QoR-driven Resource Selection for Hybrid Web Environments

Lara Kallab[1], Richard Chbeir[2], and Michael Mrissa[3]

[1] Groupe Open, Levallois Perret, 92300, France
lara.kallab@open-groupe.com
[2] Univ Pau & Pays Adour, E2S UPPA, LIUPPA, EA3000, Anglet, 64600, France
richard.chbeir@univ-pau.fr
[3] InnoRenew CoE, Livade 6, 6310 Izola, Slovenia
michael.mrissa@innorenew.eu

Abstract. In the Web of Things (WoT) context, an increasing number of objects provide functions as RESTful services (resources), that can be composed with other existing resources, to create value-added processes (compositions). However, to form a composition, selecting the suitable resources is becoming more challenging, due to: (1) the growing number of resources providing identical functions, which calls for the use of Quality of Resource (QoR) to distinguish between them, and (2) the transient nature of resource availability as a result of objects' sporadic connectivity in the WoT environments. In this chapter, we present a QoR-driven resource selection approach that forms i-compositions (with i $\in \mathbb{N}^*$) offering different implementation alternatives. This is done using a selection strategy adaptor that considers QoR constraints and Inputs/Outputs matching of related resources, as well as resource availability and users' different needs (e.g., optimal compositions having the highest scores, and optimistic compositions having acceptable scores but obtained in more satisfactory delays). Analysis are made to evaluate our resource quality model against existing ones, and experiments are conducted in different environments setups to study the performance of our work.

Keywords: Hybrid Web environments · Resource selection · i-compositions.

1 Introduction

Nowadays, a plethora of Web environments (Web applications, Web platforms, etc.), publish their functions as RESTful services, i.e., self-contained and self-describing resources that follow the REpresentational State Transfer (REST) architectural style [7]. As the Web has become a major medium of communication, integrating objects (e.g., smart devices) into the Web and taking advantage of its open popular standards (e.g., HTTP), has created an emerging trend: the Web of Things (WoT) [1]. In the WoT, objects are abstracted as RESTful services, which are resources individually identifiable with a Uniform Resource Identifier (URI) and provide functions callable using HTTP methods (e.g., GET and POST). A resource is either (i) dynamic, connected to and removed from the Web environment at different instances, or (ii) static, always connected to the environment. In many cases, a single resource is not sufficient to satisfy specific user requests, and often, resources are combined forming a composition that achieves the desired results. To form a composition, selecting the appropriate resources is essential. However, several challenges arise:

1. **Selecting the appropriate resource for a function**: With the growing number of resources realizing the same function, selecting the appropriate one while considering user constraints (if given), is non-trivial for end-users. Therefore, taking into account the Quality of Resource (QoR) attributes used to differentiate resources having identical functions [15], is important to select the suitable resource for a function. The increasing number of candidate resources and their various QoR attributes [8] (e.g., Availability and Cost) require an automatic approach that facilitates the task for end-users, and accelerates the selection process. Also, during selection, considering the matching of the input and output (I/O) parameters of the related resources is essential to generate compositions that fit efficiently users needs.
2. **Forming different composition alternatives**: The selected dynamic resource(s) for a composition may be unavailable for execution. To avoid repeating the selection process to form new suitable composition, providing i-compositions (i $\in \mathbb{N}^*$), i.e., a set of compositions having different implementation alternatives, becomes important. These compositions respond to the user request by using, each, a different set of resources. This allows to substitute a composition that misses a resource, by another one consisting of available resources. Furthermore, in some cases, users require compositions having the highest possible scores, others may need compositions having acceptable scores obtained in more satisfactory delays, etc. Thus, forming compositions that are adaptive to different user needs is essential.

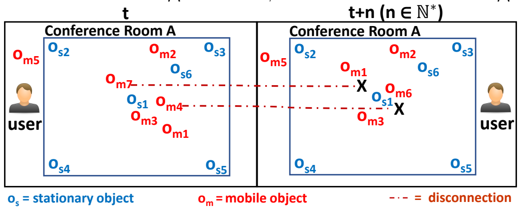
In the literature, some works [2,14,16] were based on Quality of Services (QoS) to select the most suitable ones according to user constraints or preferences, without taking into account I/O service matching and service dynamicity. Others [10,13] aimed at finding a sequence of services starting from given inputs and leading to the desired outputs, without considering service matching on the functional level, their QoS, and dynamicity. Also, none of the existing service composition approaches [3, 6], forms several types of compositions realizing different user needs (optimal compositions having the highest scores, optimistic compositions having acceptable scores but obtained in more satisfactory delays, etc.).

