Journal of Ambient Intelligence and Humanized Computing (2022) 13:849–865
https://doi.org/10.1007/s12652-021-02938-y

ORIGINAL RESEARCH

Req-WSComposer: a novel platform for requirements-driven composition of semantic web services

Maha Driss[1,2] · **Safa Ben Atitallah**[1] · **Amal Albalawi**[2] · **Wadii Boulila**[1,2]

Received: 12 February 2020 / Accepted: 28 January 2021 / Published online: 17 February 2021

Abstract

Service-Oriented Computing (SOC) describes a specific paradigm of computing that utilizes Web services as reusable components in order to develop new software applications. SOC allows distributed applications to work together via the Internet without direct human intervention. In this work, we propose a new SOC-based approach to ensure application development. This approach ensures the discovery, selection, and composition of the most appropriate Web services. With this approach, various requirements (both functional and non-functional) are specified by the developer to satisfy QoS, QoE, and QoBiz parameters and Web services are selected and composed to meet these requirements. Our approach is implemented using the Req-WSComposer (Requirements-based Web Services Composer) platform, whose functionalities are tested using an extended and enriched version of the OWLS-TC dataset, which includes around 10,830 semantic Web services descriptions. The results of our experiments demonstrate that the proposed approach enables users to extract the most appropriate composition solution that satisfies the developer's pre-determined requirements.

Keywords Service-oriented computing · Web services composition · Quality of service (QoS) · Quality of experience (QoE) · Quality of business (QoBiz)

1 Introduction

The emergence of Service-Oriented Computing (SOC) (Papazoglou and Van Den Heuvel 2007; Papazoglou et al. 2010) characterized a momentous shift in the history of the Internet. Prior to SOC, the Internet was imagined primarily as a vector enabling various forms of data exchange. With SOC, though, the Internet began moving toward an open platform that supported Web services—software components that were self-described, loosely coupled, and easily integrated with one another. The main goal of Web service technologies is to permit distributed applications to interoperate together using the available standardized Internet protocols and languages and without direct human intervention (Papazoglou 2012). Through the opportunities for inter-operability that they offer, Web services are now a central focus for multiple technological and industrial actors in different fields ranging from e-commerce to e-learning, e-government, and more (Papazoglou and Van Den Heuvel 2007; Papazoglou 2012).

To ensure the development of distributed software applications, SOC depends on the Service Oriented Architecture (SOA) and its associated standards (Papazoglou et al. 2010). SOA standardization process is based on three layers of basic infrastructure: a communication protocol, a description specification, and ultimately, publication and location specifications (Curbera et al. 2002). SOA is a means of structuring and reorganizing distributed software applications into a set of composed and interactive pre-existing services. Web services composition (Sheng et al. 2014) is the most attractive opportunity offered by SOC and SOA, since it presents real competitive advantages for several technological and industrial actors by offering them the possibility to ensure quick, and low-cost development of distributed and collaborative software applications. The service composition lifecycle (Sheng et al. 2014) consists of collecting and assembling autonomous Web services to achieve new functionalities by creating complex, value-added service-based applications. This lifecycle begins

✉ Wadii Boulila
wadii.boulila@riadi.rnu.tn

1 RIADI Laboratory, National School of Computer Sciences, University of Manouba, Manouba, Tunisia

2 IS Department, College of Computer Science and Engineering, Taibah University, Medina, Saudi Arabia

with the requirements specification phase, followed by the discovery and selection of the services that closely fit the developer's requirements, and finally concluding with the orchestration/choreography of the selected services. Functional requirements are implemented by the operations provided by Web services, while non-functional requirements can be categorized into three categories of parameters (Metzger et al. 2010): the objective, the subjective, and the business-related. These parameters in turn influence the Quality of Service (QoS) (Kritikos and Plexousakis 2009; Metzger et al. 2010) Quality of Experience (QoE) (Van Moorsel 2001; Bocchi et al 2016), and Quality of Business (QoBiz) (Van Moorsel 2001; Aljazzaf 2015), respectively.

In the literature, many studies have addressed the problem of Web service composition by offering different languages/specifications (e.g., BPEL (Alves et al. 2007), WS-CDL (Kavantzas et al. 2005), and BPEL4Chor (Decker et al. 2007)), and formalisms (e.g., Petri nets (Shijie et al. 2020; Zhou et al. 2020), timed automata (Hammal et al. 2020; Siavashi et al. 2016), and process algebras (Rai et al. 2015; Zhu et al. 2017)). Each of these works adopts a function-centered vision for the composition lifecycle. Concerning non-functional requirements, existing works focus on the different parameters (i.e., objective, subjective, and business-related) separately or neglect coupling them with functional requirements. However, in order to ensure the efficient development of service-based applications and to enhance the satisfaction of the various collaborating actors, it is essential to adopt a coupling approach that involves both functional and non-functional requirements.

In this work, our goal is to develop a new requirements-driven approach that will ensure the discovery, selection, coordination, and execution of the most appropriate Web services available. The discovery of Web services will consider functional requirements and the selection will be based on the three parameters of non-functional requirements discussed above (objective, subjective, and business-related). The proposed approach will be implemented using the Req-WSComposer (Requirements-based Web Services Composer) platform in order to satisfy developer requirements by building effective and high-quality service-based applications. Our approach is tested using an extended and enriched version of the OWLS-TC dataset, which is composed of thousands of Web services.

The remainder of our paper is organized in the following manner. In Sect. 2, the relevant literature is reviewed and several gaps are identified. Next, in Sect. 3, a synopsis of the proposed approach is presented. Then, in Sect. 4, the experimentations and results are discussed. Finally, Sect. 5 includes our conclusions and thoughts on possible directions for future work.

2 Related works

Automatic services composition is considered an open research field involving various theories, techniques, and standards (Sheng et al. 2014; Garriga et al. 2015). In the literature, many attempts have been conducted focusing on composition lifecycle phases separately [i.e., requirements specification (Zolotas et al. 2017), discovery (Cheng et al. 2016), selection (Azmeh et al. 2011; Bagga et al. 2019), and coordination and execution (Fadhlallah et al. 2017)] without necessarily addressing problems or proposing solutions related to the implementation of the composition lifecycle as a whole. Our goal in this work is to provide a holistic approach ensuring the alignment between the different phases of the Web services composition lifecycle in order to obtain a new service-based application satisfying both functional and non-functional requirements as specified in the first phase of the lifecycle. In the following sections, we will briefly discuss related research works that focused on Web services composition.

In (Driss et al. 2010, 2011a), the authors present an approach using the MAP formalism and the Intentional Service Model (ISM) to elicit and specify which requirements a user would request, both functional and non-functional. In these works, authors adopt a formal framework called the Formal Concept Analysis (FCA) to ensure that relevant and high-QoS Web services will be selected. Compared to our approach, these works focus only on objective parameters related to the service quality, which are described in terms of QoS properties, and they neglect subjective and business-related quality parameters. Also, the authors, in (Driss et al. 2010, 2011a), test their proposed approach on pure syntactic WSDL-based services and omit semantically described services, which are more convenient for ensuring effective discovery and selection of composable Web services.

In (Aznag et al. 2013), the authors propose a Web services discovery and recommendation system to help customers find services that match their various requirements, both functional and non-functional. Here, the most important new concept is the application of Rules-Based Text Tagging (RBTT) to generate a new Web services representation, which omits insignificant information. This work also proposes another Web service representation, called Symbolic Reputation (SR), which describes the service in its context (i.e., relationships with neighbors). The obtained representations are used to study and discuss their impact of use on Web service discovery and recommendation. The proposed approach is experimented with using single WSDL-based Web services to perform simple operations such as predicting weather conditions, sending/receiving SMS, and providing address information.

However, the experimentation conducted in this paper does not show composition scenarios. Besides, only QoS and reputation properties are considered by the proposed discovery and selection algorithm.

In (De Castro et al. 2014), the π-SODM platform is presented as an extension of the Service-Oriented Development Method (SODM) proposed in (De Castro et al. 2009). This platform supports Web services composition according to both functional and non-functional requirements. The π-SODM includes non-functional specifications through four metamodels: π-UseCase, π-ServiceProcess, π-ServiceComposition, and π -PEWS (De Castro et al. 2014). Here, non-functional requirements are described as being constraints that dictate processing and data, particularly as specified by pre-determined rules and conditions that must be verified during task execution. Two sets of rules are proposed in this work: the first one is formed by "model-to-model" rules of transformation, which are used to alter "platform-independent" models into new "platform-specific" models, and the second one is used to transform the resulting "platform-specific" models into workable implementations. The main limitation of this work is that only QoS parameters are considered in the composition lifecycle and specifically constraints related to security and privacy.

