Journal of Universal Computer Science, vol. 17, no. 5 (2011), 712-741
submitted: 8/2/10, accepted: 4/11/10, appeared: 1/3/11

QoS-based Approach for Dynamic Web Service Composition

Frederico G. Alvares de Oliveira Jr.
(Tecnological Institute of Aeronautics - ITA
São José dos Campos-SP, Brazil
fred@ita.br)

José M. Parente de Oliveira
(Tecnological Institute of Aeronautics - ITA
São José dos Campos-SP, Brazil
parente@ita.br)

Abstract: Web Services have become a standard for integration of systems in distributed environments. By using a set of open interoperability standards, they allow computer-computer interaction, regardless the programming languages and operating systems used. The Semantic Web Services, by its turn, make use of ontologies to describe their functionality in a more structural manner, allowing computers to reason about the information required and provided by them. Such a description also allows dynamic composition of several Web Services, when only one is not able to provide the desired functionality. There are scenarios, however, in which only the functional correctness is not enough to fulfill the user requirements, and a minimum level of quality should be guaranteed by their providers. In this context, this work presents an approach for dynamic Web Service composition that takes into account the composition overall quality. The proposed approach relies on a heuristics to efficiently perform the composition. In order to show the feasibility of the proposed approach, a Web Service composition application prototype was developed and experimented with public test sets, along with another approach that does not consider quality in the composition process. The results have shown that the proposed approach in general finds compositions with more quality, within a reasonable processing time.
Key Words: Semantic Web Service, Web Service Composition, Quality of Service
Category: H.3, H.3.5

1 Introduction

A web service can be defined as a piece of software that conforms to a set of open interoperability facilities [Erl(2007)], such as WSDL (Web Service Description Language) [Christensen et al.(2007)], SOAP (Simple Object Access Protocol) [Box et al.(2000)], and UDDI (Universal Description, Discover and Integration) [OASIS(2004)], for description, messaging protocol and discovering, respectively. These facilities allow the integration of systems written in different languages and running on computers with different platforms. However, all of them offer only a common grammar for describing, publishing and exchanging information of web service and workflows, which means that there is a lack of semantics in the information provided by services defined with those facilities [Akkiraju(2007)].

The Semantic Web is intended to extend the current web in order to make it more meaningful so that machines can interpret the information that flows across the web. In practice, Semantic Web relies on a formal description framework called ontology to describe the information produced by web resources (webpages, services etc) in a more structured manner. As a consequence, current web services (also called syntactic) may also be a beneficiary of this structured description, since their properties, capabilities, interfaces, and effects can be encoded in an unambiguous, and machine-interpretable fashion. A web service with such an encoding is known as Semantic Web Service [Mcilraith et al.(2001)].

Generally, a Semantic Web Service (SWS) is a service whose input, output, preconditions and effects (IOPEs) are associated with a formal description rather than just a datatype. A straightforward impact of having Web Services semantically annotated is the fact that their functionality can be automatically matched (discovered) and composed with functionalities provided by other services when only one service is not able to produce the desired functionality. Consequently, new functionalities or applications can be dynamically built based on existing services. By dynamic we mean that one application can be conceived automatically on demand according to users' requests by composing several services.

Sometimes, however, only the matching of service functionalities based on input and output description is not enough to fulfill the users requirements. A service user, i. e. a person or another computer, may impose constraints on the required service. For this purpose, services should guarantee a minimum level of quality delivered to their consumers. The quality of a service is often expressed by means of non-functional attributes, such as performance, availability, cost etc.

Lately, research on automatic SWSs discovery and composition has been receiving a lot of attention. However, there are not many works that deal with automatic web service composition, and make use of quality of service (QoS) constraints [Zeng et al.(2004), Aversano et al.(2004), Canfora et al.(2005), Claro et al.(2008), Aggarwal et al.(2004)] at the same time. Considering such constraints while composing web services is not a trivial task because when dealing with composite services, i.e., services that are composed of two or more other services, individual quality must be used in order to determine an overall quality of the composite service.

According to [Tsetsos et al.(2007)] service discovery consists in finding a service that matches a given set of requirements. When dealing with SWSs, service matching should be based on ontology description. A composition of web services, in turn, is a practice where new services are built relying on several other services to deliver a desired functionality [Casati and Shan(2001)]. A dynamic web service composition is defined as a task of discovering a set of services that together can fulfill a given requirement, when a single service alone is not able to do so [Sirin et al.(2004)]. This requirement is expressed by means of input and output parameters and QoS constraints.

There is a body of approaches [Zeng et al.(2004), Canfora et al.(2005), Claro et al.(2008)] that perform discovery of SWSs considering QoS based on a predefined high-level composition definition. By following this strategy it is possible to dynamically select one service for each item of the composition so that a composition meet the overall QoS imposed constraints. However, those approaches are more focused on providing the best quality values for a predefined flow of tasks.

Another body of approaches [Weise et al.(2008), Yan et al.(2008)], on the other hand, provides efficient algorithms that make use of Artificial Intelligence techniques in order to find a set of services for a given requirement, without considering QoS. Other approaches [Sirin et al.(2004), Aversano et al.(2004), Oh et al.(2007)] consider QoS during composition, but do not make use of any efficient technique and relies on an exhaustive search over the available services. This kind of strategy certainly become infeasible when a great number of services are made available.

In this paper, we propose an approach for semantic web service composition that considers QoS constraints during the composition. The work extends our previous work [de Oliveira Jr. and de Oliveira(2009)] by providing a more detailed view of the approach and presenting the experimental results. The algorithm presented follows a heuristic approach previously proposed in [Weise et al.(2008)].

The remainder of this paper is organized as follows. [Section 2] presents the related works. [Section 3] describes the proposed approach for dynamic web service composition. [Section 4] presents a motivating scenario to illustrate how the approach can be applied in a practical context. [Section 5] shows the experiments performed in order to validate the proposed approach. [Section 6] concludes the paper and presents the future works.

2 Related Works

In this section, a selected set of relevant works related to web service composition is discussed. In order to establish a comparison among all investigated approaches, a set of features that covers most of web service composition approaches was taken into consideration as parameters for comparison, as follows:

– **Dynamic Composition -** The approach provides a way to find a structured set of service according to end users' requests. Some classical approaches assume that a structure will be provided, while others assume human interactions. This parameter is used to distinguish fully dynamic and automatic composition approaches from the others;

- **Parallel Composition -** The approach takes into account that two or more services can be executed at the same time if there is no dependence between them. This should be taken into account because it can increase the time to find a solution, but once a solution is found, the execution time (and other QoS attributes) can be improved comparing to sequential solutions;
- **Quality of Service -** The approach deals with QoS constraints. As QoS is mechanism that measure the end users' satisfiability on a given service or the conditions to consume it, we believe that it is also a very important issue to be considered;
- **Fast Search -** The approach makes use of efficient search algorithms to find a composition. This is a important feature, because the time to find a solution depends on the search strategy used;
- **Optimal Solution -** The approach is able to find the optimal solution. This feature is very important, since we can take into account the trade-off between the search strategy and the solution optimality.

In [Sirin et al.(2004)], it was proposed an interactive approach for Semantic Web Service Composition that allows users to build workflows based on semantic web services. This approach makes use of OWL-S to semantically filter and select the services for a composition. First, the composer lists all available services in a repository. When the user selects one, the composer lists all services whose outputs feed the chosen service as inputs. This approach makes use of subsumption-based matching (i.e. a concept a subsumes another concept b if a is a superclass of b) to filter services that can provide a concept as input for the current service at each step. The composer also allows users to invoke the completed composition by inserting values as inputs.

The main advantage of this proposal is the possibility to build and test fragments of compositions as they are built. However, this process is semi-automated and totally dependent on human interaction. Therefore, it may be infeasible for large service repositories. Moreover, this approach does not consider QoS constraints.

In [Pistore et al.(2005)], an automatic approach for synthesis of Business Process Execution Language (BPEL) [BPEL (2007)] components was proposed. The BPEL components are modeled in State transition system (STS) and the user requirements on the composition are expressed in a specific requirement language. Given that, the synthesis is performed by using model checking. This approach has the advantage of allowing the use of complex compositional structures of BPEL like parallel composition and conditional branches. In fact, as the authors state, the work focuses on the assistance to the task of building business process structures, rather than dynamic web service composition. Thus, QoS issues are not addressed in that work. Moreover, according to the results, even for small compositions (e.g. less than ten) the approach take a considerable amount of time (more than ten seconds) to compose components. Although it can be considered acceptable at design time, it may be unacceptable for end users at runtime.