To address the aforementioned challenges and existing limitations, we present, in this chapter, a QoR-driven resource selection that forms i-compositions (with i $\in \mathbb{N}^*$) offering different implementation alternatives for user request. To do so, we define a Selection Strategy Adaptor (SSA) that allows selecting the suitable resources, while considering user QoR constraints and I/O matching of related resources, as well as resource dynamicity and user requested composition type (e.g., optimal compositions having the highest scores, and optimistic compositions having acceptable scores but obtained in more satisfactory delays). Resource selection is automatic, i.e., based on semantic annotations integrated within resource descriptions expressed using Hydra [9] in our work.

The rest of the chapter is organized as follows. Section 2 motivates our work, and describes the main challenges and needs. Section 3 discusses related work and shows the originality of our solution. Section 4 details our resource selection solution. Section 5 compares our defined QoR model against existing works, and evaluates the solution performance. Finally, Section 6 concludes the chapter.

2 Motivation, Challenges and Needs

We motivate our work through OpenCEMS[4]: a Web platform that provides solutions for energy data management in connected environments. The platform allows to connect (1) stationary and mobile objects that provide static/dynamic resources, and (2) Web applications exposed as static resources. The resources, described in Hydra, are used for: collecting on-site data, preprocessing collected data, and analyzing data. Many requests occur in OpenCEMS. We consider a building manager that wants to predict the temperature in the conference room A. To satisfy his request, specified by the "ATP" (Air Temperature Prediction) function, it is important to select the appropriate resources realizing his need[5]. However, several challenges arise, as illustrated in Figure 1:



Function	Object
Air Temperature Collection (ATC)	o_{m3}, o_{m4}, o_{m5}, o_{m6}, o_{m7}, o_{s1}, o_{s2}, o_{s5}
CO_2 Collection (COC)	o_{m1}, o_{m2}, o_{s3}, o_{s4}, o_{s6}

Fig. 1: Challenges illustration in the OpenCEMS platform

1. **Select the appropriate resources to form a suitable composition**. When several resources are identified having the same function required to answer "ATP", as "Air Temperature Collection", selecting the appropriate ones among others is not obvious for end-users, as the building manager. Therefore, QoR plays an essential role to select the suitable resources. For instance, object o_{s1} can be better than others (e.g., o_{m3} and o_{m4}) as it may have: (i) full battery capacity denoting a full availability, (ii) continuous connectivity to the environment (since it is static), (iii) cost free when using it, and (iv) high usage rate (it has been invoked many times). Considering these QoR, allows selecting the appropriate resources among other candidates. As many candidate resources can connect to openCEMS with various QoR, resource selection requires an automatic approach to facilitate and accelerate the task for the building manager. Such approach should also consider I/O matching of the linked resources to guarantee efficient composition results. Moreover, in some cases, the building manager may require:
   (a) **Prediction results using the most qualified resources.** In this case, the resources to be selected are the ones having the highest values of quality aspects, independently of the selection response time, as the building manager may need to adjust the necessary temperature, in conference room A, for a meeting that will start in the late afternoon.

[4] Connected Environment & Distributed Energy Data Management Solutions: `https://opencems.sigappfr.org/`
[5] We assume the resource discovery is already performed based on the required location

   (b) **Fast but good prediction results.** As the building manager might be feeling very hot in the conference room A, he needs fast prediction results to regulate the temperature. This is done by selecting the first resources realizing his demand without the need to check others. However, and despite requiring fast results, it is important that the selected resources have minimal quality aspects to guarantee good composition results.
   (c) **Always available results.** The building manager may need to have results at anytime of his request, i.e., even if dynamic resources are disconnected from the Web platform there are always other resources that can take over. Thus, the resources that will be selected are always connected to the environment (static resources) at both instants t and t+n.

   In other particular cases, the building manager may define other needs as: (i) **Trusted results**, generated by only static resources already provided by the OpenCEMS Web platform (e.g., o_{s1}, o_{s2}, and o_{s5}), (ii) **Cost free results**, using resources without any charge, and **Reliable results**, using resources that can be linked in the most proper way (i.e., best I/O matching between the related resources), etc. Thus, it is necessary to consider user needs and constraints, and adapt resource selection accordingly.
2. **Form several composition alternatives**. The selected dynamic resource(s) for a composition may be unavailable during execution. As such, at instant t, 5 mobile objects providing "ATC" are positioned in the conference room A. If o_{m4} provides the appropriate resource among these objects, it will be selected to take part in the composition. However, at t+n, o_{m4} is disconnected, and thus, the composition will miss a resource if the composition execution time is $\geqslant$ t+n. To avoid repeating resource selection process to form a new suitable solution with available resources, it is important to identify i-compositions during resource selection, with i $\in \mathbb{N}^*$.