In Suchithra and Ramakrishnan (2015), Suchithra and Ramakrishnan propose a new method allowing the ranking of Web services by computing a relevancy function. According to this function, the selected Web services are ranked and ordered. The proposed method allows for finding the best available Web services for a given user request. Six parameters are considered to rank services, which are throughput, response time, accessibility, availability, interoperability, and cost. Experiments are conducted using a single Web service to validate the e-mail address. Furthermore, these experiments are carried out on pure syntactic WSDL-based services. The coordination and execution phases of the composed services are not presented in this paper.

Rodriguez-Mier et al. (2015) proposed a semantic Web service discovery integrated with a composition framework. The resulting combined framework relies on an analysis of graph-based service compositions in order to discover the optimal services that semantically match in terms of input/output parameters. The proposed framework also integrates a search algorithm that focuses on optimal compositions, which allows for the extraction of a graph's best composition, thus minimizing both the length and number of Web services that are composed. Experimental results provided in this paper demonstrate the strong capabilities and performance of the proposed framework. Nevertheless, there are still limitations, which include: (1) the selection phase is merged with its discovery counterpart and is performed upon fine-grained input/output queries, (2) the formulation of non-functional requirements is missing, all required information needed to conduct the discovery phase are sets of input and output parameters, and finally (3) a validation in term of user's satisfaction is not provided.

Work (Bekkouche et al. 2017) proposed an integrated framework for automated semantic Web services composition, this time with greater QoS awareness. Bekkouche et al., use a "Harmony Search" (HS) algorithm to select the optimum solution for a Web services composition. In this work, the selected solutions are optimized in line with a set of non-functional parameters, which include cost, availability, reliability, reputation, and response time. In this paper, service discovery is performed by computing a semantic matching score between the input and output parameters of services that are involved in the composition solution. This semantic matching improves the discovery and selection of relevant services, but it remains insufficient since it is performed on WSDL specifications, which are purely technical declarations.

In Khanouche et al. (2019), the authors propose a QoS-aware services composition algorithm based on clustering, which is able to reduce the composition time while ensuring composition optimality. The algorithm starts by grouping the candidate services into different clusters using the k-means method. Every cluster defines a QoS level as either high QoS, middle QoS, or low QoS. Five QoS attributes have been utilized including; response time, throughput, availability, reliability, and cost. Using a new formulation of the utility function, the unpromising candidate services are eliminated. The next step is to filter the candidate services and remove the ones with low QoS by exploiting the lexicographic optimization method. Finally, using the services that satisfy the QoS requirements, a search tree is created to select the optimal composition. Comparing to other approaches, the proposed algorithm obtains better results, by finding an optimal solution in less time. Although the fast execution time of this algorithm, it does not address the development of new value-added service-based applications.

In Sangaiah et al. (2019), Sangaiah et al. develop a novel approach for Web services composition based on the Biogeography-based optimization algorithm (BBO) taking into account the user's QoS constraints. The proposed algorithm uses evolutionary optimization to choose the best Web services in order to obtain a composition with good efficiency and high accuracy. The BBO algorithm is employed to optimize the services discovery phase by a repetitive improvement of the candidate composition and with respect to the QoS features and fitness function. Two different categories of QoS features are introduced: (1) the positive features including availability and reliability, and (2) the negative features including cost and response time. Experiment results demonstrate the efficiency of the presented algorithm. Compared to other methods, the selected composition of services achieves the best QoS values. However, the limitations of this study include the absence of subjective non-functional requirements and the absence of validation in terms of user satisfaction.

In Khanouche et al. (2020), the authors present a Flexible QoS-aware Services Composition (FQSC) algorithm that helps to increase the services composition feasibility, reduce the composition time, and ensure the composition optimality. FQSC algorithm consists of three main phases. It starts by decomposing the global constraints of QoS, then discovering the candidate services according to the user's QoS requirements, and finally selecting the near-to-optimal services composition. Using a real dataset, the performance of the proposed algorithm has been tested in different scenarios of service composition. The obtained results demonstrate how the algorithm reduces the time of getting an optimal service composition. However, only three QoS attributes for each service are considered in the composition scenarios.

Driss et al. (2020) propose an approach based on Formal Concept Analysis (FCA) and Relational Concept Analysis (RCA) to compose semantic Web services. Services-based applications are built by selecting Web services, which provide optimal quality properties. These properties are related to QoS, QoE, and QoBiz. The proposed selection approach is semi-automated since the construction of contexts serving the FCA and RCA techniques is performed manually. This fact doesn't allow to measure the effectiveness of the proposed selection approach in terms of the response time of execution. Also, in this work, no alternative composition solutions are provided in case there is no matching between the user's requirements and the existing discovered services.

In Rodríguez et al. (2020), the authors present an approach allowing to estimate the values of the missing QoS attributes of candidate services to ensure an optimal composition as a result. To this end, the multivariate linear regression technique is explored. The evaluation of the proposed approach is carried out by considering 9 QoS attributes (e.g., response time, compliance, best practices, documentation, latency, etc.) on a dataset consisting of more than 2500 services. The missing values of the considered QoS attributes are calculated using the Soft-Audit tool, which performs statistical analysis on the service interface to estimate its quality and complexity. This work focus only on QoS attributes to ensure the composition of the optimal services. Moreover, the experiments that are presented in Rodríguez et al. (2020) are carried out on purely WSDL-based services, while semantic services are omitted.

Hu et al. (2020) present a trustworthy Web service composition and optimization framework called TWSCO. This framework is proposed to ensure the trust of the composite services and the efficiency of the composition process. In this work, the trust-based optimization problem is influenced by 3 different factors, which are: the trust of the component services, the trust of the interacting behaviors, and the optimal binding schema obtained by composing the optimal candidate services. Different QoS attributes are taken into consideration to evaluate the trust of composite services such as duration, reliability, and availability. Several limitations can be distinguished in this work: (1) conducted experiments are focusing on purely syntactic WSDL-based services that are discovered by using a public search engine, (2) the proposed framework is tested only on sequential or simple composition topologies, and finally (3) the sets of candidate services are relatively small.

Table 1 depicts a comparison of the literature discussed above. This comparison is conducted based on seven central criteria, which include: (1) the consideration of objective non-functional requirements, (2) the consideration of subjective non-functional requirements, (3) the consideration of business-related non-functional requirements, 4) the techniques used to verify user satisfaction, (5) the identification of alternative composition paths fulfilling the user's functional requirements, (6) the development, if any, of new value-added service-based applications, and finally, (7) the platform used to ensure the development of Web services compositions.

Following the analysis laid out in Table 1, our work aims to satisfy all of the criteria mentioned here through the proposal of the Req-WSComposer platform. This platform allows developing new, optimal, and value-added service-based applications. Req-WSComposer offers the following advantages:

1. the modeling of functional and non-functional user's requirements by using ontological descriptions;
2. the discovery of the semantic services that satisfy user's functional requirements by applying a four filters-based matching algorithm;
3. the selection of optimal candidate services by considering various types of non-functional requirements specified by objective, subjective, and business-related parameters;
4. the suggestion of alternative composition paths by computing the related fitness and penalty scores;
5. the verification of the user's satisfaction by computing the accuracy, the precision, and the response time of the discovery algorithm and by providing the optimal matching degrees of the selected composite services.

3 Proposed approach

In this section, we present our approach allowing to developing services-based applications by considering the user's requirements, both functional and non-functional. This approach, which is implemented using the Req-WSComposer platform, consists of four phases: (1)

Table 1 Comparison of the relevant literature

Criteria	Driss et al. (2010), Driss et al. (2011a)	Aznag et al. (2013)	De Castro et al. (2014)	Suchithra and Ramakrishnan (2015)	Rodriguez-Mier et al. (2015)	Bekkouche et al. (2017)	Khanouche et al. (2019)	Sangaiah et al. (2019)	Khanouche et al. (2020)	Driss et al. (2020)	Rodríguez et al. (2020)	Hu et al. (2020)	Our approach
Objective non-functional requirements: QoS	√	√	√	√	×	√	√	√	√	√	√	√ Trust related QoS	√
Subjective non-functional requirements: QoE	×	×	×	×	×	√	×	×	×	√	×	×	√
Business non-functional requirements: QoBiz	×	×	×	√	×	√	√	√	√	√	×	×	√
User satisfaction verification technique	√ recall and precision functions, QoS monitoring technique	√ recommendations, Web services ranking	×	√ relevancy function	×	×	√ execution time, composition optimality, and composition space	×	√	√ recall, precision, and QoS monitoring	×	×	√ Recall, precision, response time of execution, and degrees of matching of composite services
Identification of alternative composition paths	×	×	×	×	×	√ Computation of fitness and penalty functions	×	×	composition time, utility value, and POC size	×	×	×	√ Computation of fitness and penalty functions

Table 1 (continued)

Criteria	Driss et al. (2010), Driss et al. (2011a)	Aznag et al. (2013)	De Castro et al. (2014)	Suchithra and Ramakrishnan (2015)	Rodriguez-Mier et al. (2015)	Bekkouche et al. (2017)	Khanouche et al. (2019)	Sangaiah et al. (2019)	Khanouche et al. (2020)	Driss et al. (2020)	Rodríguez et al. (2020)	Hu et al. (2020)	Our approach
Development of new value-added service-based applications	×	×	√	×	√	×	×	×	×	√	√	√	√
Platform ensuring the development of Web services compositions	×	√	√	×	√	×	×	×	×	×	×	×	Req-WSComposer

requirements specification phase, (2) discovery phase, (3) selection phase, and (4) coordination and execution phase. Figure 1 provides an overview of our approach complete with its different phases.