In [Aversano et al.(2004)], an extension of the Semantic Web Service Composition algorithm described in [Akkiraju et al.(2003)] was proposed in order to consider non-functional properties such as Response Time, Cost and Availability. Services are discovered based on three different levels of similarity: (i) lexical level, where the tool Wordnet [Fellbaum(1998)] is used to identify synonymous terms (e. g. Clinic and Hospital); (ii) property level, where the properties (object properties and datatype properties) of two concepts are compared; (iii) the semantic level, where class relations such as subsumption-based ones are used. These three different levels are useful to resolve similarities of concepts defined in different ontologies.

The composition algorithm adopts a backward strategy and then starts by finding services whose outputs match with sufficient similarity to the required outputs (goal). Next, the concepts needed as inputs of the found services are made as the new goal, and the algorithm finds services that are able to provide outputs that match with this new goal. These steps are repeated until the service inputs can be provided by the inputs provided by the request. It is also considered when two or more independent services are used to provide a required goal or when it exceeds a beforehand specified timeout. When more than one service can provide a given goal, the one with best QoS values is chosen.

The algorithm also distinguishes sequential (called vertical) and horizontal (called parallel) compositions so that it is possible to make a right computation of QoS values. For example for Response Time in a parallel composition should be the maximum among services in the composition. So these services are composed in a horizontal way (parallel composition).

This approach has several advantages. Besides performing a dynamic web service composition considering the QoS criteria, it also provides both sequential and parallel composition. On the other hand, a drawback of this approach is the fact that the algorithm performs an exhaustive search in the repository. In fact, the solution proposed is similar to an Iterative Depth-first Search approach [Russell and Norvig(2003)] which becomes infeasible as the number of available service increases.

An algorithm for semantic and syntactic web service composition was proposed in [Oh et al.(2007)]. The problem is modeled into an Artificial Intelligence Planning Problem [Russell and Norvig(2003)], where the initial state corresponds to the set of inputs provided in the request, the target (goal) state is such that the set of matched services are able to provide all the required outputs specified in the request, and the available services correspond to the search space. The cost of invocation of a service is fixed in 1.

The algorithm is divided into two phases: the first one performs a forward search, and thus starts from the initial state (input data provided in the request) and, following a subsumption reasoning, matches services that require this data as input to be invoked until all required outputs specified in the request can be provided by the matched services. Each matched service is added in a graph whose nodes store the cost needed to go from the initial state to it. The second phase performs a backward search in the graph produced in the first phase in order to get the optimal path, i.e. the path with minimum cost, from the start to the target node.

An advantage of this approach is that it is able to find the optimal solution. However, it performs a exhaustive search in order to obtain the optimal solution. In addition, even though the cost of executing a service is taken into account, QoS models are not mentioned in the algorithm.

In [Weise et al.(2008)], three different approaches for service composition were used and compared: (i) an Iterative Depth-First Search approach; (ii) a Greedy Approach; (iii) and an Evolutionary Approach [Russell and Norvig(2003)]. All the three approaches consider a subsumption-based matching of services, but none of them considers non-functional characteristics or QoS attributes. The Iterative Depth-First Search [Russell and Norvig(2003)] approach performs a backward search similar to that proposed in [Aversano et al.(2004)]. The Greedy Approach, which is based on Greedy Algorithm [Russell and Norvig(2003)], also performs a backward search, but relies on a heuristic function that takes into account some properties of the candidate compositions and always chooses the composition which, according to the heuristics, is the most appropriate. Finally, the third approach presents a Genetic Algorithm (GA) [Russell and Norvig(2003)] in order to take into account multiple objectives such as the size and the correctness composition and thus provide an optimal solution.

Experiments have shown that the heuristic-based approach outperformed (in execution time) the others in many situations. Although this approach does not guarantee neither the optimal nor the algorithm completeness, the experiments showed that the heuristics provided results approximated to the optimal solution and was able to provide solutions for all experiments performed. The Genetic Approach, on the other hand, guarantees the optimal solution but at the cost of a worse performance. Finally, the experiments showed that the Iterative Depth-First Search approach has an acceptable performance in small repositories, but fails in large ones.

In [Yan et al.(2008)], an approach for dynamic semantic web service composition based on AND-OR Graphs [Konar(2000)] was proposed. The problem is modeled into a Service Dependency Graph (SDG) in which two kind of nodes are allowed: services and data (inputs and outputs). Two service nodes are linked through a data node also following the subsumption relations. The approach implements the SDG in a AND-OR graph, where AND-nodes correspond to service nodes and OR-nodes corresponds to data nodes. Another dummy AND-node representing the goal (required outputs) is also created.

The algorithm performs a backward search starting from the dummy AND-node until it can be resolved, i.e. when all its input OR-nodes are also resolved. An OR-node can be resolved when an AND-node can provide it as output, and so on. The selection criterion of a node is the cost it represents from the dummy node to it. This cost is a fixed cost of a service and the number of inputs required by the service to be invoked. Thus, all the possible paths in the AND-OR Graph are taken into account, and the best one, that is, the optimal solution is chosen. In addition, it is also considered parallel composition of services rather than just sequences. All possible solutions are analyzed, but the overhead is mitigated by the use of a reverse index technique on the data provided by each service. Despite the cost of a service node is taken into account, no QoS model is mentioned as well as nothing is stated about multiple QoS criteria such as Availability, Reputation etc.

[Tab. 1] summarizes all the investigated works. The objective of our approach is to combine the advantages of other approaches, like the use of good search strategies like in [Weise et al.(2008)] while considering QoS during the composition, as proposed in [Aversano et al.(2004)].

Approach	**Dynamic**	**Optimal**	**Parallel**	**Fast Search**	**QoS**
Sirin et al.	No	Yes	Yes	No	No
Oh et al.	Yes	Yes	No	No	No
Weise et al.	Yes	No	No	Yes	No
Yan et al.	Yes	Yes	Yes	Yes	No
Aversano et. al.	Yes	Yes	Yes	No	Yes
Pistore et. al.	Yes	No	Yes	No	No

Table 1: Summarization of the related works main characteristics.

3 QoS-Based Approach for Dynamic Web Service Composition

Dynamic Web Service Composition consists in composing at runtime two or more services in order to solve a problem that cannot be solved by any service individually. This section aims to present an approach for Dynamic Web Service Composition [de Oliveira Jr. and de Oliveira(2009)]. The proposed approach extends the Heuristic-based Approach for Dynamic Web Service Composition proposed by [Weise et al.(2008)] by adding Quality of Service constraints to the heuristic.

The section is organized as follows. Firstly, the main concepts involved in the proposed approach are presented. Then, the algorithm is presented along with the proposed heuristic.

3.1 Proposal Overview

Semantic Web Service Composition consists in composing two or more well-described (ontology-based) services in order to accomplish a requested functionality. The proposed approach in this work is intended to perform Dynamic Service Composition taking into account both the semantic description of a service and its non-functional properties that composes the quality it delivers. The algorithm to perform this composition receives as input a request, which consists of the provided input concepts, required output concepts and QoS constraints, and produces as output a set of services that together can provide the required concepts specified in the request, as illustrated in [Fig. 1].

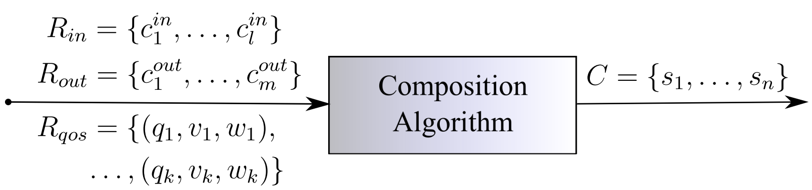

Figure 1: Proposal Overview.

Each concept in the request input $(c_1^{in}, \ldots, c_m^{in})$ or output $(c_1^{out}, \ldots, c_m^{out})$ is defined in a ontology named Domain Ontology. Each QoS constraint consists of a triple with the quality criterion, a value representing a constraint on this criterion and a weight representing the user's preference on this criterion. Following are some definitions extended from [Weise et al.(2008)] to incorporate QoS.

Definition 3.1 (Domain Ontology) *A domain ontology $\mathcal{O}$ consists of a set of concepts used to describe the domain in which a group of services is inserted. These concepts are related to each other according to subsumption relations, i. e. a concept $c_1 \in \mathcal{O}$ subsumes another concept $c_2 \in \mathcal{O}$ if c_1 is a superclass of c_2; c_1 and c_2 subsume each other if they are the same concept.*

Definition 3.2 (Request) *A composition request $\mathcal{R}$ consists of a set of provided inputs $\mathcal{R}_{in} \subseteq \mathcal{O}$, a set of required outputs $\mathcal{R}_{out} \subseteq \mathcal{O}$ and a set of quality of service constraints $\mathcal{R}_{qos} = \{(q_1, v_1, w_1), (q_2, v_2, w_2), \ldots, (q_k, v_k, w_k)\}$, where q_i $(i = 1, 2, \ldots, k)$ is a quality criterion, v_i is the required value for criterion q_i, w_i is the weight assigned to this criterion such that $\sum_{i=1}^{k} w_i = 1$, and k the number of quality criteria involved in the request. A quality criterion can be either negative, i.e. the higher the value the lower the quality, or positive, i.e. the higher the value the higher the quality.*

A service composition is built by discovering services available in a Service Repository that match with concepts required by either service inputs or request output. A service is selected to take part of certain composition if at least one of its output concepts is related to a required concept (request output or a required input of another service in the composition) according to the subsumption relations above mentioned.

