A User Requirements Oriented Semantic Web Services Composition Framework

Sana Baccar
CES Research Unit
National school of Engineers of Sfax
Sfax, Tunisia
sana.baccar@ceslab.org

Mohsen Rouached
Computer Science College
Taif University
Taif, Saudi Arabia
m.rouached@tu.edu.sa

Mohamed Abid
CES Research Unit
National school of Engineers of Sfax
Sfax, Tunisia
mohamed.abid_ces@yahoo.fr

***Abstract*—Web services provide an instantiation of the loosely coupled Service Oriented Architecture (SOA) and facilitate the process of enterprise application integration. However, with the explosive growth of the number of Web services published over the Internet, identifying a high quality of composite services by taking into account both functional and non-functional requirements of end users has become a real challenge that needs to be addressed. We propose in this paper an approach that adresses this challenge by considering a two-phase composition process. The composition first proceeds to generate an abstract plan based on Web service types using Dynamic Description Logics (DDL). This abstract plan is then concretized into an executable plan by selecting the appropriate Web service instances based on non-functional requirements.**

***Keywords*-Web service composition; Matching; ontology; Dynamic Description Logic; planning; QoS**

I. Introduction

Web services are becoming the prominent paradigm for distributed computing and electronic business. They have received much interest in industry due to their potential in facilitating seamless business-to-business or enterprise application integration. One of the key benefits of utilizing services is the potential for automatically formulating compositions of services resulting in integrated software applications. Web service composition is the process of realizing the requirements of a new Web service using the existing component Web services. The specification of composite Web services must reflect their functional and non functional capabilities. The functional capability of a Web service describes its core functionality. The non-functional capabilities, on the other hand, help in characterizing the service further by capturing its optional features, such as QoS parameters.

With the rapid expansion of Web service related applications in fields such as e-business, e-government and e-health, there is a clear need for infrastructures and frameworks that can be used to develop applications on the basis of Web service compositions. However, the explosive growth of the number of services published over the Internet makes the identification of high quality composite services by taking into account both functional a non-functional requirements a real challenging task that should be addressed.

In this context, this paper presents a two-step Web services composition framework that provides support for semantic matching while composing services, employs efficient decoupling of functional and non-functional requirements, and leads to improved discovery, selection, and composition processes. First, this framework differentiates between Web service types, which are groupings of similar (in terms of functionality) Web services, and the actual Web service instances that can be invoked. This separation will allows us to work efficiently with large collection of Web services. Then, the composition process starts by generating an abstract plan based on Web service types (abstract composition), which is subsequently concretized into an executable plan by selecting the appropriate Web service instances (concrete composition). Composition at the abstract level is reduced to formulas satisfiability checking in Dynamic Description Logic (DDL) language, extension of the description logics (DLs) with a dynamic dimension. By embracing knowledge in actions into DLs, DDLs couple the static information provided by ontologies and the dynamic processing provided by Web services, and offer a uniform way to represent and reason about both static and dynamic aspects of the Web content. Our approach employed classical DL-TBoxes to capture the constraints of the domain, DL-ABoxes to describe the states, and DL-formulas to encode the client requirements respectively. Actions in DDLs were used to abstract the functionalities of the existing Web services [1].

Concerning the composition at the concrete level, in order to turn the abstract plan into an executable workflow that can be deployed and executed, specific instances must be chosen for the component service types in the plan. This level uses optimization techniques in selecting the best Web service instances to produce an executable workflow. The focus is now on quantitatively exploring the available Web service instances for workflow execution. It queries the registry for deployed Web service instances and performs end to end QoS optimization to accomplish this task. The generated executable workflow must then be deployed onto a runtime infrastructure, and executed in an efficient and scalable manner. This latter task is beyond the scope of this paper and is outlined as future work.

The remainder of the paper is structured as follows.

Section 2 reviews and discusses the related existing efforts. In section 3, we introduce the main ingredients of our approach and discuss its main features. Section 4 details the composition process at abstract and concrete levels. Finally, Section 5 concludes the paper and outlines the future work.

II. Related Work

Recently, research on automatic Web services discovery and composition has been receiving a lot of attention. In what follows, we briefly review and discuss the existing efforts that are more closed to our contribution.