3.1 Requirements specification phase

The first phase in our approach is the requirements specification phase. During this phase, the user identifies a set of functional and non-functional requirements that the final product must fulfill. As briefly described above, the desired non-functional requirements are categorized into three categories of parameters: objective, subjective, and business-related, which in turn influence the Quality of Service (QoS), the Quality of Experience (QoE), and the Quality of Business (QoBiz) respectively. Considering non-functional requirements in the composition lifecycle will produce effective and value-added services-based applications.

QoS (Metzger et al. 2010) is defined as a set of parameters describing the behavior of Web services in terms of performance parameters. Among these parameters, we can cite response time, availability, scalability, and robustness. QoS parameters can be grouped into two categories: measurable and non-measurable. In this paper, we consider three measurable QoS parameters: availability, throughput, and response time.

- Availability: describes a Web service capacity in terms of execution and use (D'Mello and Ananthanarayana 2009).
- Response time: describes the total time required to dispatch a service request and obtain the service's response (D'Mello and Ananthanarayana 2009).
- Throughput: describes the maximum number of services that the client can use in a specified time with a successful response (D'Mello and Ananthanarayana 2009).

QoE (Bocchi et al 2016) is a measure of the end-to-end performance of a whole system as both resulting and taken from the user's point of view. Therefore, QoE is an indicator of how the system satisfies user needs.

To enhance the composition lifecycle in our approach, we consider the following QoE parameters:

- Friendliness: defines whether and how the service is clear and easy to use (D'Mello and Ananthanarayana 2009).
- Success rate: describes the percentage of attempts with which a web service completes the requested operation successfully within the specified processing time (D'Mello and Ananthanarayana 2009).
- Reputation: describes whether the service can be trusted to fulfill promised functions; this indicator is obtained from the aggregate of rankings provided by users who have requested such functions from the service (D'Mello and Ananthanarayana 2009).

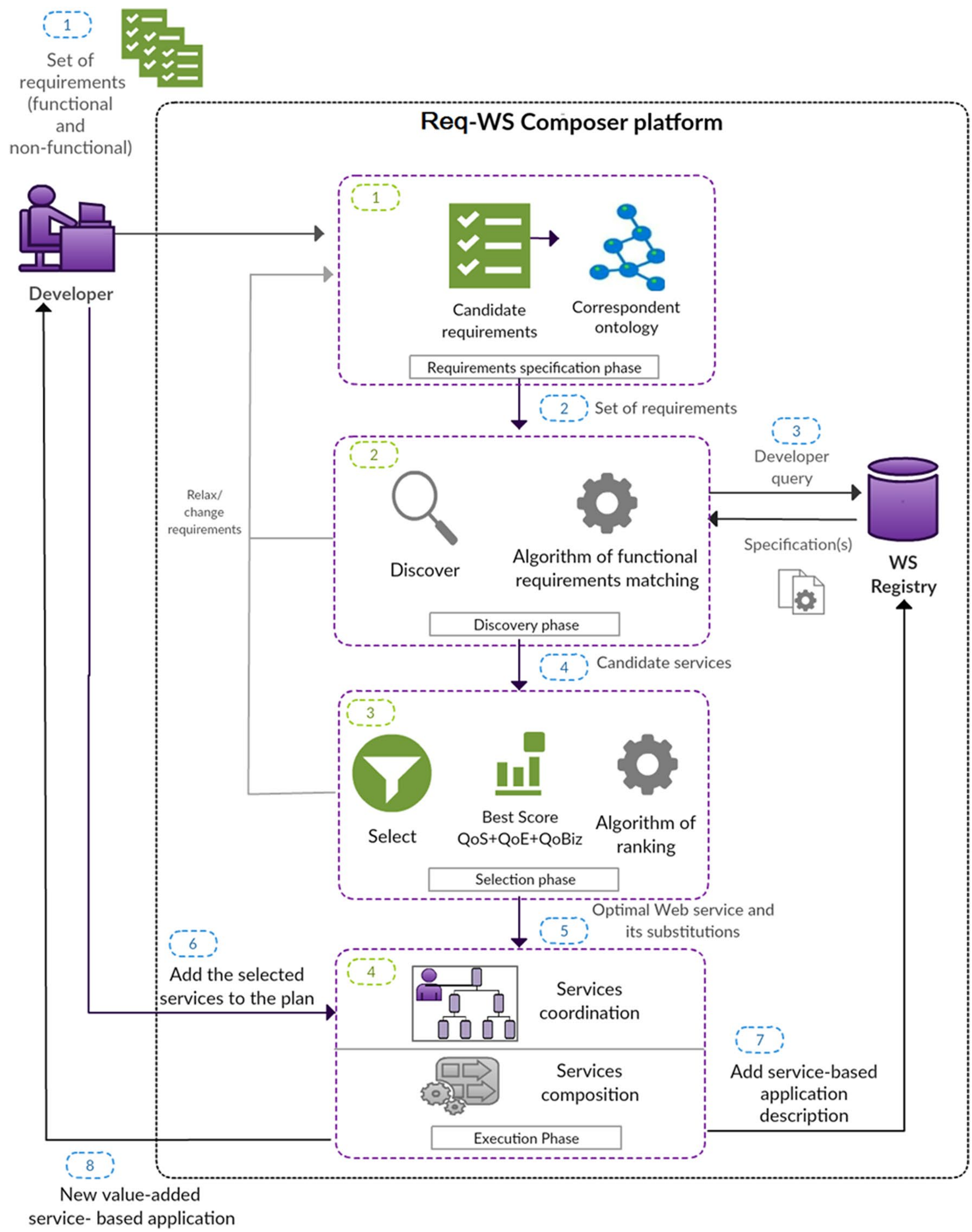

Fig. 1 Proposed approach

Finally, QoBiz (Aljazzaf 2015) parameters describe the financial aspects of service provisioning, such as the price of service, the costs of service provisioning, the service provisioning revenue, and the revenue per transaction. In our work, the cost per transaction is deemed a central QoBiz parameter because it represents the financial requirement(s) of executing each required operation.

To specify functional and non-functional requirements, we use OWL-S, an Ontology Web Language (OWL)-based service ontology intended to define the characteristics and functionalities of Web services. OWL-S is meant to provide a clear description of Web services that allows them to be machine-interpretable. Three main components are used to describe Web services using OWL-S, which are: (1) service grounding, (2) service model, and (3) service profile (Martin et al. 2004). The service grounding is responsible for the protocols, coordinating service usages by mapping with Web services standards like WSDL and SOAP. The service model explains the function, processes, and execution of a Web service. The service profile delineates the service function (i.e., what actions the service can perform and what actions can help in the discovery phase).

The discovery of Web services is mainly performed by considering information in the service profile component. The latter is composed of human-readable parameters, functionality description, and profile attributes. However, non-functional requirements are not considered in this component. Therefore, in our work, we propose extending the OWL-S service profile to include QoS, QoE, and QoBiz parameters. We add a "Non-functional Requirements Description" element that includes the QoS, QoE, and QoBiz parameters and their values. Figure 2 shows an extended OWL-S profile component.

To ensure the discovery of Web services that satisfy the specified requirements, we propose to convert the textual description of initial requirements into a semantic description to be compared with the service profile in the OWL-S file. The requirements specification includes two parts: a functional requirements description and a non-functional requirements description. The first part describes the needed service operation, its input, and its output. The second part includes the required value and priorities of QoS, QoE, and QoBiz parameters. The taxonomy of the requirement's semantic description is depicted in Fig. 3.