A valid composition is a composition where all the required output concepts can be provided by its constituent services as well as each service input concepts should be provided by either the available concepts in the request or other services in the composition. In addition, a valid composition also must meet the imposed QoS constraints.

Definition 3.3 (Service Repository) *A Service Repository $\mathcal{D}$ consists of a set of available services. Each service $s \in \mathcal{D}$ is composed of a set of required inputs $s_{in} \subseteq \mathcal{O}$, a set of provided outputs $s_{out} \subseteq \mathcal{O}$ and a set of provided quality criteria $s_{qos} = \{(q_1, v_1), (q_2, v_2), \ldots, (q_k, v_k)\}$, where q_i $(i = 1, 2, \ldots, k)$ is a quality criterion, v_i is the value associated to criterion q_i provided by the service, and k is the number of criteria involved.*

Definition 3.4 (Service Composition) *A Service Composition $\mathcal{C}$ is a Direct Acyclic Graph (DAG) with vertices $\mathcal{C}_v = \{s \mid s \in \mathcal{D}\}$ and edges $\mathcal{C}_e = \{(u, v) \mid \forall u, v \in \mathcal{C}_v\ \exists c_1 \in u_{out}\ \exists c_2 \in v_{in} : c_2 \textsf{ subsumes } c_1\}$*

Definition 3.5 (Valid Composition) *A composition $\mathcal{C}$ is considered valid for a request $\mathcal{R}$ if the predicate $valid(\mathcal{C}, \mathcal{R})$ in Equation 1, holds.*

$$\begin{aligned} valid(\mathcal{C}, \mathcal{R}) \Leftrightarrow\ & \forall c_1 \in (\mathcal{R}_{out} \textstyle\bigcup_{\forall s \in \mathcal{C}_v} s_{in})\ \exists c_2 \in (\mathcal{R}_{in} \textstyle\bigcup_{\forall s \in \mathcal{C}_v} s_{out}) : c_1 \textsf{ subsumes } c_2 \\ & \wedge\ \forall (q, v, w) \in \mathcal{R}_{qos}\ (q \textsf{ is negative } \wedge overall(\mathcal{C}, q) \leq v) \vee \\ & \qquad (q \textsf{ is positive } \wedge overall(\mathcal{C}, q) \geq v)) \end{aligned} \tag{1}$$

where $overall(\mathcal{C}, q)$ *represents the model that calculates the overall value of criterion* q *in composition* $\mathcal{C}$.

The following subsections gives an overview of the Greedy Search strategy, used to construct the algorithm which is the basis of the proposed approach.

3.1.1 The Greedy Approach

Greedy Search Algorithm [Russell and Norvig(2003)] is a kind of best-first search algorithm which, by its turn, is a graph search algorithm where a node (vertex) is selected to be expanded based on an evaluation function $f(n)$. The evaluation function measures the distance between a node n to the target node. In a web service composition scenario, a target scenario would be a composition that holds for predicate *valid* (Equation 1). Thus, the node with lowest evaluation is usually selected. In a Greedy Search Algorithm, the evaluation function $f(n)$ is a heuristic function $h(n)$. A heuristic function is an estimation of the cost of the best path from a node n to the target node.

The greedy algorithm for service composition proposed by [Weise et al.(2008)] receives as input a request similar to that shown in Figure 1, but without the QoS constraints. The algorithm internally sorts a list of currently known states according to a comparator function $c_h(x_1, x_2) \in \mathbb{R}$. Here, states are possible solutions, that is, candidate compositions, and the target node is achieved when a candidate composition does not require any concept, i.e. all concepts required by either the request outputs and services inputs can be provided by the request input and services outputs.

The comparator function $c_h(x_1, x_2)$ compares the results of the heuristics applied to each state ($h(x_1)$, $h(x_2)$) and returns a negative value if $h(x_1) > h(x_2)$, a positive value if $h(x_2) > h(x_1)$ or zero if $h(x_1) = h(x_2)$. Thus, the last element of the (sorted) list will be the most suitable candidate solution and should be chosen to be expanded. In fact, instead of using the candidate's properties to build an heuristics, the comparator function is derived directly. And each property of one candidate is compared to the corresponding of the other candidate.

Every time a candidate solution cannot be expanded anymore, i.e. whenever there is no more services that can supply the concepts required by that composition, this composition is discarded and the second most suitable composition is chosen. This behavior is very similar to a Depth-first search [Russell and Norvig(2003)] and indicates the worst case of execution when all available states (candidates) achieve a stage where it is not possible to expand any state anymore. The time and memory complexity in the worst case is $O(b^m)$ where b is the maximum number successors of any nodes, namely, the number of services that can produce any concept required by the candidate composition, and m is the maximum depth of the search space, namely, the maximum number of services in a composition. Moreover, a greedy strategy is neither complete (it does not guarantee to find a solution when it exists) nor optimal (not always find the best available solution). It is not complete because can invest in a infinite path and never return to check other candidate solutions.

On the other hand, a good heuristics can lead good approximations to the optimal solution as well as to a real improvement in the algorithm complexity [Russell and Norvig(2003)]. Experiments performed in [Weise et al.(2008)] and in this work [see Section 5] show that.

Despite of the fact that the approach proposed by [Weise et al.(2008)] is suitable even for big repositories, the algorithm and heuristics does not take into account quality of service of the compositions, that is, the heuristics considers only properties of compositions that lead to a correct composition from the functional point of view. In addition, the algorithm does not restrain candidate compositions that no longer meet the constraints imposed in the request.

3.2 Quality of Service Models

As previously mentioned, the quality of a given service is expressed by means of nonfunctional properties. Although this work is not limited to any particular set of QoS criteria, a simplified version of QoS Models proposed in [Zeng et al.(2004)] is used. Other models such those proposed in [Cardoso(2002)] could also be used.

[Zeng et al.(2004)] consider five generic quality criteria for elementary (atomic) services, namely:

- **Duration** ($q_{du}(s)$) of a service s measures the expected delay between the moment when a request is sent and the moment when the result is received. It is calculated by summing the service processing time, which is given by the service provider as a fixed value or as a method to inquire about it; and the average of transmission times of past executions;

- **Reputation** ($q_{rep}(s)$) of a service s is a measure of its trustworthiness. It is calculated by the average of different end users' ranged ranking ([0,1]) on the service;

- **Availability** ($q_{av}(s)$) of a service s is the probability that it is accessible and

- **Price** ($q_{pr}(s)$) of a service s is the fee that its requesters have to pay for invoking it.

In addition to the means of computation of the criteria defined above, [Zeng et al.(2004)] also proposed an aggregation function for each criterion in order to

Criterion	**Function**
Reputation	$q_{rep}(\mathcal{C}) = \frac{1}{n}\sum_{i=1}^{n} q_{rep}(s_i)$
Price	$q_{pr}(\mathcal{C}) = \sum_{i=1}^{n} q_{pr}(s_i)$
Duration	$q_{du}(\mathcal{C}) = \sum_{i=1}^{n} q_{du}(s_i)$
Availability	$q_{av}(\mathcal{C}) = \prod_{i=1}^{n} q_{av}(s_i)$
Successful Execution Rate	$q_{rat}(\mathcal{C}) = \prod_{i=1}^{n} q_{rat}(s_i)$

Table 2: Aggregation Functions for QoS Models

provide the QoS Models for composite services. Although that work also provides models for computing quality values in composition with loop and conditional choices constructs, this work omits these models since they go beyond its scope.

The aggregation functions used in this work are described in [Tab. 2], for a given composition $\mathcal{C}$ where $\mathcal{C}_v = \{s_1, \ldots, s_n\}$. The Overall Reputation model is given by the average of all constituent services' reputation. The Overall Price, in turn, is calculated simply by summing the price of all constituent services.

Unlike the Duration model defined in [Zeng et al.(2004)], where the critical path (i.e. the more time-consuming path in the composition DAG) is considered, the Overall Duration model was defined in [Tab. 2] as a sum of all constituent services duration of a given composition. The reason is that this work does not consider parallel compositions, which allows multiple execution paths. After all, a sequential composition will be always the critical path.

Similarly, simplified versions of Availability and Successful Execution Rate models were also defined in [Tab. 2]. Since parallel compositions are not considered in this work, both of them can be redefined as a simple product of factor of probability without taking into account whether or not a service is part of a critical path.