To address these challenges and respond to user needs, we propose a QoR-driven resource selection adapted to: (i) different requested composition types (e.g., Optimal, Optimistic, and Optimistic cost-free), and (ii) QoR constraints. Our solution considers I/O matching between the related resources and resource dynamicity (when it is necessary), to form the required i-compositions.

3 Related Work

3.1 QoS-based Approaches

In [2], a quality-driven solution for resource selection is presented. The approach uses a set of quality attributes incorporated into each resource description expressed with Hydra, and implements a skyline-based algorithm that reduces the set of candidates for a given task. In [4], a heuristic is proposed to solve QoS-aware Web service composition problem. It uses a backtracking algorithm on the results computed by a Linear Program relaxation, while considering user constraints given to the overall composition. In [14], an approach for a service selection based on both qualitative and quantitative user QoS preference with services trust properties is presented. The solution is applicable in Big Data Web environments consisting of massive migrated services, i.e., business applications,

to the cloud. The work in [5] proposes a QoS-aware service composition based on QoS correlations. It produces the optimal composite service, and considers service plans with sequence structure.
Discussion: Although these works consider QoS attributes and user constraints/ preferences, they neglect services dynamicity and I/O matching between the linked services. Moreover, they are not adapted to generate different composition types to realize different user needs.

3.2 I/O-based Approaches

Work [13], presents a graph-based framework for automatic service composition, by focusing on the semantic I/O services matching. It produces a service composition containing the minimum number of services, and multiple compositions can be extracted satisfying user request in terms of I/O. In [10], a formal model is provided for an AI planning-oriented service composition. It precomputes the I/O semantic similarity between services, according to causal links, which are logical dependencies among input and output parameters of different services.
Discussion: Despite computing I/O services semantics, these works do not consider the functional aspect of the related services, nor even the dynamicity of services. Also, QoS attributes and user constraints/preferences are neglected.

3.3 k-compositions Approaches

A top-k automatic service composition solution is presented in [6]. It adopts the idea of MapReduce, by mapping the top-k service compositions into multiple tasks that can be executed in parallel. The solution considers one quality of service, i.e., Response time, and I/O services similarities to filter the services. In [3], an approach for composing the top-k DaaS (Data as a Service) services is proposed. The top-k compositions are computed based on a fuzzy score, associated for each service and service composition, and fuzzy user preferences expressed in fuzzy terms (e.g., “cheap” for services price).
Discussion: Although these works produce several service compositions, and consider QoS attributes, they are not designed to handle service dynamicity, nor the generation of different composition types answering user needs. Also, only in [6], I/O matching and QoS attributes are used in the same approach.

4 QoR-driven Resource Selection for i-compositions

4.1 General Overview

Figure 2 shows the process overview of our resource selection, applicable in hybrid Web environments providing static and/or dynamic resources. The process is used to form i-compositions responding to user request, r, and adapted to user request type. The latter includes one of the following desired compositions types: (i) optimal, denoting compositions having the highest scores, (ii) optimistic, referring to compositions having acceptable scores, i.e., $\geqslant$ a specific computed threshold (see Section 4.4), or (iii) hybrid, denoting compositions having acceptable scores but whose dynamic aspect is considered, guaranteeing the existence of a composition at any instance. A composition type can be followed optionally by other subtypes (e.g., trusted, denoting that only static resources can be part of the compositions). As for the user request, r, it is defined formally as:

Definition 1 - $\boldsymbol{r} = (\boldsymbol{f}, \boldsymbol{P}, \boldsymbol{C})$, *where:*