3.2 Discovery phase

The services discovery phase aims to search for the appropriate Web services that match with functional requirements specified by the developer. During this phase, we suggest a new semantic matching algorithm to allow for quick and efficient identification of Web services having high-matching profiles with the developer's functional requirements, as specified in the operation name, input parameters, and output parameters.

Figure 4 illustrates the matching between the requirement's semantic description and the OWL-S specifications of candidate services. This matching is carried out as follows:

- The operation name instance in the requirement's description is compared with the service name in the service profile.
- The input instance from the requirement description is compared with hasInput in the service profile.
- The output instance from the requirement description is compared with hasOutput in the service profile.

To perform this phase, we propose a semantic discovery algorithm, which includes four filters. The first filter extracts the Web services from the registry according to the OWL-S file names. The similarity of each word in the requested Web service name and the available OWL-S files in the registry is computed. If the resulted value is more than an empirical threshold, then this OWL-S file will be added to the first list of matched services, which will be passed later to the second filter. The second filter checks the concept names in the domain ontologies described by OWL files. The similarity between a concept name and the service name entered by the user is calculated. If it is greater than the threshold, the Web service will be added to the second filter list, otherwise, the Web service will be omitted. The third filter checks Web services names specified in the profile specification. If it matches the requested service name, it will be added to the third list of matched services. If not, it will be deleted from the list. Finally, the fourth filter compares between desired input and output parameters and existing ones described in OWL-S specifications. In our work, we consider five degrees of matching as commonly explored in the relevant literature (Paolucci et al. 2002). These degrees are listed below:

- `Exact`: the service input and output perfectly matched the request, accounting for the logic-based equivalence of their formal semantics.
- `Subsume`: the input matched with the request, but the output is more specific than the requested one.
- `Plug-in`: the service output matched with the user request. However, the service requires more inputs than what is specified by the user and one of the service inputs matches the requested input.
- `Sibling`: the service requires more inputs than what is specified by the user and one of the service inputs matches with the requested input. The output is more specific than the requested one.
- `Fail`: there is no relatedness between the service output/input and the requested ones.

All services in the third filter list will be checked. If the matching degree is either `Exact`, `Subsume`, `Plug-in`, or `Sibling`, the service will be passed. But, if the matching degree is `Fail`, the service will be eliminated.

The implementation of our semantic discovery method is depicted by algorithm 1.

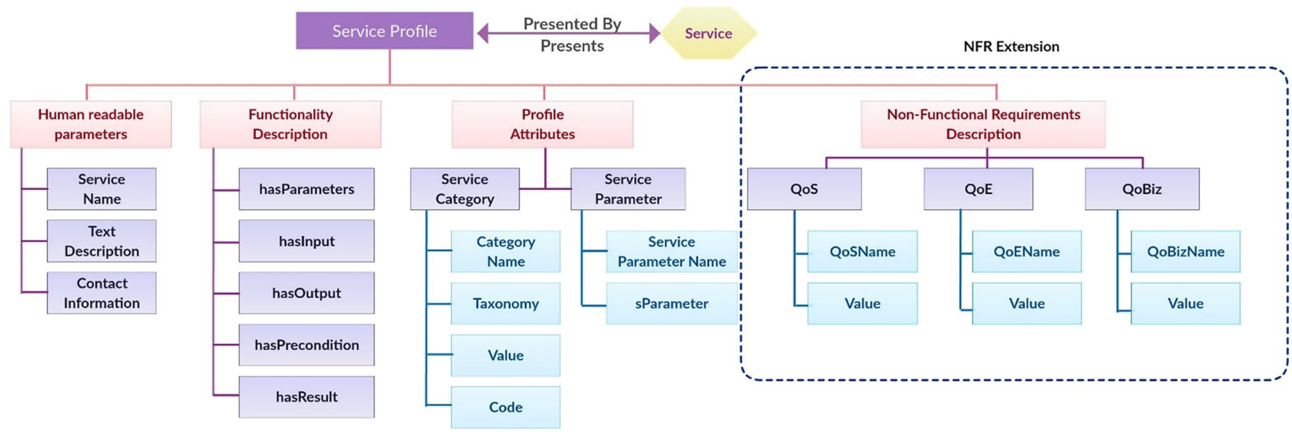

Fig. 2 Extended OWL-S profile integrating non-functional requirements description

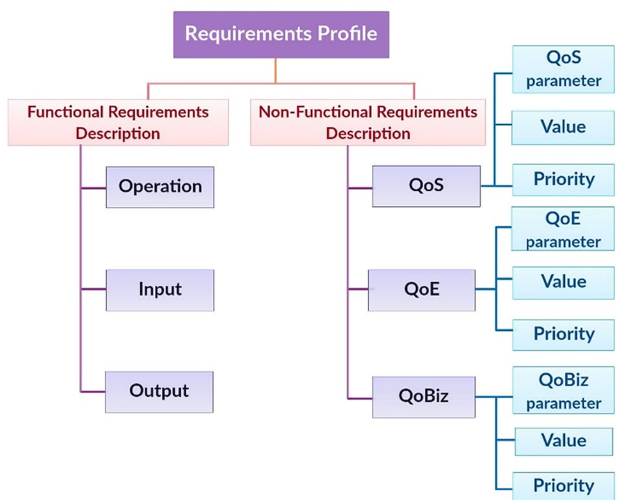

Fig. 3 Requirements' semantic description taxonomy

The discovered services will be ranked based on their non-functional parameters in the next phase. In case there are no services that match the developer's request after applying the discovery phase, the developer has to relax/ change his requirements as it is indicated in Fig. 1.

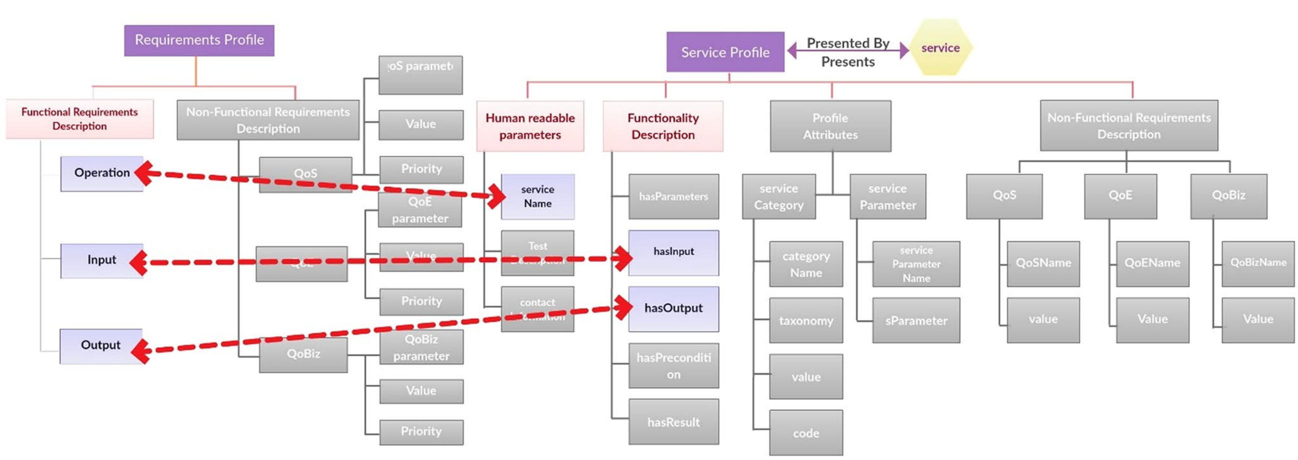

Fig. 4 Functional requirements matching

Algorithm 1: Semantic Discovery

Input: *u_serv*: user service names; *serv_file_names*: list of services' file names; *serv_names:* list of services' names *u_in*: user input; *u_out*: user output; *s_input:* service input; *s_output:* service output; *ont_names*: class names of domain ontology, *th*: threshold

Output: *Filter4List*: list of functional matched services

```
Filter1List, Filter2List, Filter3List, Filter4List: lists of user service names
/* First filter process */
for all f_name in serv_file_names
sim= d(u_serv, f_name)
    if (sim >= th) then
        Filter1List.add(u_serv)
    end if
 end for
/* Start the second filter process */
for all serv in Filter1List
sim=d(serv,ont_names)
    if (sim >= th) then
        Filter2List.add(serv)
    end if
 end for
/* Start the third filter process */
for all serv in Filter2List
sim=d(serv, serv_names)
    if (sim >= th) then
        Filter3List.add(S)
    end if
 end for
/* Start the fourth filter process */
for all serv in Filter3List
   sim_input = d(s_input, u_input)
   sim_output =  d(s_output, u_output)
  if (count(s_input) =1 ) then
      if ((sim_input >=th) and (sim_output >=th)) or ((sim_input >=th) and (u_output ∈ s_output)) then
         Filter4List.add(serv)
      end if
  else if (count(s_input) >1) then
      if ((sim_input >=th) and (count(s_input)=1) and (sim_output >=th)) or ((sim_input >=th) and (count(s_input)=1) and u_output ∈
s_output)) then
          Filter4List.add(serv)
  end if
 end for
```

3.3 Selection phase

The purpose of the services' selection phase is to select the optimal Web services from a set of candidate services returned by the discovery phase. The selection is performed by considering the developer's non-functional requirements, which include QoS (comprised of availability, response time, and throughput), QoE (comprised of friendliness, success rate, and reputation), and QoBiz (comprised of cost per transaction) parameters. For each parameter, the developer specifies a required value and sets a priority. The selection is performed by comparing the required value in the requirements' semantic description with the assigned value in the non-functional requirements' specification of each candidate service. This comparison allows to rank candidate services according to the developer's non-functional requirements requested values and priorities. Figure 5 presents the non-functional requirements matching applied during the selection phase.