Dynamic description logic is used in [2], [3], [4], [5] to model and reason about Web services and their dynamic behaviors. In [2], service realizability and executability are handeled by checking formulas satisfiability and consistency through a simple DDL-based service model. [3] uses an imperative algorithm to enumerate all composition sequences. [4] is a DDL-based Web service composition approach that consists in generating a partial order diagram and running the composition process using a fast algorithm according to the generated diagram. In [5], the authors proposed an extension of the Arithmetic and Logic Class (ALC), called Service Dynamic Description Logic (SDDL), to support the discovery and composition of Web services. [6] proposes a new DDL-based composition model, which supports context-based service using bottom-up filtering approach. This contribution increases the DDL-reasoning efficiency by providing context-aware support and reduces the reasoning space. But it is limited by the fact that it is based on abstract reasoning without detailing the orchestration process and the action-based reasoning. Also none of these approaches considers non-functional characteristics or QoS attributes.

Some other approaches tried to simplfy the composition process by analyzing and reformulating the user-requirements into mathematical or formal expressions through different languages. For example, authors in [7] and [8] studied the problem of rewriting queries using views to speed up the query computation in an optimized manner. Some algorithms [9], [10] have been developed to enable a query referring to the view relations by reformulating the user-query based on virtual schema into a query that directly refers to the existing data sources. These algorithms have some limits when it comes to maximally query containment and when the comparison predicates still be included in the queries or the views.

In [11], an extension of the semantic Web service composition algorithm described in [12] was proposed in order to consider non-functional properties such as response time, cost and availability. However, this algorithm performs an exhaustive search in the repository which becomes infeasible as the number of available service increases.

In [13], 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. These approaches consider a subsumption-based matching of services, but none of them considers non-functional characteristics or QoS attributes.

[14] presents an integrated approach for automated semantic Web service composition using AI planning techniques. An important advantage of this approach is that the composition process, as well as the discovery of the atomic services that take part in the composition, are significantly facilitated by the incorporation of semantic information. The implementation was performed through the development and integration of two software systems, namely PORSCE II and VLEPPO. However user preferences and QoS were not adressed in this approach and just outlined as future goal.

Lin et al. [15] described a way to augment the Web service composition process by using qualitative user preferences. Bellur et al. [16] suggested the techniques for augmenting existing matchmaking algorithms with preconditions and effects in the context of Web services. Lecue et al. [17] addressed the scalability issue by selecting compositions that satisfy a set of constraints rather than attempting to produce an optimal composition. The method in Shin et al. [18] considers the functional semantics of a Web service on the basis of domain ontology. However, they conducted experiments using only a small number of Web services. Bener et al. [19] presented an architecture for semantic matching of Web services on the basis of input and output descriptions of semantic Web services as well as precondition and effect matching.

[20] proposed an approach to trigger and perform composite service replanning during execution because the actual QoS values may deviate from the estimation. WSQosX [21] is a workflow engine which calculates an execution plan that maximizes the overall QoS by use of heuristics. [22] suggested an efficient QoS-oriented Web services composition algorithm that combines tabu search and simulated annealing meta search and [23] proposed a Qos-aware composite service binding approach based on genetic algorithms (GAs). Recently, LOEM [24] is proposed to reduce computation time for determining the optimal composite solution by a heuristic service composition method. QSynth [25] is proposed to address both scalability and accuracy based by using QoS objectives of service request. The problem with these approaches is that they are almost QoS local optimization or mono-objective based, and can not resolve the problem of Web services selection with QoS global optimization and multi-objective.

III. User Queries and Services Specification Using DDL

In this section, we start by briefly reviewing the main features of the Dynamic Description Language formalism. Then, we explain how these features can be used to specify and compose Web services.

A. Dynamic Description Language (DDL) overview

Description languages [26] are unsufficient to handle the dynamic knowledge such as services and behaviors. To overcome this shortfall, DDLs were proposed as extensions of description logics with a dynamic dimension. DDL is based on the idea that the occurrence and the development of actions can influence the world evolution from one state to another. DDL is referred to any one of the general family of logics with the ability of supporting dynamicity such as D_{ALC}, D_{SHOIQ} and $\mathrm{D}_{ALCO@}$. In this paper, we use D_{ALCO}. With this logic, description logic concepts can be used for describing the state of the world, and the preconditions and effects of atomic actions; Complex actions can be modeled with the help of standard action operators, such as the test, sequence, choice, and iteration operators; And both atomic actions and complex actions can be used as modal operators to construct formulas.

