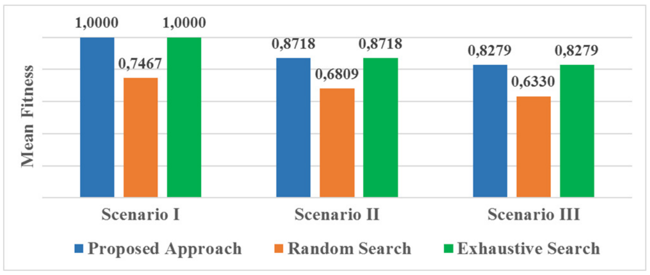

Figure 6. Experimental Results

Besides, in relation to random searches, in the first scenario, in which all specifications have perfect candidates, the solutions recommended by the proposed approach are around 33,92% more efficient than those recommended by random searches, according to the fitness function in (7). In the second scenario, in which three specifications have perfect candidates, the efficiency of the proposed approach in relation to random searches is reduced to approximately 28,04%. Finally, in the last scenario, in which there is an absence of perfect candidates, the efficiency of the proposed approach becomes stable around 30,79%. It is important to stress that, in the best cases, the efficiency ratios turned to 50,56%, 42,85% and 51,35%, respectively.

As another interesting outcome, it must be highlighted the low processing cost of the proposed approach, which can be perceived by its fast convergence around 0,3 seconds, against the exhaustive search that takes around 300 seconds. In all scenarios, the genetic algorithm has converged on average between 5 and 7 generations, ranging from 2 generations in the best cases to 19 generations in the worst cases.

VI. CONCLUDING REMARKS

Considering the relevance of web service selection in the context of SOA-based software development, this paper represents an interesting contribution by presenting an automated approach based on functional and structural metrics. The approach provides measures that evaluate the functionalities provided by candidate web services, as well as their dependencies at the architectural level. Besides, the approach proposes a heuristic selection algorithm based on Genetic Algorithms, which has low processing cost and mitigates the chances of suggesting a local optimum.

Despite their key contributions, previous work has largely been concerned with non-functional requirements. Differently, the proposed approach deals with functional and structural properties, which clearly represents its main contribution. As an additional contribution, the proposed approach represents an extremely complex problem in a systematic and structured way, discovering good-enough or even optimal solutions among candidate web services.

Experimental outcomes demonstrate the effectiveness of the proposed approach not only in terms of the quality of the recommend solutions, but also in terms of low processing cost in all evaluated scenarios. Despite contributions and benefits, as future work, the proposed approach needs to be evaluated in more complex scenarios, composed by a large number of highly interconnected services. It is important to note that, in such future experiments, the expectation is to find more interesting results, once that, generally, metaheuristic-based approaches can find better results in contrast with random search in scenarios with large search spaces.

ACKNOWLEDGMENT

This work was supported by the National Institute of Science and Technology for Software Engineering (INES – www.ines.org.br), funded by CNPq, grants 573964/2008-4.

REFERENCES

[1] I. Sommerville, "Software engineering", 9th edition, Addison-Wesley, 2011.

[2] OASIS, “Reference model for service oriented architecture 1.0”, Committee Specification 1, 2006.

[3] Q. H. Mahmoud, “A service-oriented architecture (SOA) and web services: the road to enterprise application integration (EAI)”, 2005. http://www.oracle.com/ technetwork/articles/ javase/soa-142870.html [retrieved: May, 2016].

[4] S. Becker, A. Brogi, I. Gorton, S. Overhage, A. Romanovsky, and M. Tivoli, “Towards an engineering approach to component adaptation”, Springer-Verlang. 2006.

[5] G. Feng, C. Wang, and H. Li, “Web services based cross-organizational business process management”, 7th Asia-Pacific Web Conference, 2005, pp. 548-559.

[6] M. Harman and B. F. Jones, “Search-based software engineering”, Information and Software Technology, vol. 43, 2001, pp. 833-839.

[7] M. Harman, “Why the virtual nature of software makes it ideal for search based optimization”, 13th International Conference on Fundamental Approaches to Software Engineering, 2010, pp. 1-12.

[8] R. Linden, “Genetic algorithms: an important tool for computational intelligence”, 2nd edition, Brasport, Rio de Janeiro, 2008 (in portuguese).

[9] H. Fetthallah, M. A. Chikh, and D. Y. Mohammed, "QoS-aware service selection based on genetic algorithm", 3rd International Conference on Computer Science and its Applications, 2011, pp. 291-300.

[10] A. Lifeng and M. Tang, "A penalty-based genetic algorithm for QoS-aware web service composition with inter-service dependencies and conflicts", 3rd International Conference on Computational Intelligence for Modelling Control and Automation, 2008, pp. 738-743.

[11] A. Vescan, “A metrics-based evolutionary approach for the component selection problem”, 11th International Conference on Computer Modeling and Simulation, 2009, pp. 83-88.

[12] Z. Maamar, Q. Z. Sheng, and B. Benatallah, “Selection of web services for composition using location of provider hosts criterion”, CAiSE Workshops, 2003, pp. 67-76.

[13] Q. C. Tang and H. K. Cheng, “Optimal location and pricing of web services intermediary”, Decision Support Systems, vol. 40, issue 1, 2005, pp. 129-141.

[14] J. Cheesman and J. Daniels, "UML components: a simple process for specifying component-based software", Addison-Wesley, 2001.