- $\boldsymbol{f}$, *is the user requested function, selected from a list of functions, F, provided by the resources connected to the Web environment at the current instant.*
- $\boldsymbol{P}$, *is the set of parameters necessary to execute f, such as the required location (e.g., conference room A). P is mainly used by the resource discovery process, which is out of scope of this chapter, to identify the needed resources.*
- $\boldsymbol{C}$, *is the given user constraints according to which, i-compositions are obtained, and such that* $\boldsymbol{C} = \boldsymbol{Q_c} \cup \boldsymbol{i} \cup \boldsymbol{W} \cup \boldsymbol{d}$, *with:*
  - $\boldsymbol{Q_c} = \boldsymbol{Q_c^{res}} \cup \boldsymbol{Q_c^{f}}$, *refers to the set of constraints given to the resources* ($\boldsymbol{Q_c^{res}}$) *and to their provided functions* ($\boldsymbol{Q_c^{f}}$), *with* $\boldsymbol{Q_c^{res}} = \bigcup_{\boldsymbol{i=1}}^{\boldsymbol{n}} \left\{\boldsymbol{q_i^{res}}\right\}$, *and* $\boldsymbol{Q_c^{f}} = \bigcup_{\boldsymbol{j=1}}^{\boldsymbol{m}} \left\{\boldsymbol{q_j^{f}}\right\}$, *and where:*
    - *n is the number of attributes describing a resource, and m the number of attributes describing its provided functions (see Section 4.2).*
    - $\boldsymbol{q_i^{res}}|\boldsymbol{q_j^{f}} = [\boldsymbol{min_{i|j}}\text{-}\boldsymbol{max_{i|j}}]$, *with* $\boldsymbol{min_{i|j}}$, $\boldsymbol{max_{i|j}}$ *are respectively the minimum and maximum values given for* $\boldsymbol{q_i^{res}}$ *and* $\boldsymbol{q_j^{f}}$.
  - $\boldsymbol{i} \in \mathbb{N}^*$, *is the desired number of compositions. By default i=1, and can be only specified for optimal and optimistic compositions. For the hybrid compositions, the number of solutions depends from resource dynamicity aspect of the formed compositions (see Section 4.4).*
  - $\boldsymbol{W} = \{\boldsymbol{w_{qor}}, \boldsymbol{w_{io}}\}$, *are the weights given respectively to the score of the resources and their I/O matching, during compositions score calculation (see Section 4.3).* $\boldsymbol{w_{qor}}$, $\boldsymbol{w_{io}} \in \boldsymbol{R}^+$ *and are bounded by [0, 1].*
  - $\boldsymbol{d}$, *is the degree value rate (in %) of the threshold, T (see Section 4.4), that refers to the minimal acceptable score of the i-compositions.*

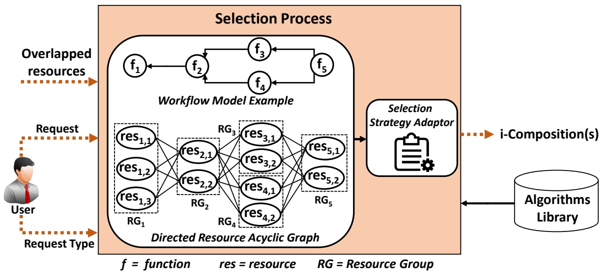


Fig. 2: Overview of the resource selection process

The selection process, illustrated in Figure 2, is executed when there are several candidate resources for at least one required function necessary to realize user request, r. The required functions for r form a Workflow Model, WM. During the process, the resources realizing an identical function are grouped into the same resource group, RG. Each resource of a RG can be linked to the resources included in the RG related to the next function (as defined in WM), forming a Directed Resource Acyclic Graph, DRAG. In order to satisfy user different requested composition types, we define a Selection Strategy Adaptor (SSA) that adapts to user needs to form the required i-set of resource compositions. Using a graph algorithm, as DFS (Depth First Search), from the algorithms library, SSA can traverse DRAG to compute the I/O matching of the linked eligible resources (whose QoR values respect user constraints defined in Q_c), and produces optimal compositions having the highest scores. SSA allows also to form compositions having acceptable scores, i.e., $\geqslant$ a computed threshold, T (see Section 4.4, obtained in more satisfactory delays. Such threshold is computed for the compositions types: optimistic and hybrid.

4.2 Preliminaries

In our work, a resource, $\boldsymbol{res}$, can be either static, $\boldsymbol{res^s}$, or dynamic, $\boldsymbol{res^d}$. It provides a set of functions: $\mathbf{F} = \bigcup_{i=1}^{\mathbb{N}^*} \{\mathrm{f_i}\}$, with: $\boldsymbol{f_i} = (\boldsymbol{n, I, O, m, Q_f})$ denotes a single function, and where: **n** refers to $\boldsymbol{f_i}$ name, **I** denotes the input(s) of $\boldsymbol{f_i}$, **O** denotes the output(s) of $\boldsymbol{f_i}$, **m** is the HTTP verb used to call $\boldsymbol{f_i}$, and $\boldsymbol{Q_f}$ is the set of quality attributes related to each function ($\boldsymbol{f_i}$). $\boldsymbol{Q_f} = \bigcup_{i=1}^{\mathbb{N}^*} \{(\boldsymbol{qf_i : vf_i})\}$, with $\boldsymbol{qf_i}$ is the name of the quality attribute, and $\boldsymbol{vf_i} \in \mathbb{R}^+$. Each resource has also directly related quality attributes: $\boldsymbol{Q_{res}} = \bigcup_{i=1}^{\mathbb{N}^*} \{(\boldsymbol{qres_i : vres_i})\}$, with $\boldsymbol{qres_i}$ is the name of the attribute, and $\boldsymbol{vres_i} \in \mathbb{R}^+$.