Only by looking at the models defined in [Tab. 2] it is possible to see the consequences of not considering parallelism. Indeed, the execution time can be increased and the availability decreased. The difficulty is in how to calculate the model during the composition, since it will be necessary to maintain a graph structure and apply a critical path algorithm, while in a sequence structure, although it is represented as graph as well, its implementation is much simpler so that the QoS model is much easier to be calculated.

3.2.1 The Composition Algorithm

The Algorithm 1 performs a service composition for a given request $\mathcal{R}$. It starts by finding all services that can provide as output a concept which is semantically equivalent (by subsumption reasoning) to the required outputs specified in

$\mathcal{R}_{out} \in \mathcal{R}$ (Algorithm 1, lines 2-9). A service will be a candidate solution if it can meet the QoS constraints specified in $\mathcal{R}_{qos} \in \mathcal{R}$. The predicate $meetQoS(\mathcal{R}, \mathcal{C})$ (Algorithm 1, lines 5-7) filters the candidate compositions that no longer meet the constraints imposed in the request and therefore should be discarded. The Equation 1 cannot be used since in some criteria, such as Reputation, it is not possible to determine if a certain composition does not meet the constraints imposed by the request without knowing the final number of services in the composition, because its value is determined by the average of the services' individual values. In such cases, this kind of criterion is just ignored and is not checked.

The candidate compositions are sorted according to a comparator function which compares two candidate compositions $\mathcal{C}_1$ and $\mathcal{C}_2$ and return a value below zero if $\mathcal{C}_2$ is a more suitable composition than $\mathcal{C}_1$, above zero if $\mathcal{C}_1$ is a more suitable composition than $\mathcal{C}_2$, and zero if both are equally evaluated (Algorithm 1, line 11). The comparator function proposed by [Weise et al.(2008)] considers four composition properties: (i) *known* concepts; (ii) *unknown* concepts; (iii) *eliminated* concepts; and the number of services in a composition. The property *known* (Equation 2) of a given composition $\mathcal{C}$ and request $\mathcal{R}$ is the set of all known concepts, i.e. the input concepts provided by the requester and output concepts provided by the output of all services presented in composition $\mathcal{C}$. The property *unknown* (Equation 3) of a given composition $\mathcal{C}$ is a set of the required concepts needed to make the composition $\mathcal{C}$ (functionally) valid. This set is composed of the concepts required in the request and concepts required to execute each service in the composition $\mathcal{C}$. The property *eliminated* (Equation 4) is a set of unknown concepts that has already been provided.

$$known(\mathcal{C}) = \mathcal{R}_{in} \bigcup_{\forall s \in \mathcal{C}_v} s_{out} \tag{2}$$

$$unknown(\mathcal{C}) = \mathcal{R}_{out} \bigcup \{c \mid \exists s \in \mathcal{C}_v : c \in s_{in} \wedge c \notin known(\mathcal{C})\} \tag{3}$$

$$eliminated(\mathcal{C}) = \{c \mid \exists s \in \mathcal{C}_v : c \in s_{in} \wedge c \in known(\mathcal{C})\} \tag{4}$$

In this work, a fifth property was added: The Overall Quality of Service Score. This property aims at representing in a single value all the quality criteria values of a composition. For this purpose, the Multiple Criteria Decision Making (MCDM) [Yoon et al.(1995)] named Simple Additive Weighting (SAW) [Yoon et al.(1995)] technique is used. This technique, also used in [Zeng et al.(2004)], allows the calculation of a score taking into account several, and sometimes conflicting, criteria. First, all criteria are put in the same scale. Equations 5 and 6 provides a function that calculates the scaled value of a criterion q of composition $\mathcal{C}$ considering if q is a positive (5) or negative (6).

Input: $\mathcal{R}$ - the user Request, $\mathcal{D}$ - the Service Repository
Result: $\mathcal{C}$ - the composition found or $\emptyset$
Data: X - the set of candidate compositions
begin
    foreach $outR \in \mathcal{R}_{out}$ **do**
        foreach $s \in \mathcal{D} \mid \exists\ c \in s_{out}(outR\ \textsf{subsumes}\ c)$ **do**
            $\mathcal{C}_v \longleftarrow \{s\};$
            if $meetQoS(\mathcal{R}, \mathcal{C})$ **then**
                $append(X, \mathcal{C});$
            end
        end
    end
    while $X \neq \emptyset$ **do**
        $sort(X, comparator);$
        $\mathcal{C} \longleftarrow removeMostSuitable(X);$
        if $valid(\mathcal{C}, \mathcal{R})$ **then**
            return $\mathcal{C};$
        end
        foreach $outR \in unknown(\mathcal{C})$ **do**
            if $\neg\exists s \in \mathcal{D}(\exists\ c \in s_{out}(outR\ \textsf{subsumes}\ c))$ **then**
                break;
            end
            foreach $s \in \mathcal{D} \mid \exists\ c \in s_{out} \cdot (outR\ \textsf{subsumes}\ c)$ **do**
                $new_v \longleftarrow \mathcal{C}_v \cup \{s\};$
                $new_e \longleftarrow \mathcal{C}_e \cup_{\forall s2\ \in\ c_v(\exists\ c2\ \in\ s2_{in} \cdot\ c_2\ \textsf{subsumes}\ outR)} \{(s, s2)\};$
                if $meetQoS(\mathcal{R}, new)$ **then**
                    $append(X, new);$
                end
            end
        end
    end
    return $\emptyset;$
end

Algorithm 1: Composition Algorithm Considering the QoS Criteria.

$$scale(\mathcal{C}, q) = \begin{cases} \frac{overall(\mathcal{C},q) - q_{min}(q)}{q_{max}(q) - q_{min}(q)} & \text{if } q_{max}(q) - q_{min}(q) \neq 0 \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

$$scale(\mathcal{C}, q) = \begin{cases} \frac{q_{max}(q) - overall(\mathcal{C},q)}{q_{max}(q) - q_{min}(q)} & \text{if } q_{max}(q) - q_{min}(q) \neq 0 \\ 1 & \text{otherwise} \end{cases} \tag{6}$$

where $q_{max} = Max\{overall(\mathcal{C}, q) : \mathcal{C} \in X\}$ and $q_{min} = Min\{overall(\mathcal{C}, q) : \mathcal{C} \in X\}$ and X is the list of candidate compositions.

The overall QoS value of a composition considering all quality criteria can be determined by a score which takes into account the scaled values and the weights provided in the request $\mathcal{R}$ associated with each criterion. Equation 7 defines the function that calculates the Overall QoS Score for a given composition $\mathcal{C}$.

$$score(\mathcal{C}) = \sum_{\forall(q,v,w)\in\mathcal{R}_{qos}} scale(\mathcal{C}, q) * w \tag{7}$$

Originally, the comparator function choose one of the compositions that has no *unknown* concepts left. If both compositions does not need any *unknown* concept left, it returns the composition that has less services. If both of them have *unknown* concepts, the function returns the composition with more *eliminated* concepts. But, if both have the same number of *eliminated* concepts, it returns the composition that has less *unknown* concepts. If both of them have the same number of *unknown* concepts, it returns the composition with less services. But if both of them have the same number of services, the function returns the composition with more *known* concepts.

The comparator function now should also take the Overall QoS Score into account, along with the other four composition properties (*unknown* concepts, *eliminated* concepts, composition size, and *known* concepts). A number of experiments were performed in order to figure out the best place in terms of priority to put the QoS Score property in the comparator function without affecting the algorithm performance. The comparator function works as follows:

1. **unknown = 0 (↑) -** The comparator function returns the composition that has no *unknown* concepts left, that is, the composition that is functionally valid.
   - 1.1. **score (↑) -** The Overall QoS Score takes place as a tiebreaker when both compositions have the same number of *unknown* concepts. Thus, when both compared compositions have no *unknown* concepts left, the compositions with higher score is chosen.
     - 1.1.1. **size (↓) -** If both compositions have the same score, the one with less services is chosen.
2. **eliminated (↑) -** The composition with more eliminated concepts is chosen when both compositions still have *unknown* concepts.
3. **unknown (↓) -** When both compositions have the same number of *eliminated* concepts, the one with less *unknown* concepts is chosen.
4. **score (↑) -** The composition with higher score is chosen, when both compositions have the same number of *unknown* concepts.

5. **size (↓) -** When both compositions have the same score, the one with less services is chosen.

6. **known (↑) -** Finally, if both compositions are of the same size, the composition with more *known* concepts is chosen.

Once the list is sorted using the comparator function, it is possible to choose the composition which is the most appropriate in terms of both functionality and Quality of Service (Algorithm 1, line 12).