To perform this matching, we first scale the values of each quality parameter specified in the non-functional requirements specification. As is suggested in (Zeng et al. 2004), negative parameters (e.g., response time and cost per transaction) are scaled according to Eq. (1), and positive parameters (e.g., availability, throughput, friendliness, success rate, and reputation) are scaled according to Eq. (2).

$$V_{i,j}=\begin{cases} \frac{Q_j^{\max}-Q_{i,j}}{Q_j^{\max}-Q_j^{\min}} & if \quad Q_j^{\max}-Q_j^{\min}\neq 0 \\ 1 \quad if & Q_j^{\max}-Q_j^{\min}=0 \end{cases} \tag{1}$$

$$V_{i,j}=\begin{cases} \frac{Q_{i,j}-Q_j^{\min}}{Q_j^{\max}-Q_j^{\min}} & if \quad Q_j^{\max}-Q_j^{\min}\neq 0 \\ 1 \quad if & Q_j^{\max}-Q_j^{\min}=0 \end{cases} \tag{2}$$

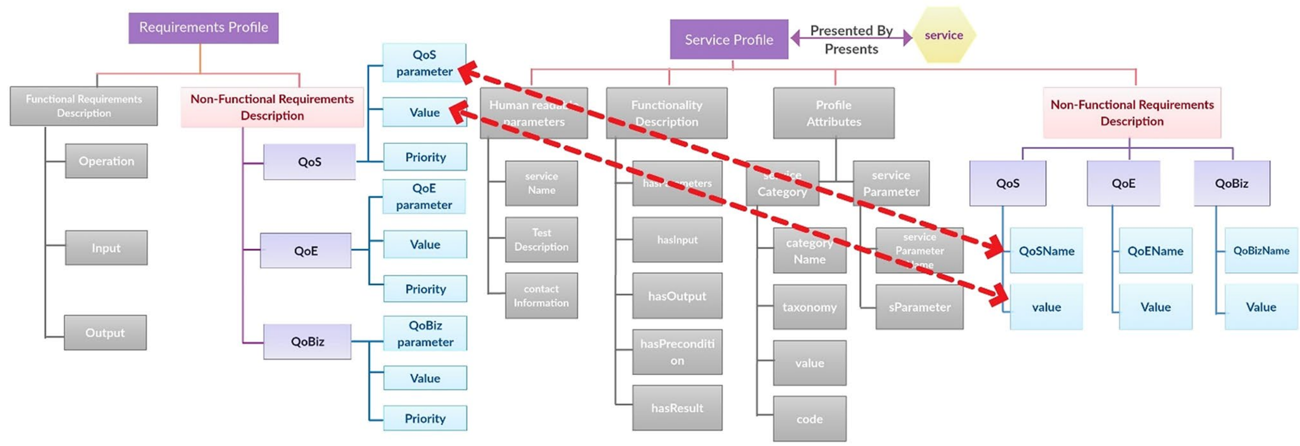

Fig. 5 Non-functional requirements matching

In the above equations, Q_j^{max} is the maximum value of a quality parameter, while Q_j^{min} is the minimum value.

After that, we calculate a fitness function, which allows quantifying the overall quality of each proposed composition solution according to Eq. (3) (Bekkouche et al. 2017).

$$F(\mathrm{sol}) = \sum_{k=1}^{n} P_{Qk} * Q_k \tag{3}$$

where, P_{Qk} is the priority for each quality parameter k specified by the developer, and Q_k are the scaled non-functional requirements values of each parameter.

Composition solutions that do not meet exactly the developer quality constraints and can substitute the optimal composition are penalized using a static function F', as adopted from (Lécué 2009). Equation (4) presents F'.

$$F'(\mathrm{sol}) = F(\mathrm{sol}) - \left(\sum_{k=1}^{n} \left(\frac{\Delta Q}{g_k^{max} - g_k^{min}} \right) \right)^2 \tag{4}$$

where g_k^{max} and g_k^{min} are the maximum and minimum values of quality constraints, respectively, n represents the number or quantity of non-functional requirements constraints, while ΔQ (Yu and Bouguettaya 2009) is defined by the formula (5).

$$\Delta Q = \begin{cases} Q_k - g_k^{max} & if \quad Q_k > g_k^{max} \\ 0 \quad if & g_k^{min} \le Q_k \le g_k^{max} \\ g_k^{min} - Q_k if Q_k < g_k^{min} & \end{cases} \tag{5}$$

At the end of this phase, the composition that has the best fitness with the lowest penalty value is returned to the developer as an optimal solution. In case there are no services that match the developer's request after applying the selection phase, the developer should relax and/or change requirements, as indicated in Fig. 1.

3.4 Composition phase

In this final phase, the services selected to form an optimal composition solution are coordinated and orchestrated/choreographed utilizing an engine that is able to host, execute, and run composite services such as ours using the standardized Web Services Business Process Execution Language (WSBPEL/BPEL) (Alves et al. 2007).

4 Experimentation and Results

The experimentation of the proposed approach is conducted using the OWLS-TC dataset,[1] which includes descriptions of 10,000 + Web services specified with the OWL-S language. This dataset is intended to support evaluations of OWL-S semantic Web service matchmaking algorithms and consists of nine different domains of Web services: education, simulation, medical/healthcare, food/food and beverage services, travel and tourism, communications, finance and economy, weaponry, and finally geography. In this dataset, the collected Web services are described using both OWL-S 1.0 and OWL-S 1.1. For the purpose of our experiment, we selected services that had been described using the most recent OWL-S version, which is OWL-S 1.1.

In order to consider the non-functional properties of these services adequately, we have also enriched the OWL-S files using seven parameters, which include availability (capacity for use), response time (time required for use cycle),

[1] http://projects.semwebcentral.org/projects/owls-tc/.

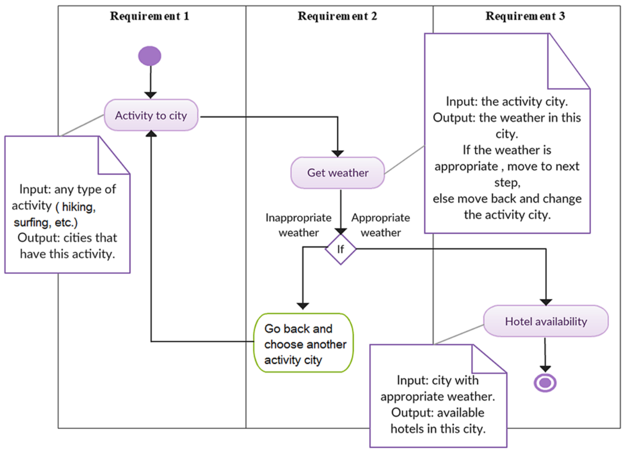

Fig. 6 UML activity diagram describing the trip-planning scenario

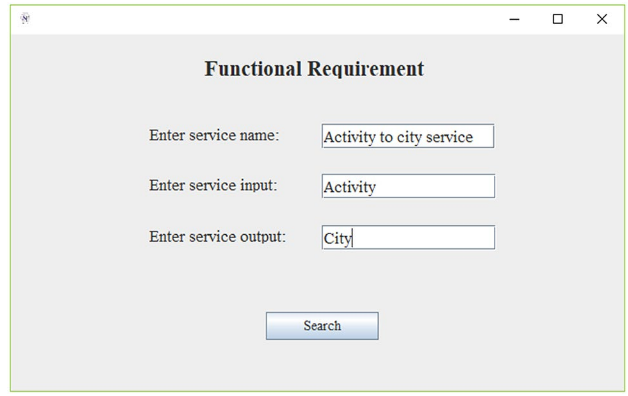

Fig. 7 Req-WSComposer interface illustrating the specification of the "Activity to city" functional requirement

throughput (maximum technical usability), friendliness (user-centric clarity), success rate (percentage of completion), reputation (ability to fulfill promised functions), and cost (financial requirement per transaction). As other researchers have suggested (Yu and Bouguettaya 2009; Driss et al. 2020), values suggesting cost are best set between \$0 and \$30, while values regarding response time are best set between 0 and 300 ms, and all additional parameter values are best set in the range of 70% and 100%. In addition, we also applied a manual pre-treatment on each file name, using the underscore symbol to separate each individual word in the names of each Web service.