In the literature, we find various QoR used to differentiate resources with similar functions [15]. In this chapter, we consider 4 QoR, where some are related to the resource itself, and others related to each of its provided functions:

- **Dynamicity**, is the quality aspect of whether the resource is always available ($\boldsymbol{res^s}$) or not ($\boldsymbol{res^d}$). It is equal to 0 for $\boldsymbol{res^s}$, and to 1 for $\boldsymbol{res^d}$.
- **Availability**, is the degree (%) to which $\boldsymbol{res}$ is operational/ready for immediate use. For resources provided by objects, it denotes their battery capacity.
- **Cost**, is the amount of money to pay to use a function of $\boldsymbol{res}$.
- **Usage**, is a value that increments when a resource function is used. To avoid the re-initialization of the usage when a $\boldsymbol{res^d}$ is disconnected, we define for each function of a $\boldsymbol{res^d}$, a Time To Live value denoting the maximum time during which a $\boldsymbol{res^d}$ can be disconnected before the usage decrements by 1.

Based on $\boldsymbol{res}$ definition, we extended Hydra-based resource description[6], to include the QoR values. QoR are grouped as: (i) maximization attributes, whose values should be maximized (e.g., Availability), and (ii) minimization attributes, whose values should be minimized (e.g., Cost). They are used to compute, for each provided resource function, a global score that is defined as: score($\boldsymbol{res_f}$) $= \sum_{i=1}^{\mathbb{N}^*} \{\boldsymbol{vres_i}\} + \sum_{i=1}^{\mathbb{N}^*} \{\boldsymbol{vf_i}\}$, where $\boldsymbol{vres_i}$ (excepting the "Dynamicity" attribute value) and $\boldsymbol{vf_i}$ are normalized based on the work in [11].

4.3 QoR-based Resource Graph: Formal Modeling

The functions required for **f** in **r**, define with their dependencies a Workflow Model, WM. Based on the functions order in WM, the overlapped resources are linked, forming a Directed Resource Acyclic Graph, DRAG, that is defined as:
Definition 2 - $\boldsymbol{DRAG = (RES, Rel, f_{RES}, f_{Rel})}$:

[6] Hydra description example is available online: https://tinyurl.com/tose56k

- ***RES**, is the set of the identified and overlapped static/dynamic resources.*
- ***Rel**, is the set of relations linking the resources together.*
- *$\boldsymbol{f_{RES}}$, is the function computing the score of each resource function.*
- *$\boldsymbol{f_{Rel}}$, is the function linking the resources together, and computing their link score based on their I/O similarities.*

The resources providing the same function, form a resource group, $\boldsymbol{RG_f}$, relative to that function, where: $\boldsymbol{RG_f} = \bigcup_{i=1}^{m} \left\{\boldsymbol{res_{(f,i)}}\right\}$, with m is the number of candidate resources realizing function f, and $\boldsymbol{res_{(f,i)}}$ refers to the resource $\boldsymbol{res_i}$ providing f. A resource composition, RC, consists of a set of resources included, each, in a different $\boldsymbol{RG_f}$, where: $\mathbf{RC} = \bigcup_{f=1}^{n} \left\{\boldsymbol{res_{(f,i)}}\right\}$, such that n is the number of functions in WM, and $\boldsymbol{i \in m}$, with m denotes the number of resources in the correspondent $\boldsymbol{RG_f}$. During selection, I/O matching score between every linked eligible resources is computed, using any similarity measure function between keywords (as Jaccard measure [12]), as follows: sim($\boldsymbol{res_{(f,i)}}$, $\boldsymbol{res_{(f',j)}}$) $= \sum_{u=1}^{U} \sum_{v=1}^{V} \boldsymbol{sim(out_u^{res(f,i)}, in_v^{res(f',j)})}$, with:

- $\boldsymbol{res_{f,i}}$, $\boldsymbol{res_{f',j}}$, denote resources that belong, respectively, to $\boldsymbol{RG_f}$ and $\boldsymbol{RG_{f'}}$, where f precedes f' in WM.
- $\boldsymbol{out_u}$, is an output of $\boldsymbol{res_{f,i}}$, and U is the number of $\boldsymbol{res_{f,i}}$ outputs.
- $\boldsymbol{in_v}$, is an input of $\boldsymbol{res_{f',j}}$, and V is the number of $\boldsymbol{res_{f',j}}$ inputs.

Each RC in DRAG has a score, score(RC), with: **score(RC) = Score(RES) + Score(Rel)**, and where:

- **Score(RES)** $= \sum_{f=1}^{n} \boldsymbol{score(res_{(f,i)})}$, is the scores sum of the involved resources realizing the required WM, with n is the number of WM functions.
- **Score(Rel)** $= \sum \boldsymbol{sim(res_{(f,i)},\ res_{(f',j)})}$, is the sum of I/O similarity scores of each 2 eligible linked resources in RC, where: f precedes f' in WM, and sim($\boldsymbol{res_{(f,i)}}$, $\boldsymbol{res_{(f',j)}}$) $\in [0, 1]$.