If the chosen composition is valid, that is, if the composition does not need any additional concept, to supply the services inputs or request outputs, and meets the constraints imposed in the request, it is returned by the algorithm (Algorithm 1, lines 13-15). Otherwise, the chosen composition is expanded to form new candidate compositions with services that provide concepts required by the composition (Algorithm 1, lines 16-27). Again, if the actual composition does not meet the QoS constraints, it is discarded (Algorithm 1 lines 23-25). If at least one unknown concept cannot be provided with the services in the repository this composition is also discarded (Algorithm 1, lines 17-19). The algorithm runs until a candidate solution is found or all candidate solutions are expanded and rejected, which is the worst case of execution.

If the Overall QoS Score property had preference against the *unknown* and *eliminated*, the algorithm would perform similar to a Breadth-first search. For instance, suppose the set of criteria Price, Duration and Availability. The compositions with less services would more likely have higher scores, since it would be cheaper, more available and with less time-demanding. As a consequence, these compositions would be always selected to be expanded. However, as the composition is expanded, it gets larger, decreasing its chance to be chosen in the next time.

On the other hand, by considering the Overall Score as a tiebreaker when properties *eliminated* and *unknown* of the compared compositions are equal, the search process will not be affected very much in terms of performance. The advantage then is that it will always expand the composition with higher score. Experiments [see Section 5] show a significant improvement in the QoS of the composition by simply considering the Overall QoS Score in the comparator function.

4 Motivating Scenario

This section describes a scenario elaborated by the Brazilian Air Force [Marques et al.(2010)] and how the proposed approach is suitable for it.

4.1 Scenario Description

The most common involvement of the Brazilian armed forces outside conventional military operations occurs in the form of humanitarian relief operations. To reflect this fact, the 2008 Santa Catarina flooding [SC-Flood (2008)] where the whole Itajaí's valley was completely flooded for a month and an enormous effort took place to provide assistance to the affected population, was chosen as a case. During this event, it was necessary to mobilize a small Numbered Air Force to ensure efficient coordination of the air transportation assets and to establish a supply corridor in order to receive the medical supplies, food and clothes to the affected population.

The main goal of the application is to make available to all organizations involved with the flood situation the following: available assets (ground-based, fluvial and aerial); location of all support installations; field solicitations for rescue missions and transport of supplies; and the schedule of the assets movement throughout the affected area. All those informations are made available by web services. However, as one organization has no previous knowledge about the access to services of another organization and can make available new services at any time, a small ontology is used as the domain descriptor and will be used to establish the basic concepts present in the semantic service descriptions. The idea is that organizations can query the information necessary to plan its missions on-demand. To this end it might be also necessary to compose several services.

4.2 Application of The Proposed Approach

The main elements of the scenario are modeled as follows:

- **Ontology.** The domain ontology $\mathcal{O}$, based on the relevant information about the events is given in the form of a taxonomy, as shown in [Fig. 2].
- **Repository.** The service repository $\mathcal{D}$ is the set of web services made available by the organizations involved in the flood. Their description consists of Inputs and Outputs, parameters associated to concepts of ontology $\mathcal{O}$ and values for each quality criterion, as shown in [Tab. 3]. As can be seen, there are some services with similar or the same purpose. For example, both WeatherCond1 and WeatherCond2 aim at providing the weather conditions of a given address, but one via GSM and the other one by using the GSM network.
- **Request.** The request $\mathcal{R}$ has a set of available inputs $\mathcal{R}_{in} = \{Victim, Weight, Address\}$, set of required outputs $\mathcal{R}_{out} = \{Route\}$ and a vector of quality $\mathcal{R}_{qos} = \{(q_{du}, 175, 0.2), (q_{av}, 0.55, 0.2), (q_{rep}, 0.8, 0.2), (q_{pr}, 1.4, 0.2), (q_{rat}, 0.55, 0.2)\}$, where q_{du} is Duration, q_{av} is Availability, q_{rep} is Reputation,

q_{pr} is Price and q_{rat} is Successful Execution Rate. This request expresses a scenario where the user wants to the system to give him/her a Route to transport a certain Victim with a certain Weight to a certain Address

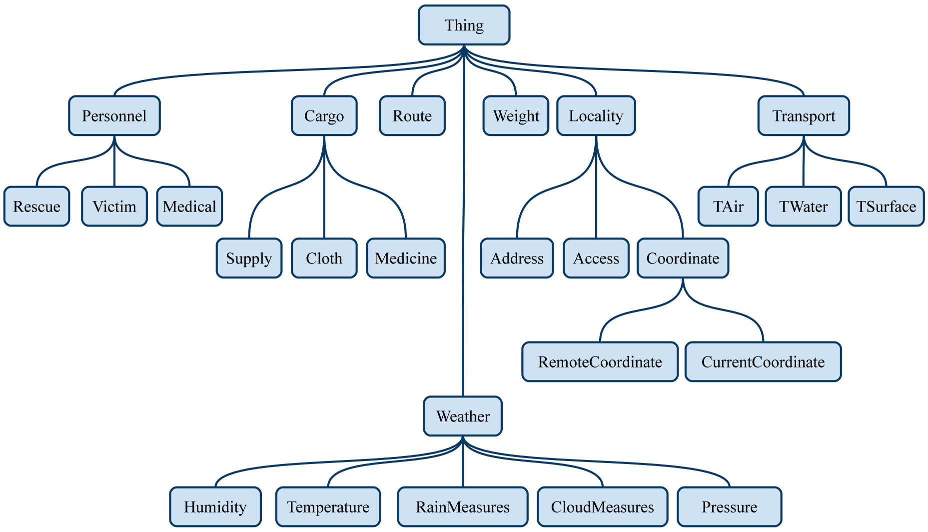

Figure 2: Domain Ontology of the Motivating Scenario.

The algorithm starts by trying to find out the services that produce as output concepts that are related under subsumption reasoning to the concept Route. Service TraceRoute is then selected as a candidate composition, as shown in [Tab. 4], where e is the number of eliminated concepts, u is the number of unknown concepts and k is the number of known concepts. At that moment, the algorithm will search for services that provide as output the concepts needed to execute service TraceRoute which are Transport, CurrentCoordinate or RemoteCoordinate. The second iteration will provide as result the candidate compositions shown in [Tab. 5]. The process will be iteratively repeated until no more *unknown* concepts (u) is necessary and the composition is valid. In the last iteration two compositions are functionally valid: {CurrCoordGPS, WeatherCond2, GetAccess, GetRemCoord, GetPersTransp, TraceRoute} and {CurrCoordGSM, WeatherCond2, GetAccess, GetRemCoord, GetPersTransp, TraceRoute}, as shown in [Tab. 6]. However, the first one has a score higher than the second one. Moreover, the overall duration of the second composition is 179 i.e. it does not meet the constraints imposed in the request.

Service	s_{in}	s_{out}	q_{du}	q_{av}	q_{rep}	q_{pr}	q_{rat}
GetAccess	RemoteCoordinate RainMeasures CloudMeasures	Access	45	0.9	0.92	0.15	0.95
GetRemCoord	Address	RemoteCoordinate	32	0.95	0.9	0.1	0.87
TraceRoute	Transport CurrentCoordinate RemoteCoordinate	Route	19	0.96	0.86	0.25	0.99
GetPersTransp	Personnel Weight Access	Transport	35	0.85	0.95	0.07	0.9
GetCargoTransp	Cargo Weight Access	Transport	40	0.91	0.97	0.17	0.9
WeatherCond1	Address	RainMeasures CloudMeasures	30	0.89	0.99	0.2	0.95
WeatherCond2	Address	RainMeasures CloudMeasures Temperature Humidity Pressure	23	0.92	0.95	0.4	0.86
CurrCoordGPS		CurrentCoordinate	16	0.98	0.9	0.33	0.94
CurrCoordGSM		CurrentCoordinate	25	0.89	0.9	0.3	0.96

Table 3: Service repository of the Motivating Scenario ($\mathcal{D}$).

It should be noticed that other solutions could still be composed. For example, if the service WeatherCond1 had better QoS values than WeatherCond2, it could as well be selected to take part in the composition.

Candidates	e	u	k	q_{du}	q_{av}	q_{rep}	q_{pr}	q_{rat}	**QoS Score**
{TraceRoute}	1	3	3	19,00	0,96	0,86	0,25	0,99	1,00

Table 4: Candidate compositions in the first iteration.

Candidates	e	u	k	q_{du}	q_{av}	q_{rep}	q_{pr}	q_{rat}	QoS Score
{GetPersTransp,TraceRoute}	4	3	3	54,00	0,82	0,90	0,32	0,89	0,45
{GetCargoTransp,TraceRoute}	3	4	3	59,00	0,87	0,92	0,42	0,89	0,48
{CurrCoordGPS,TraceRoute}	2	2	4	35,00	0,94	0,88	0,58	0,93	0,56
{CurrCoordGSM,TraceRoute}	2	2	4	44,00	0,85	0,88	0,55	0,95	0,41
{GetRemCoord,TraceRoute}	3	2	4	51,00	0,91	0,88	0,35	0,86	0,40

Table 5: Candidate compositions in the second iteration.