4.1 Experimental scenario

In this study, the proposed approach is applied to a trip-planning scenario, which we describe using a UML activity diagram in Fig. 6. The scenario can be performed by a customer planning to organize a trip. In this scenario, the customer might begin by searching for a city that offers the space and resources for a favorite activity (e.g., swimming). Once a city is selected, the customer might then check the weather during the desired travel dates to see whether it would be suitable for swimming or not. Finally, the customer might check hotel availability in the selected city. Considering that these three functions are implemented by different providers' Web services, our goal is to search for the best composition of services and solutions that will satisfy the customer's requirements.

In the following subsections, we detail how to perform our approach phases on the trip-planning scenario, and we present Req-WSComposer interfaces related to each phase.

4.2 Experimental results

During the first phase, the developer specifies his functional and non-functional requirements. In a first step and for the first requirement in the trip-planning scenario, for example, the developer needs to enter the service name, input, and output, as shown in Fig. 7. In a second step, he specifies the required values and priorities of the QoS, QoE, and QoBiz parameters, as illustrated in Fig. 8.

The specified functional and non-functional requirements are converted into an ontological description, which is implemented using Protégé,[2] as specified in Fig. 9. This description includes two main classes: functional requirements and non-functional requirements. The functional requirements class represents the required Web service operations, which are expressed through a set of keywords, including service name, input, and output data. Whereas, the non-functional requirements class consists of QoS, QoE, and QoBiz classes, where the developer's quality preferences and priorities will be saved. The main purpose of this requirement specification is to discover and select the most appropriate Web services that are semantically relevant to the developer's query. The elaborated ontological description is then updated by inserting desired preferences and priorities using the JDOM API of Eclipse. Figure 10 illustrates the ontological description of the first requirement in our trip-planning scenario, which is the "Activity to city" requirement.

During the discovery phase, our semantic matching algorithm is based on the WordNet Similarity for Java (WS4J) library[3], which offers a pure Java API that supports numerous algorithms measuring semantic relatedness or similarity. We use the WuPalmer algorithm (Wu and Palmer 1994) that

[2] https://protege.stanford.edu/.

[3] https://code.google.com/archive/p/ws4j/ws.

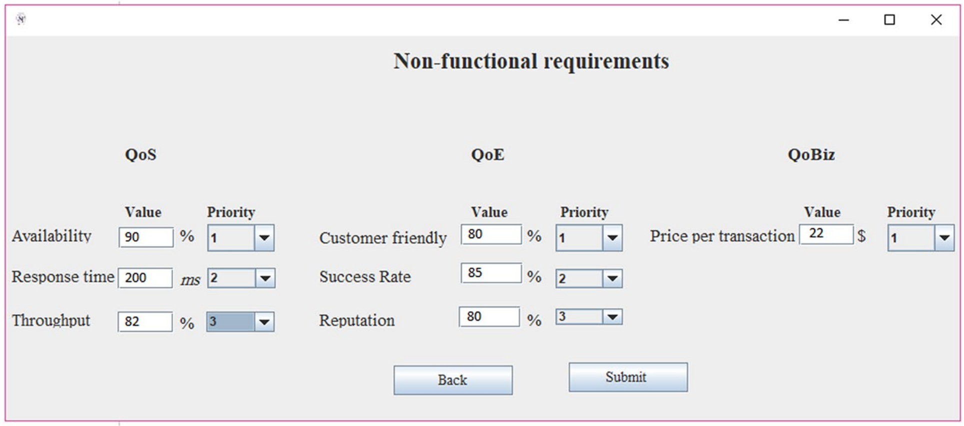

Fig. 8 Req-WSComposer interface illustrating the specification of QoS, QoE, and QoBiz related to the "Activity to city" requirement

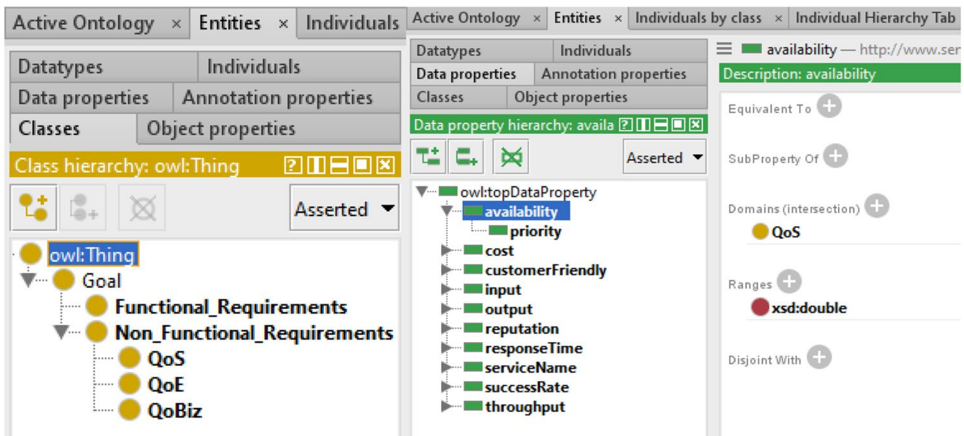

Fig. 9 Requirements ontology specified with Protégé

computes the similarity between two terms by considering their depths in the WordNet taxonomies and returns a score between 0 and 1. In this study, the threshold of empirical similarity is set to 0.8 (Driss et al. 2011b, 2020). Figure 11 presents the Req-WSComposer interface showing the services list obtained after performing the four filters of our semantic discovery algorithm applied for the "Activity to city" requirement.

```
<rdf:RDF>
    <owl:NamedIndividual >
            <functionalRequirements>
          <serviceName> Activity to city service </serviceName>
          <input> Activity </input>
          <output> City </output>
            </functionalRequirements>
        <nonFunctionalRequirements>
          <availability priority="1"> 90 % </availability>
            <responseTime priority="2" > 200 ms
</responseTime>
          <throughput priority="3"> 82 % </throughput>
          <customerFriendly priority="1"> 85%
</customerFriendly>
          <reputation priority="2"> 80% </reputation>
          <successRate priority="3"> 85 % </successRate>
          <cost priority="1"> 22 $ </cost>
        </nonFunctionalRequirements>
    </owl:NamedIndividual>
</rdf:RDF>
```

Fig. 10 Ontological description of the "Activity to city" requirement

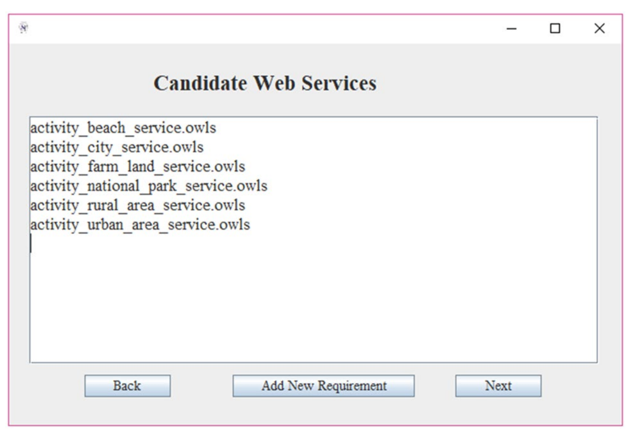

Fig. 11 Req-WSComposer interface illustrating the discovery phase result for the "Activity to city" requirement

Table 2 Discovery phase results obtained for the trip-planning scenario

Requirements	Filter 1 (#services)	Filter 2 (#services)	Filter 3 (#services)	Filter 4 (#services)	Precision (%)	Recall (%)	Response Time (ms)
Activity to city	33	26	21	6	83.33	100	5445
Get weather	11	9	9	8	100	100	3173
Hotel availability	43	40	23	19	100	95	8258
Average					94.44	98.33	5625

In order to validate our results, we begin by checking each returned service manually, assessing whether it can satisfy the developer's functional requirements: (1) at all and (2) in the most accurate and efficient manner possible. Precision and recall measures (Frakes 1992) are used to transpose this work with information retrieval, assessing how many true and relevant services have been returned by the discovery phase. Precision is used to assess the number of true and relevant services identified among those returned, while recall is used to assess the overall number of returned services. This can be formulated according to the following equations:

$$\text{Precision} = \frac{|\{\text{True relevant services}\} \cap \{\text{Returned services}\}|}{|\{\text{Returned services}\}|} \tag{6}$$

$$\text{Recall} = \frac{|\{\text{True relevant services}\} \cap \{\text{Returned services}\}|}{|\{\text{True relevant services}\}|} \tag{7}$$

Table 2 summarizes the results obtained in the discovery phase for the trip-planning scenario. It also validates these results according to precision, recall, and response time as delivered by the Req-WSComposer platform.