Score(RES) and Score(Rel) can be multiplied, each, by a weight value defined in **r**, allowing users to assign them a priority for compositions score calculation.

4.4 Selection Strategy Adaptor for i-compositions

In order to form i-compositions satisfying different user needs, we define the Selection Strategy Adaptor (SSA) that allows the generation of 3 main compositions types: (1) optimal, (2) optimistic, and (3) hybrid) (see Section 4.1). Each of these composition types can be followed optionally by several composition subtypes, e.g., (i) trusted, refers to compositions having only static resources that are already provided by the Web environment, and (ii) efficient, denotes compositions that include resources with minimal acceptable normalized usage value. As shown in Table 1, the composition subtypes are defined according to either: (i) a minimal QoR attribute value (e.g., Dynamicity for trusted compositions), or (ii) a set of minimal QoR attributes values (Availability, Cost, and Usage for qualified compositions), or (iii) a minimal Score(Rel) (as for reliable compositions), computed based on “l”, which is the number of dependencies links between the required functions in WM. However, in addition to these constraints highlighted in red, both optimistic and hybrid composition types should respect other QoR attributes and Score(Rel) values, to ensure having compositions with an acceptable score(RC), and thus, good compositions results. If optimal compositions subtypes are needed (e.g., optimal cost-free), a filtering process is executed before score(RC) calculations. Such filtering is done based on the maximum values of the constraints for which the values are highlighted in red in Table 1, among the resources in DRAG. For example, Optimal Efficient compositions will include resources having Max(Usage), which refers to the maximum value of the Usage attribute among the resources in DRAG.

Table 1: QoR values and Score(Rel) in optimistic and hybrid compositions subtypes

	Dynamicity	**Availability**	**Cost**	**Usage**	**Score(Rel)**
Trusted	0	$\geqslant 0.5$	$\leqslant 0.25$	$\geqslant 0.5$	$\geqslant (l \times 0.5)$
Cost-free	0 | 1	$\geqslant 0.5$	0	$\geqslant 0.5$	$\geqslant (l \times 0.5)$
Efficient	0 | 1	$\geqslant 0.5$	$\leqslant 0.25$	$\geqslant 0.75$	$\geqslant (l \times 0.5)$
Effective	0 | 1	$\geqslant 0.75$	$\leqslant 0.25$	$\geqslant 0.5$	$\geqslant (l \times 0.5)$
Qualified	0 | 1	$\geqslant 0.75$	$\leqslant 0.25$	$\geqslant 0.75$	$\geqslant (l \times 0.5)$
Reliable	0 | 1	$\geqslant 0.5$	$\leqslant 0.25$	$\geqslant 0.5$	$\geqslant (l \times 0.75)$

Based on the given composition type, SSA forms the necessary i-compositions. The i value ($\in \mathbb{N}^*$) can be determined by the user in r, for the 2 main types: optimal and optimistic, where SSA retrieves the i-compositions having the best or acceptable scores respectively. As for hybrid compositions, SSA stops generating the solutions having acceptable scores until having one containing only static resources, guaranteeing the existence of a composition at any instance. If the user defines in r, constraints that do not align with the designated composition subtype constraints, the latter are considered.

When optimistic or hybrid compositions are required, SSA applies several steps:

1. **Compute the minimum score of an acceptable composition**. A composition is considered acceptable, if its score(RC) is $\geqslant$ a specific Threshold, T. When optimistic or hybrid composition types are requested without subtypes, T is defined as: T = $\boldsymbol{[(n \times Avg(Q_c)) + (l \times 0.5)] \times (d/100)}$, where:
   - **n** is the total number of functions in WM.
   - $\mathbf{Avg(}\boldsymbol{Q_c}\mathbf{)}$ are the average of the normalized QoR constraints defined in r (excepting the "Dynamicity" attribute). If $\boldsymbol{Q_c}$ are not given, the average of each QoR is calculated using their maximum values among DRAG.
   - **l**, is the number of the dependencies links between WM functions. We consider that there is, at least, an I/O similarity match (=0.5) between any two linked resources.
   - **d**, is the composition acceptance degree value **(%)** given in user request.