Candidates	e	u	k	q_{du}	q_{av}	q_{rep}	q_{pr}	q_{rat}	QoS Score
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
{CurrCoordGPS,...,TraceRoute}	12	0	10	170,00	0,63	0,91	1,30	0,60	0,20
{CurrCoordGSM,...,TraceRoute}	12	0	10	179,00	0,57	0,91	1,27	0,61	0,16

Table 6: Candidate compositions in the last iteration.

5 Experiments

This section aims at describing the experiments used to validate the approach presented in [Section 3]. The main purposes of these experiments are to analyze the feasibility of the proposed approach concerning the processing time as well as the benefits of its use in terms of quality of service. The experiments were performed in a computer with processor Intel Core 2 Duo - 2.16 GHz, with 2GB of RAM. The remaining parts of this section is organized as follows. Firstly, the experimental environment created to perform the experiments is explained. Next, the results obtained from those experiments are shown. Finally, the experimental results are discussed and analyzed.

5.1 Benchmark

The experiments were performed under a variety of test sets from the Web Service Challenge (WSC) [Blake et al.(2007)]. These test sets consist of groups of repositories, ontologies, requests and solutions; namely, for each repository, there is an ontology associated, a set of requests and their respective solutions, as illustrated in [Tab. 7]. In WSC, the service specification was limited to required inputs and provided outputs, so Quality of Service was not considered in this competition. Thus, it was necessary to prepare the WSC test sets in order to

Request	No of Services	No of Concepts	Composition Size	No of Solutions
1	118	1590	2	9
2	118	1590	3	8
3	118	1590	4	81
4	481	15541	4	81
5	481	15541	3	125
6	481	15541	2	100
7	481	15541	4	64
8	481	15541	3	30
9	481	15541	2	48
10	1000	56210	5	3125
11	1000	56210	12	500000
12	2000	58254	15	160000
13	4000	10891	8	6561
14	4000	10891	4	81
15	4000	58254	30	78125
16	8000	58254	40	177147
17	10000	58254	10	337500

Table 7: Experimental Test Sets.

accommodate quality values for each service presented in each test set. The set of criteria used in these experiments is composed of those defined in [Section 3], that is, Duration, Price, Availability, Successful Execution Rate and Reputation, with values ranging from 20 to 500, 0.10 to 1.75, 0.3 to 1.0, 0.3 to 1.0 and 0.3 to 1.0, respectively, randomly generated, following a uniform distribution.

5.2 Experimental Setup

The experiments are intended to evaluate the impacts of the insertion of QoS in the algorithm proposed by [Weise et al.(2008)]. The impacts can be negative, for instance the overhead in the processing time, or positive, such as on the improvement of the overall quality of the compositions. The experiments were performed by executing the original algorithm, i.e. the algorithm not considering the QoS criteria, and the algorithm considering QoS criteria under each request of test sets of [Tab. 7]. Primarily, several executions for each request were performed in order to measure the mean processing time of both versions of the algorithm. Then, the overall criteria values of each composition found were computed and contrasted in order to determine whether or not a composition is better than the other in terms of a specific QoS criterion.

In addition, as the WSC provides the possible solution for each request, it is possible to have a QoS-based ranking of these possible solutions by exhaustively searching the solutions with highest QoS score (Equation 7). This allows the comparison of compositions found by the algorithm considering, the algorithm not considering QoS and the optimal solution for a certain request.

All requests were performed using the following weight distribution: Duration: 0.3; Price: 0.3; Availability: 0.1; Reputation: 0.2; Successful Execution Rate: 0.1. The restriction imposed by the user on each quality criteria was omitted in these experiments, since it is just simple control checking performed after a composition is found. As previously stated, these experiments aim at evaluating the proposed approach in terms of feasibility and improvement on the QoS of the compositions.

5.3 Results

This subsection presents the results from the execution of the experiments mentioned in the last subsection. These results are expressed in several charts, where the first one expresses the mean processing time, in *milliseconds* (ms), of the algorithm considering and not considering the QoS criteria for each request. The second one expresses the overall score of the compositions found by the two algorithms and the optimal solution for each request. The other charts are plotted per criteria, showing the contrast between the two algorithms. Due to the lack of space, only the results of Overall Duration and Overall Price are described. To have further information about the results see [de Oliveira Jr.(2009)].

5.3.1 Mean Processing Time

[Fig. 3] shows the mean processing time of the execution under each one of the 17 requests (50 executions per request) for the algorithm with and without QoS concerns. In practice, both algorithms have the same processing time, which indicates that even large compositions in huge repositories with huge ontologies, the use of QoS criteria does not affect the performance of the original algorithm. From request 1 to 10, the processing times was less than 20ms. It starts increasing as the composition and repository sizes increase. Nevertheless, the maximum mean processing time was not greater than 450ms, which can be considered acceptable.

Although it is not presented in the WSC test sets and therefore not shown in [Fig. 3], it is important to discuss the behavior of the algorithm when there is no solution for a given request. The algorithm execution time will depend of course on the depth of candidate compositions, since all of them have to be analysed and discarded. As said before, in this case, the behavior will be very similar to a Depth-first search algorithm.

With respect to request 13, despite the differences of the execution time between the two approaches, so far there is no easy explanation for that. Things like this deserve more investigations.

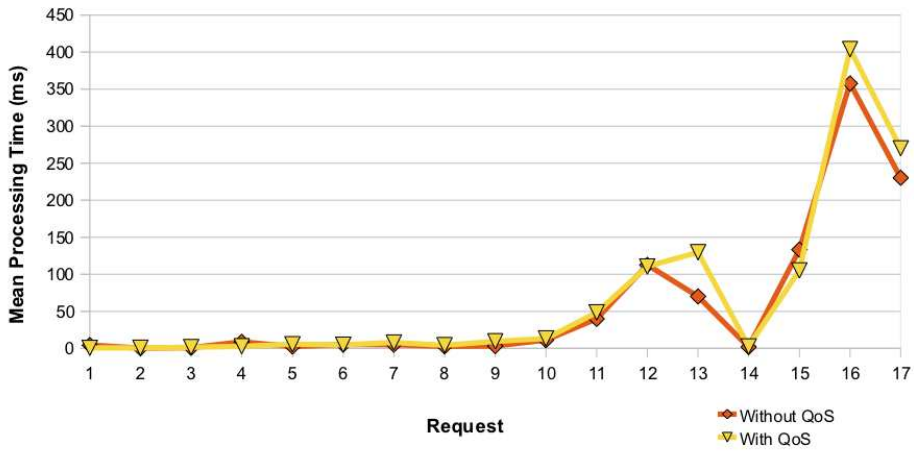

Figure 3: Mean Processing Time.

5.3.2 Overall Quality of Service Score

[Fig. 4] corresponds to a chart that compares the overall score (Equation 7) of the algorithm considering QoS, not considering QoS, and the optimal solution (with respect to QoS) for each given request. It can be seen that the algorithm considering QoS is always nearer to the optimal solution than the algorithm not considering QoS. In average, the overall score of the compositions found by the algorithm considering QoS is 95% (standard deviation 0.07) of that of the optimal solution, against 65% (standard deviation 0.16) from those found by the algorithm not considering QoS.

5.3.3 Overall Quality of Service per Criterion

This section presents the results obtained from each one of the 17 requests with the algorithm considering the QoS criteria, the algorithm not considering the QoS criteria and the optimal solutions focusing on each criterion individually.

[Fig. 5] shows the Overall Duration of the composition found by both algorithms as well as the Overall Duration of the optimal solution. In 12 requests

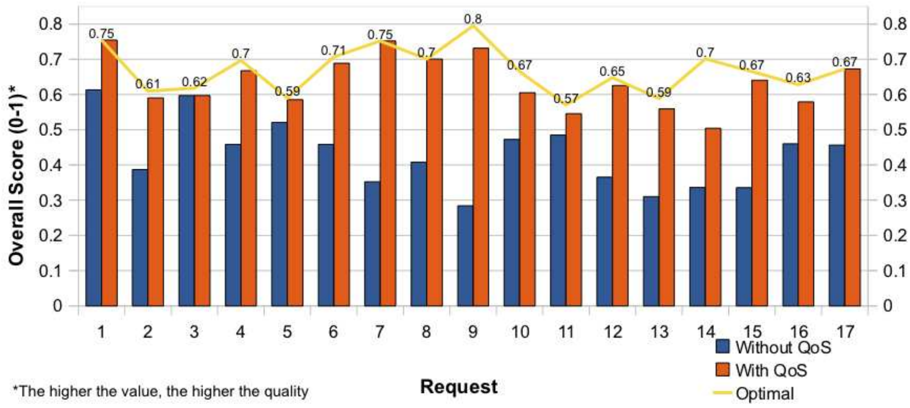

Figure 4: Overall Composition Score.