As it is shown in Table 2, Req-WSComposer delivers good results in terms of precision (94.44%), recall (98.33%), and response time (5625 ms). Within the same scenario, a previous approach (Driss et al. 2010) had provided 88.89% precision and just 84.62% recall.

After performing the discovery phase, three lists of candidate Web services satisfying the three developer's requirements, respectively, are generated and all possible composition solutions, which are assembled using services from these lists, are identified as it is shown in Fig. 12.

These obtained lists are then passed to the selection phase to identify the best composition solution of Web services by considering the values and the priorities of non-functional requirements specified by the developer. For each discovered Web service, a quality vector is computed. After that, an overall quality score is calculated for each composition solution and the best solution with the highest score is then identified.

The previous steps are also performed for the remaining developer's requirements, which are: "Get weather" and "Hotel availability". For the trip-planning scenario, the

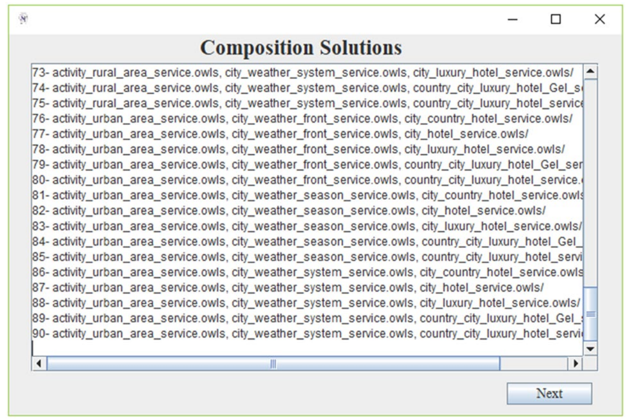

Fig. 12 Req-WSComposer interface illustrating the possible composition solutions for the trip-planning scenario after performing the discovery phase

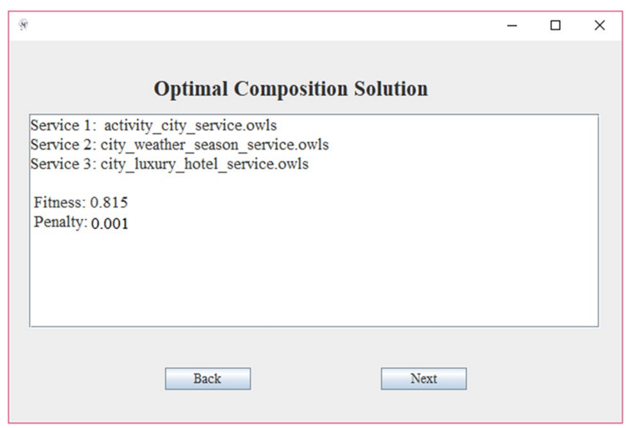

Fig. 13 Optimal composition solution for the trip-planning scenario

obtained services forming the best composition solution are shown in Fig. 13.

Tables 3 and 4 show the obtained matchings between the developer's functional and non-functional requirements and the resulting selected composition solution, respectively.

Table 5 shows the fitness and penalty values of the best composition solution. Obtained fitness and penalty values are 0.8 and 0.001, respectively. The response time to get the optimum Web services from the selection phase

Table 3 Obtained matching between the developer's functional requirements and selected services

Functional requirements			Selected web services			Degree of matching
Operation	Input	Output	Service name	Input	Output	
Activity to city	Activity	City	`activity_city_service`	Activity	City	`Exact`
Weather service	City	Weather	`city_weather_season_service`	City	Weather season	`Subsume`
Hotel availability service	City	Hotel	`city_luxury_hotel_service`	City	Luxury hotel	`Subsume`

Table 4 Obtained matching between developer's non-functional requirements and selected services

	Availability (%)	Response time (ms)	Throughput (%)	Customer friendly (%)	Success rate (%)	Reputation (%)	Cost ($)
Developer specified values	90	200	82	85	85	80	22
`activity_city_service`	92	150	80	88	90	90	25
`city_weather_season_service`	90	130	77	89	88	79	17
`city_luxury_hotel_service`	90	170	91	88	80	85	27

Table 5 Fitness, penalty, and response time values related to the best composition solution satisfying the trip-planning scenario

Fitness	Penalty	Response Time
0.814	0.001	4640 ms

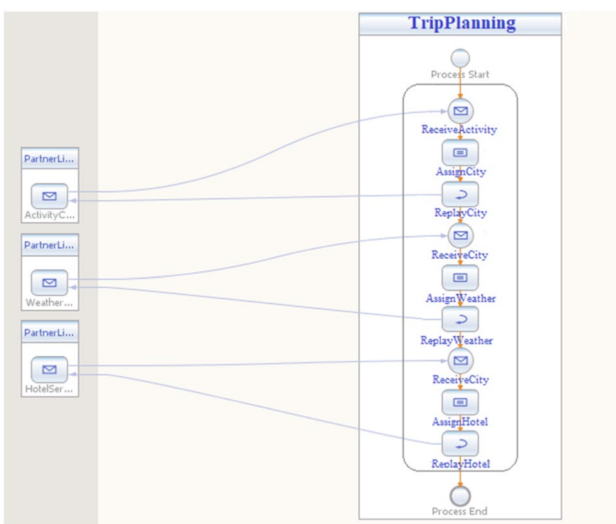

Fig. 14 BPEL process for the trip-planning scenario

is about 5 s. Within the same trip-planning scenario, our previous approach (Driss et al. 2011a) provides a different composition solution with a fitness value of 0.544, a penalty of 0.07, and a response time of 7665 ms. These results can be justified by the fact that the discovery method in (Driss et al. 2011a) is performed on pure syntactic WSDL-based services. Besides, the proposed formal selection algorithm in (Driss et al. 2011a) considers only two QoS parameters, which are the response time and availability.

During the last phase, the services that form an optimal composition solution are organized and orchestrated using the BPEL 2.0 API of Eclipse, as is shown in Fig. 14.

5 Conclusion

In this work, we have introduced and explored a new approach that ensures the discovery, selection, and orchestration/choreography of appropriate Web services matching both functional and different types of non-functional requirements, which are specified by the developer. This approach is implemented using the Req-WSComposer platform to support software developers to build new and value-added service-based applications in a more efficient manner. The proposed approach is tested using an extended and enriched version of the OWLS-TC dataset. The results of our experimentation demonstrate a successful extraction and optimal composition that satisfy developer requirements with high degrees of accuracy and efficiency. For the future, we are looking to improve our Req-WSComposer platform by proposing adaptation strategies to enhance the quality of the composition solutions with a high penalty value. Furthermore, the current work can be extended using larger, more complex cloud-based datasets of microservices in Internet-of-Things enviroments (Ben Atitallah et al. 2020; Hajjaji et al. 2021).