   If subtype compositions are requested, T is computed as: $\boldsymbol{T_{subtype}} = \boldsymbol{[(n \times Q) + (l \times s)] \times (d/100)}$, with Q denotes the minimum values of the attributes as in Table 1 (excepting the "Dynamicity"), and s $\in$ [0,1] is the minimum I/O similarity matching score between any two linked resources. s = 0.75 whenever subtype = reliable, and s = 0.5 for the rest of subtype values.
2. **Compute the score of a composition formed by eligible resources**. To do so, a generator is used to get the possible set of RC without score calculation. During RC generation, the following conditions are performed:
   (i) If a resource in RC is not eligible, it is registered in array, **arr_notEl**, and another possible RC is generated
   (ii) If all the resources of RC are eligible, score(RC) is computed. If score(RC) $\geqslant$ **T**, RC is saved into the suitable compositions array, **arr_suitRC**, if not, another possible RC is generated

While analyzing a RC, if a resource is in **arr_notEl**, another possible RC is generated. If not, conditions (i) and (ii) are tested. If optimistic compositions are required, the generator stops when having i-compositions respecting T. However, if hybrid solutions are needed, the generator stops when having a composition that respects T, and contains only static resources (always available). SSA results are the set of RC included in **arr_suitRC**.

5 Evaluation and Discussion

In this section, we compare our QoR model to existing works, and evaluate our resource selection solution in a simulated environment provided by OpenCEMS. OpenCEMS offers 2 types of operation: real and simulated. As the real environment is currently being developed with a limited number of resources, in this chapter, we evaluated our work in the simulated functioning of OpenCEMS. The evaluation in the real environment will be presented in a dedicated work. In the experiments, we considered the requested composition type is hybrid, to focus on resources dynamcity aspect while forming the compositions. During the tests, conducted on a Linux Debian (64 bits) virtual machine, with 1 dedicated Intel® Core™ i7-46000 CPU @ 2.10GHz 2.70GHz processor and 1 GB RAM, we show the response time (in ms) based on an average of 5 sequential executions.

5.1 Comparison with Existing QoS Models

The comparison between our QoR model against the QoS model of existing works [2, 4, 14], is done independently of the number and type of the QoS used, since our work supports various attributes as long as they are presented in resources descriptions. The work in [2] uses 3 attributes: Performance: [0-10], Availability: [0-100], and Reputation: [0-5]. During selection and based on user constraints, a service is chosen over a candidate service if all of its QoS are equal or better than the other, preventing thus having optimistic/hybrid compositions having acceptable overall QoS. Also, and besides neglecting I/O matching of the linked services, by applying our threshold formula, a service with a high attribute value (Availability = 90) and a very low value for another one (Performance = 2) is selected over a service with acceptable values for both attributes (Availability = 70 and Performance = 6), as QoS are not normalized. In [4], several QoS are used to describe the services as Response Time and Availability. Contrary to our work, user constraints are given to the global composition (e.g., the overall response time should be < 50s) and not to each service, thus, aggregation functions are used for every QoS parameter. Moreover, weights are given to each QoS while computing the composition score. However, in our approach, user constraints are given to each of the involved services, and weights are assigned to (i) the overall services score and (ii) the overall I/O score that is not considered in [4]. In [14], a service has a score based on the sum of weighted utility functions relative to each QoS attribute. Similar to our work, QoS are normalized and user constraints are given to each service. However, the work does not define a global composition score, since the service of a specific task with the highest score is selected. Therefore, and apart of neglecting I/O matching of related services, no optimistic/hybrid solutions with acceptable scores can be formed.

5.2 Resource Selection Performance

For the selection tests, we varied: (1) the number of overlapped resources per function, and (2) the number of functions required in the WM. For these 2 cases, we applied several scenarios: (i) All static resources in DRAG are eligible, (ii) 50% of the static resources in DRAG are eligible, and (iii) all DRAG resources are dynamic and eligible. For (i) and (ii), the first generated possible compositions (without score calculation) include dynamic resources, thus, the selection process continue generating compositions until having one including only static resources with acceptable score. In the best cases, static resources are traversed first, and the selection process responds more rapidly. In the tests, each resource has 2 inputs and 2 outputs, and user constraints are given to: Dynamicity (where static and dynamic resources can be selected), Availability, Cost, and Usage. In the experiments, we assumed that the I/O similarity score between a resource and another related one is $\geqslant 0.5$. In Figure 3, in which the workflow consists of 3 functions, response time evolves with the growing number of resources.