(1, 2, 4, 6, 7, 8, 10, 13, 14, 15, 16, 17), the Overall Duration of the compositions found by the algorithm considering QoS was nearer to the corresponding value of the optimal solution than the Overall Duration of the compositions found by the algorithm not considering QoS. In 4 requests (5, 9, 11, 12), the Overall Duration of the compositions found by the algorithm not considering QoS was nearer to that of the optimal solution than the corresponding value of compositions provided by the other algorithm. In only one case (request 3) the Overall Duration of compositions found by both algorithms was equally near to that of the optimal solution.

According to results presented in [Fig. 6], in 13 requests (1, 2, 4, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17), the Overall Price of the compositions found by the algorithm considering QoS was nearer to the corresponding value of the optimal solution than the Overall Price of the compositions found by the algorithm not considering QoS. In 3 requests (5, 6, 15), the Overall Price of the compositions found by the algorithm not considering QoS was nearer to the corresponding value of the optimal solution than that of compositions provided by the other algorithm. Only in one case (request 3), the Overall Price of the compositions found by both algorithms was equally near to the corresponding value of the optimal solution.

One important point to be considered is the fact that it is hard to draw conclusions on the quality provided by each criterion isolated, since a low value of one criteria may be compensated by a high value of another one. Thus, the results shown in this section do not have the objective to compare the quality criteria provided by two found compositions. Instead, these results are useful to illustrate

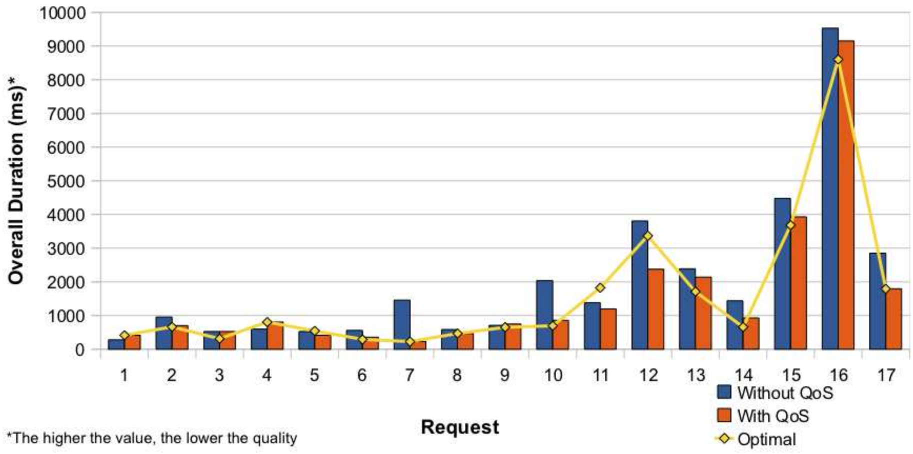

Figure 5: Overall Duration.

the approximation to the optimal solution for each criterion in the light of the weights assigned to them. The chart in [Fig. 4], on the other hand, provides a more clear idea about the overall QoS provided by a certain composition because it takes into account all criteria together as well as the weights assigned to each one of them.

5.4 Discussion

The results presented in the last section show that the proposed approach can dynamically find compositions that besides being correct in terms of functionality, also provide acceptable values of non-functional attributes that encompass their quality. In the matter of the approach feasibility, the experiments have shown that even in extreme situations such as in scenarios with huge service repositories, ontologies with a large number of concepts and large compositions, the proposed approach was able to find either the best solution, or a solution that is near to the best one.

Considering the approaches mentioned in [Section 2], some of them [Sirin et al.(2004), Oh et al.(2007), Weise et al.(2008), Yan et al.(2008)] do not consider QoS during composition. [Aversano et al.(2004)] considers QoS while performing dynamic composition, but does not make use of efficient strategy to make their approach feasible for large repositories. Therefore, the main advantage of the approach proposed in this paper in comparison with the others is the possibility

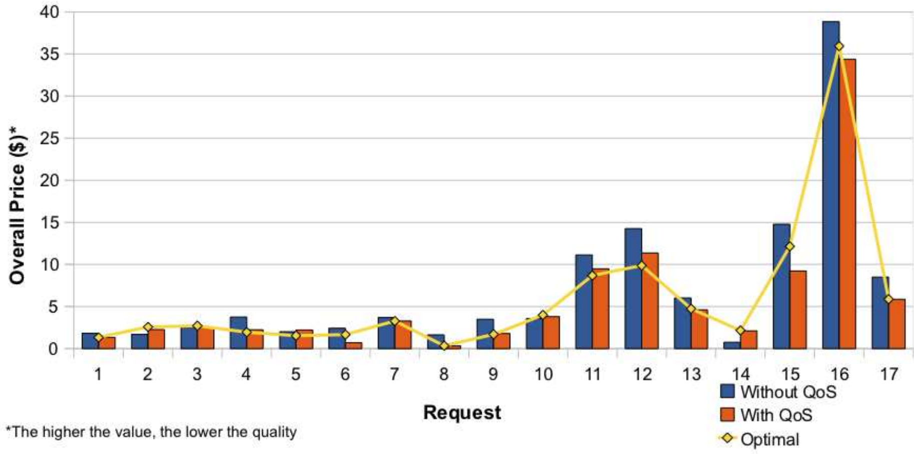

Figure 6: Overall Price.

to efficiently and dynamically find a composition with a certain level of quality, even if it is not the optimal one.

Although the algorithm proposed by [Weise et al.(2008)] and consequently its QoS version proposed in this work are not complete, i.e. even if there is a solution, the algorithm might not find one, and have time and memory complexity $O(b^m)$, where b is the maximum number of services that can produce any concept required by the candidate composition, and m is the maximum number of services in a composition, the executed experiments have shown its practical feasibility, even when executed in large repositories and ontologies. Moreover, the algorithm was able to find solutions for each one of the requests in reasonable time.

By contrasting the compositions resulting from the proposed approach to those resulting from the original algorithm, it is possible to perceive the substantial improvement in the overall quality of those compositions and the importance of taking into account the QoS criteria in the heuristics, since the heuristics is the main responsible for sorting the candidate compositions according to their functionality and quality.

As mentioned before, the user constraint on QoS criteria was not considered in these experiments, since it is just a simple checking on overall values. Suppose that, for request 8 of [Tab. 7], a user establishes a maximum Duration of 600, maximum Price of 0.75, minimum Availability of 0.2, minimum Successful Execution Rate of 0.3 and minimum reputation of 0.5. The composition found for this request by the proposed approach has Overall Duration of 467, Overall Price 0.35, Overall Availability 0.28, Overall Successful Execution Rate 0.32 and Overall Reputation 0.71. Hence, the found composition meets the QoS constraints imposed by the user. If the composition had other values so that it was not possible for it to meet the QoS constraints, it would be discarded and the algorithm would continue to try to find another composition.

6 Conclusion

Over the last years, web services have been used as an important tool for integration of distributed systems over the Internet. Recently introduced, the semantic web aims at adding meaning to current web and services published in it so that computers can be able to interpret information provided by them. As a consequence, machines can automatically find and compose more than one service in order to achieve a given goal. However, only the correctness of the composition goal might not be enough to fulfill the user requirements, and other constraints such as maximum cost or minimum execution duration should be taken into account. In other words, the composition should guarantee a minimum level of quality.

In this context, this paper presented an efficient algorithm for dynamic web service composition that considers a set of Quality of Service (QoS) criteria constraints imposed by the user. The proposed approach extends the one proposed by [Weise et al.(2008)] by considering QoS aspects of the composition.

In order to show the feasibility of the proposed approach, experiments were performed in a public benchmark and a significant improvement in the overall quality of the found compositions was noticed. This section shows the main contributions proposed by this work as well as the future work.

6.1 Contributions

The proposed approach relies on a heuristics function in order to guide the composition process. In practice, this heuristics tries to estimate how far a candidate composition is from the solution. The approach presented in this paper added to this heuristics a value that represents all the quality criteria involved in the composition construction process.

In addition, based on a modified version of the QoS models proposed by [Weise et al.(2008)], the algorithm for dynamic web service composition ignores the candidate compositions that no longer meet the QoS constraints imposed by the requester who, in turn, can assign different weights to criteria as means of expressing preference on a particular set of criteria.

In order to validate the proposed approach, a composer prototype that takes into account QoS was created. The prototype was modeled so that it is possible to accommodate several composition strategies and QoS criteria.

6.2 Future Work

Although the proposed approach has advanced in some aspects regarding the dynamic composition of semantic web services, some other important problems were not addressed due to the lack of time or to the limitation inherent to the adopted strategy. The overcoming of these limitations opens great opportunities for future research.