References

Aljazzaf ZM (2015) TQoSM: total quality of service model. In: 2015 International Conference on Industrial Engineering and Operations Management (IEOM). IEEE, pp. 1–8

Alves A, Arkin A, Askary S, Barreto C, Bloch B, Curbera F, Ford M, Goland Y, Guízar A, Kartha N, Liu CK, Khalaf R, König D, Marin M, Mehta V, Thatte S, Van der Rijn D, Yendluri P, Yiu A (2007) OASIS web services business process execution language (WSBPEL) TC. https://www.oasis-open.org/committees/wsbpel/. Accessed 12 January 2021

Azmeh Z, Driss M, Hamoui F, Huchard M, Moha N, Tibermacine C (2011) Selection of composable web services driven by user requirements. In: 2011 IEEE International Conference on Web Services. IEEE, pp 395–402

Aznag M, Quafafou M, Durand N, Jarir Z (2013) Web services discovery and recommendation based on information extraction and symbolic reputation. Int J Web Serv Comput (IJWSC) 4(1):1–18

Bagga P, Joshi A, Hans R (2019) QoS based web service selection and multi-criteria decision making methods. Int J Interact Multim Artif Intell 5(4):113–121

Bekkouche A, Benslimane SM, Huchard M, Tibermacine C, Hadjila F, Merzoug M (2017) QoS-aware optimal and automated semantic web service composition with user's constraints. Serv Orient Comput Appl 11(2):183–201

Ben Atitallah S, Driss M, Boulila W, Ghézala HB (2020) Leveraging deep learning and IoT big data analytics to support the smart cities development: review and future directions. Comput Sci Rev 38:100303

Bocchi E, De Cicco L, Rossi D (2016) Measuring the quality of experience of web users. ACM SIGCOMM Comput Commun Rev 46(4):8–13

Cheng B, Zhao S, Li C, Chen J (2016) A web services discovery approach based on mining underlying interface semantics. IEEE Trans Knowl Data Eng 29(5):950–962

Curbera F, Duftler M, Khalaf R, Nagy W, Mukhi N, Weerawarana S (2002) Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Comput 6(2):86–93

D'Mello DA, Ananthanarayana VS (2009) Semantic web service selection based on service provider's business offerings. IJSSST 10(2):25–37

De Castro V, Marcos E, Wieringa R (2009) Towards a service-oriented MDA-based approach to the alignment of business processes with IT systems: from the business model to a web service composition model. Int J Cooperat Inform Syst 18(02):225–260

De Castro V, Musicante MA, Da Costa US, de Souza Neto PA, Vargas-Solar G (2014) Supporting non-functional requirements in services software development process: an mdd approach. International Conference on Current Trends in Theory and Practice of Informatics. Springer, Cham, pp 199–210

Decker G, Kopp O, Leymann F, Weske M (2007) BPEL4Chor: Extending BPEL for modeling choreographies. In IEEE international conference on web services (ICWS 2007). IEEE, pp 296–303

Driss M, Moha N, Jamoussi Y, Jézéquel JM, Ben Ghézala HH (2010) A requirement-centric approach to web service modeling, discovery, and selection. International conference on service-oriented computing. Springer, Berlin, Heidelberg, pp 258–272

Driss M, Jamoussi Y, Moha N, Jézéquel JM, Ben Ghézala HH (2011) Une approche centrée exigences pour la composition de services web. Ingénierie des Systèmes d'Information 16(2):97–125

Driss M, Jamoussi Y, Jézéquel JM, Ben Ghézala HH (2011a) A multi-perspective approach for web service composition. In: Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services. ACM, pp 106–111

Driss M, Aljehani A, Boulila W, Ghandorh H, Al-Sarem M (2020) Servicing your requirements: An FCA and RCA-driven approach for semantic web services composition. IEEE Access 8:59326–59339

Fadhlallah B, Le Sommer N, Mahéo Y (2017) Choreography-based vs orchestration-based service composition in opportunistic networks. In: 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). IEEE, pp 1–8

Frakes WB (1992) Information retrieval: Data structures and algorithms. Pearson Education India

Garriga M, Flores A, Cechich A, Zunino A (2015) Web services composition mechanisms: a review. IETE Tech Rev 32(5):376–383

Hajjaji Y, Boulila W, Farah IR, Romdhani I, Hussain A (2021) Big data and IoT-based applications in smart environments: a systematic review. Comput Sci Rev 39:100318

Hammal Y, Mansour KS, Abdelli A, Mokdad L (2020) Formal techniques for consistency checking of orchestrations of semantic web services. J Comput Sci 44:101165

Hu C, Wu X, Li B (2020) A framework for trustworthy web service composition and optimization. IEEE Access 8:73508–73522

Kavantzas N, Burdett D, Ritzinger G, Fletcher T, Lafon Y, Barreto C (2005) Web services choreography description language version 1.0. https://www.w3.org/TR/ws-cdl-10/. Accessed 12 January 2021

Khanouche ME, Attal F, Amirat Y, Chibani A, Kerkar M (2019) Clustering-based and QoS-aware services composition algorithm for ambient intelligence. Inf Sci 482:419–439

Khanouche ME, Gadouche H, Farah Z, Tari A (2020) Flexible QoS-aware services composition for service computing environments. Comput Netw 166:106982

Kritikos K, Plexousakis D (2009) Requirements for QoS-based web service description and discovery. IEEE Trans Serv Comput 2(4):320–337

Lécué F (2009) Optimizing QoS-aware semantic web service composition. International semantic web conference. Springer, Berlin, Heidelberg, pp 375–391

Martin D, Burstein M, Hobbs J, Lassila O, McDermott D, McIlraith S, Narayanan S, Paolucci M, Parsia B, Payne T, Sirin E, Srinivasan N, Sycara K (2004) OWL-S: semantic markup for web services. https://www.w3.org/Submission/OWL-S/. Accessed 12 January 2021

Metzger A, Benbernou S, Carro M, Driss M, Kecskemeti G, Kazhamiakin R, Krytikos K, Mocci A, Di Nitto E, Wetzstein B, Silvestri F (2010) Analytical quality assurance. Service research challenges and solutions for the future internet. Springer, Berlin, Heidelberg, pp 209–270

Paolucci M, Kawamura T, Payne TR, Sycara K (2002) Semantic matching of web services capabilities. International semantic web conference. Springer, Berlin, Heidelberg, pp 333–347

Papazoglou M (2012) Web services: principles and technology, 2nd edn. Pearson Education, Essex

Papazoglou MP, Van Den Heuvel WJ (2007) Service-oriented architectures: approaches, technologies, and research issues. VLDB J 16(3):389–415

Papazoglou M, Pohl K, Parkin M, Metzger A (Eds) (2010) Service research challenges and solutions for the future internet: S-cube-towards engineering, managing and adapting service-based systems (vol. 6500). Springer

Rai GN, Gangadharan GR, Padmanabhan V (2015) Algebraic modeling and verification of Web service composition. Procedia Computer science 52:675–679

Rodríguez G, Mateos C, Misra S (2020) Exploring web service QoS estimation for web service composition. International Conference on Information and Software Technologies. Springer, Cham, pp 171–184

Rodriguez-Mier P, Pedrinaci C, Lama M, Mucientes M (2015) An integrated semantic web service discovery and composition framework. IEEE Trans Serv Comput 9(4):537–550
Sangaiah AK, Bian GB, Bozorgi SM, Suraki MY, Hosseinabadi AA, Shareh MB (2019) A novel quality-of-service-aware web services composition using biogeography-based optimization algorithm. Soft Computing, pp 1–13
Sheng QZ, Qiao X, Vasilakos AV, Szabo C, Bourne S, Xu X (2014) Web services composition: a decade's overview. Inf Sci 280:218–238
Shijie Z, Xu P, Xu Y (2020) Web service composition verification based on symbol model checking and Petri nets. In: Developments of Artificial Intelligence Technologies in Computation and Robotics, Proceedings of the 14th International Flins Conference (Flins 2020). World Scientific, vol 12, p 309
Siavashi F, Truscan D, Vain J (2016) On mutating UPPAAL timed automata to assess robustness of web services. In ICSOFT-EA, pp 15–26
Suchithra M, Ramakrishnan M (2015) Efficient discovery and ranking of web services using non-functional QoS requirements for smart grid applications. Procedia Technol 21:82–87
Van Moorsel A (2001) Metrics for the internet age: quality of experience and quality of business. In: Fifth International Workshop on Performability Modeling of Computer and Communication Systems, Arbeitsberichte des Instituts für Informatik, Universität Erlangen-Nürnberg, Germany, vol 34, No 13, pp 26–31
Wu Z, Palmer M (1994) Verbs semantics and lexical selection. In: Proceedings of the 32nd annual meeting on Association for Computational Linguistics. Association for Computational Linguistics, pp. 133–138
Yu Q, Bouguettaya A (2009) Foundations for efficient web service selection. Springer Science & Business Media
Zeng L, Benatallah B, Ngu AH, Dumas M, Kalagnanam J, Chang H (2004) QoS-aware middleware for web services composition. IEEE Trans Softw Eng 30(5):311–327
Zhou F, Ma C, Qu J, Song X, Zhang C (2020) A service composition optimization model based on petri nets and service contracts. In: 2020 IEEE 8th International Conference on Information, Communication and Networks (ICICN). IEEE, pp 177–181
Zhu Y, Huang Z, Zhou H (2017) Modeling and verification of web services composition based on model transformation. Softw Pract Exp 47(5):709–730
Zolotas C, Diamantopoulos T, Chatzidimitriou KC, Symeonidis AL (2017) From requirements to source code: a model-driven engineering approach for RESTful web services. Automat Softw Eng 24(4):791–838

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.