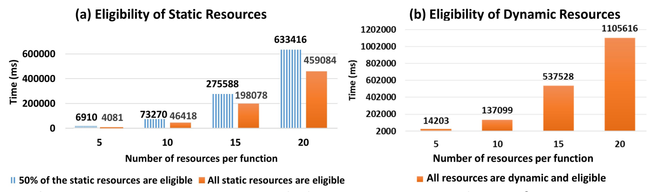


Fig. 3: Response time while varying number of resources

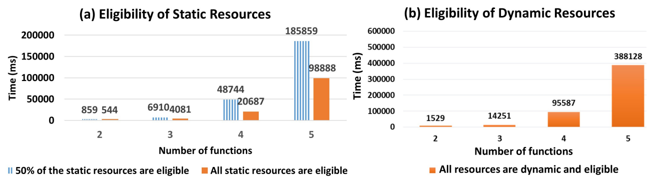


Fig. 4: Response time while varying number of functions

Comparing Figure 3-(a) to Figure 3-(b), response time increments less significantly with the existence of static eligible resources, as the selection process stops when having a composition of static resources. Figure 3-(b) represents the worst case scenario, where DRAG contains only dynamic resources that are eligible. Thus, all compositions scores are computed, causing an important response time with the evolution of resources number. Figure 4-(a), in which resource number per function is fixed to 5, shows that response time increases with the number of required functions. Similar to the first case, the selection process is faster when there are static eligible resources, as the selection process stops before the generation of the rest of the possible compositions. As for Figure 4-(b), response time evolves significantly, since DRAG contains dynamic eligible resources, thus, all compositions scores are calculated. The results highlight the importance of the existence of static eligible resources in DRAG, as the selection process stops when having a composition including static resources with an acceptable score. When dynamic resources exist, the response time increases, as their dynamicity is considered. The results also show that the increasing number of resources affects more the response time, comparing to the number of functions.

6 Conclusion

This chapter presents a QoR-driven resource selection approach that forms i-compositions (i $\in \mathbb{N}^*$) in Hybrid Web environments connecting static resources (always available), and dynamic resources (connected/disconnected at different instances). To do so, we defined a Selection Strategy Adapter that considers different compositions types (e.g., Optimal and Optimistic) and subtypes (e.g., Cost-free), QoR constraints, as well as resource I/O matching and dynamicity. Analysis are made to compare our QoR model with existing works, and tests are conducted to study our solution in a simulated Web environment offered by OpenCEMS. In the future, we plan to evaluate our work in the OpenCEMS real functioning, while considering subtypes compositions. We also seek to deploy an automatic orchestration to execute the compositions.

References

1. Payam Barnaghi, Amit Sheth, and Cory Henson. From data to actionable knowledge: Big data challenges in the web of things [guest editors' introduction]. *IEEE Intelligent Systems*, 28(6):6–11, 2013.
2. Mahdi Bennara et al. Linked service selection using the skyline algorithm. In *MEDI*, pages 88–97. Springer, 2016.
3. Karim Benouaret et al. Top-k web services compositions: A fuzzy-set-based approach. In *ACM-Symp. on Applied Computing (SAC)*, pages 1038–1043, 2011.
4. Rainer Berbner et al. Heuristics for qos-aware web service composition. In *ICWS'06*, pages 72–82. IEEE, 2006.
5. Shuiguang Deng et al. Service selection for composition with qos correlations. *IEEE Transactions on Services Computing*, 9(2):291–303, 2014.
6. Shuiguang Deng et al. Top-k automatic service composition: A parallel method for large-scale service sets. *IEEE T-ASE*, 11(3):891–905, 2014.
7. Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Irvine, USA, 2000.
8. Katawut Kaewbanjong and Sarun Intakosum. Qos attributes of web services: A systematic review and classification. *Journal of Advanced Management Science Vol*, 3(3), 2015.
9. Markus Lanthaler and Christian Gütl. Hydra: A vocabulary for hypermedia-driven web apis. *LDOW*, 996, 2013.
10. Freddy Lécué and Alain Léger. A formal model for semantic web service composition. In *International semantic web conference*, pages 385–398. Springer, 2006.
11. Yohei Murakami, Donghui Lin, and Toru Ishida. *Services Computing for Language Resources*. Springer, 2018.
12. Suphakit Niwattanakul et al. Using of jaccard coefficient for keywords similarity. In *Proceedings of IMECS*, volume 1, pages 380–384, 2013.
13. Pablo Rodriguez-Mier et al. An integrated semantic web service discovery and composition framework. *IEEE TSC*, 9(4):537–550, 2015.
14. Hongbing Wang et al. Effective bigdata-space service selection over trust and heterogeneous qos preferences. *IEEE TSC*, 11(4):644–657, 2015.
15. Lijuan Wang et al. A survey on bio-inspired algorithms for web service composition. In *Proceedings of the 2012 IEEE CSCWD*, pages 569–574. IEEE, 2012.
16. Xiaofei Xu et al. Novel artificial bee colony algorithms for qos-aware service selection. *IEEE Transactions on Services Computing*, 12(2):247–261, 2016.