Parallel composition is one of the problems that were not addressed in this work. In fact, it is a strong limitation of the proposed approach, because Parallelization of Services has an impact on the overall quality, more specifically, in the Execution Duration of the composition. According to the models proposed by [Zeng et al.(2004)], other criteria like Availability and Successful Execution Rate could also be improved in compositions with parallel service execution. Therefore, it is really important to address this problem.

The fact that the proposed approach is neither complete nor optimal is indeed a limitation. However, as the experiments have shown, the heuristics was efficient either in finding a solution when it exists or in approximating to the optimal solution. Other techniques have to be applied in order to provide the optimal solution. It would be interesting to assess the tradeoff of having, in most of cases, a good approximation to the optimal solution versus the optimal solution in more time.

Other programming techniques could not be further analyzed in this work due to the lack of time. Techniques such as Dynamic Programming [Cormen et al.(1990)], where a complex problem is solved based on previous calculation of subproblems, indicates a promising direction toward the problem of the optimal dynamic web service composition taking into account the QoS aspects.

Despite the WSC provides a good benchmark for evaluating the scalability of service composition approaches, we are aware that for generality reasons it will be necessary to evaluate our approach in other benchmarks or rather in real world repositories and requests. It is also our intent to experimentaly evaluate and compare our solution with other approaches like [Aversano et al.(2004)] and [Yan et al.(2008)]. In addition, we should also investigate the overhead incurred by the composition algorithm in our approach.

Finally, although this work focused on web service composition, the proposed approach has a great potential to be applied in other other procedural abstractions. So, the use of our approach in other kinds of component-based application must also be investigated. In a near future after aditional experiments, we intend to make the algorithm available for download.

Acknowledgement

Thanks are due to CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) for funding this work.

References

[Aggarwal et al.(2004)] Aggarwal, Rohit, Verma, Kunal, Miller, John, Milnor, Willie, 2004. "Dynamic web service composition in meteor-s. Technical report, LSDIS Lab, Computer Science Deptartment", University of Georgia.

[Akkiraju(2007)] Akkiraju, R.: "Semantic web services"; K. Klinger, K. Roth, J. Neidig, S. Reed, S. Berger, J. LeBlanc, eds., Semantic Web Services: Theory, Tools and Applications; 191–216; IGI Global, London, UK/Hershey, PA, USA, 2007.

[Akkiraju et al.(2003)] Akkiraju, R., Goodwin, R., Doshi, P., Roeder, S.: "A method for semantically enhancing the service discovery capabilities of uddi"; S. Kambhampati, C. A. Knoblock, eds., IIWeb; Acapulco, Mexico, 2003.

[Aversano et al.(2004)] Aversano, L., Canfora, G., Ciampi, A.: "An algorithm for web service discovery through their composition"; Web Services, IEEE International Conference on; 0 (2004), 332.

[Blake et al.(2007)] Blake, M. B., Cheung, W. K., Jaeger, M. C., Wombacher, A.: "Wsc-07: Evolving the web services challenge"; E-Commerce Technology, IEEE International Conference on, and Enterprise Computing, E-Commerce, and E-Services, IEEE International Conference on; 0 (2007), 505–508.

[Box et al.(2000)] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., Winer, D.: Simple Object Access Protocol (SOAP) 1.1 (2000).

[BPEL (2007)] Web Service Business Process Execution Language (WSBPEL). Available from: `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`. Last visited in may 24th 2010.

[Canfora et al.(2005)] Canfora, G., Penta, M. D., Esposito, R., Villani, M. L.: "Qos-aware replanning of composite web services"; Web Services, IEEE International Conference on; 0 (2005), 121–129.

[Cardoso(2002)] Cardoso, J.: Quality of Service and Semantic Composition of Workflows; Ph.D. thesis; University of Georgia, Athens, GA (2002).

[Casati and Shan(2001)] Casati, F., Shan, M.-C.: "Dynamic and adaptive composition of e-services"; Information Systems; 26 (2001), 3, 143–163.

[Christensen et al.(2007)] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: "Web services description language (wsdl) 1.1"; W3C note; W3C (2007).

[Claro et al.(2008)] Claro, D. B., Licchelli, O., Albers, P., de Araújo Macêdo, R. J.: "Personalized reliable web service compositions"; F. L. G. de Freitas, H. Stuckenschmidt, H. S. Pinto, A. Malucelli, Ó. Corcho, eds., WONTO; volume 427 of CEUR Workshop Proceedings; CEUR-WS.org, Salvador-Bahia, Brazil, 2008.

[Cormen et al.(1990)] Cormen, T. H., Leiserson, C. E., Rivest, R. L.: Introduction to algorithms; MIT Press and McGraw-Hill, Cambridge, MA, USA, 1990.

[de Oliveira Jr.(2009)] de Oliveira Jr., F. G. A.: A QoS-Based Approach for Dynamic Web Service Composition; Master's thesis; Instituto Tecnológico de Aeronáutica, São José dos Campos, Brasil (2009).

[de Oliveira Jr. and de Oliveira(2009)] de Oliveira Jr., F. G. A., de Oliveira, J. M. P.: "A heuristic-based runtime ranking for service composition"; ICITST-2009: International Conference for Internet Technology and Secured Transactions; IEEE Computer Society, London, UK, 2009.

[Erl(2007)] Erl, T.: SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl); Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

[Fellbaum(1998)] Fellbaum, C., ed.: WordNet An Electronic Lexical Database; The MIT Press, Cambridge, MA ; London, 1998.

[Konar(2000)] Konar, A.: Artificial intelligence and soft computing: behavioral and cognitive modeling of the human brain; CRC Press, Inc., Boca Raton, FL, USA, 2000.

[Marques et al.(2010)] Marques, Henrique C.; Oliveira, Jos M. P.; and Costa, Paulo C. G.: "C2 framework for interoperability among an air component command and multi-agencies systems"; Accepted to the Fifteenth International Command and Control Research and Technology Symposium (ICCRTS 2010). June 22-24, 2010, Santa Monica, CA, USA.

[Mcilraith et al.(2001)] Mcilraith, S. A., Son, T. C., Zeng, H.: "Semantic web services"; Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications]; 16 (2001), 2, 46–53.

[OASIS(2004)] OASIS: "Universal description, discovery, and integration (UDDI) version 3.0.2"; (2004).

[Oh et al.(2007)] Oh, S.-C., Yoo, J.-W., Kil, H., Lee, D., Kumara, S. R. T.: "Semantic web-service discovery and composition using flexible parameter matching"; E-Commerce Technology, IEEE International Conference on, and Enterprise Computing, E-Commerce, and E-Services, IEEE International Conference on; 0 (2007), 533–542.

[Pistore et al.(2005)] Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: "Automated Synthesis of Composite BPEL4WS Web Services"; ICWS '05: Proceedings of the IEEE International Conference on Web Services; 293–301. IEEE Computer Society. Washington, DC, USA, 2005.

[Russell and Norvig(2003)] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach; Prentice-Hall, Englewood Cliffs, NJ, USA, 2003; 2nd edition edition.

[SC-Flood (2008)] "2008 Santa Catarina Floods in Wikipedia". Available from: `http://en.wikipedia.org/wiki/2008_Santa_Catarina_floods`. Last visited in may 24th 2010.

[Sirin et al.(2004)] Sirin, E., Parsia, B., Hendler, J.: "Filtering and selecting semantic web services with interactive composition techniques"; IEEE Intelligent Systems; 19 (2004), 4, 42–49.

[Tsetsos et al.(2007)] Tsetsos, V., Anagnostopoulos, C., Hadjiefthymiades, S.: "Semantic web services"; K. Klinger, K. Roth, J. Neidig, S. Reed, S. Berger, J. LeBlanc, eds., Semantic Web Services: Theory, Tools and Applications; 191–216; IGI Global, London, UK/Hershey, PA, USA, 2007.

[Weise et al.(2008)] Weise, T., Bleul, S., Comes, D., Geihs, K.: "Different approaches to semantic web service composition"; ICIW '08: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services; 90–96; IEEE Computer Society, Washington, DC, USA, 2008.

[Yan et al.(2008)] Yan, Y., Xu, B., Gu, Z.: "Automatic service composition using and/or graph"; E-Commerce Technology and Enterprise Computing, E-Commerce and E-Services, IEEE Conference and Fifth IEEE Conference; 0 (2008), 335–338.

[Yoon et al.(1995)] Yoon, P. K., Hwang, C.-L., Yoon, K.: Multiple Attribute Decision Making: An Introduction (Quantitative Applications in the Social Sciences); Sage Publications Inc, Thousand Oaks, CA, USA, 1995.

[Zeng et al.(2004)] Zeng, L., Benatallah, B., Ngu, A. H., Dumas, M., Kalagnanam, J., Chang, H.: "Qos-aware middleware for web services composition"; IEEE Transactions on Software Engineering; 30 (2004), 5, 311–327.