The primitive symbols of D_{ALCO} contain a set of concept names (N_C), a set of relation names (N_R), a set of individual names (N_I) and a set of atomic action names (N_A) included in construction operators named TBox. D_{ALCO} specifies actions by their preconditions and effects, together with their categories. The knowledge about action categories can be used to construct a hierarchy structure of actions.

Concepts in D_{ALCO} are defined as follows:
C,D$\rightarrow \{u\}|\neg C|C \cap D|\exists R.C| \leq nR.C| \geq nR.C| < \pi > D$
Where C, D $\in$ N_C, u $\in$ N_I , R $\in$ N_R, $\leq$ n R.C and $\geq$n R.C are qualifying number restriction and π is an action.
In the sequel, we use $\bot, \top$, C$\cup$ D), and $\forall$R.C to abbreviate (C$\cap\neg$C), $\neg\bot$, $\neg(\neg C \cap \neg$ D), and $\neg\exists$ R. $\neg$C, respectively.

Formulas in D_{ALCO} are built up with the following rules: $\Phi, \Psi \rightarrow C(u)|R(u,v)|\Phi \rightarrow \Psi|\neg\Phi|\forall u\Phi|\Phi \vee \Psi|[\pi]\Phi| < \pi > \Phi$, Where u, v$\in$ N_I , R$\in$ N_R and π. Formulas of the form C(u), R(u,v),$\Phi \rightarrow \Psi$,$\neg\Phi$,$\neg\Phi$, $\forall u\Phi$, $\Phi \vee \Psi$, $[\pi]\Phi$, $< \pi > \Phi$ are respectively named as concept assertion, role assertion, formula assignment, negation formula, universal formula, disjunction formula, conditional assertion and diamond assertion.

We still define the logical connectives "$\rightarrow$" and "$\leftrightarrow$" in terms of "$\neg$", "$\vee$", as usual, and define "$[\pi]\Phi$" as "$\neg < \pi > \neg\Phi$".

A ***Formula assignement*** is defined by σ where:
σ $\{\Phi_1,..,\Phi_n\}\rightarrow\{\Phi'_1,..,\Phi'_n\}$ and means that if Φ is a variable formula, Φ' is the instance of Φ with assignment σ.

An ***Action*** in D_{ALCO} is defined as the couple $< P, E >$ where P is a finite set of formulas specifying the condition for the execution of the atomic action and E is a finite set of formulas of the form $C(u)$ or $R(u,v)$, or their negations;

Complex actions can be constructed with classical action constructors in dynamic logics, and both atomic and complex actions are actions that are built up as follows. $\pi, \pi' \rightarrow \alpha|\Phi?|\pi \cup \pi'|\pi;\pi'|\pi^*$, where α, π, π' are atomic services, and Φ is a formula in ALCO.

Intuitively, an atomic action encodes the changes of the domain caused by the action through the specifications of the pre-conditions under which the action is applicable and how the action affects the states of the domain. Moreover, both atomic and complex actions can be used as modal operators to construct formulas. Such mechanism enables to make assertions about the properties of the accessible worlds from the current world. For example, we can state that "the action π can be executed with the formula Φ holds after its execution" by the formula: "$< \pi > \Phi$", while "$[\pi]\Phi$" encodes "after each execution of π", the formula Φ holds in the reached world".

The definitions of ***ABox*** and ***TBox*** in D_{ALCO} are the same as that in ALCO. TBoxes capture the domain constraints of an application domain. ABoxes (with no semantic conflicts) describe the worlds at certain times, and the transitions on the possible worlds are captured by actions.

An ***action box*** is a finite set $ActBox$ of atomic actions in D_{ALCO}.

In D_{ALCO}, a domain specification is defined as a tuple: DS = $< T, A, ActBox >$, where T is a TBox, consisting of domain constraints; A is an ABox for the initial world; ActBox is an action box that captures the dynamic knowledge about world evolvements. Table I gives the $TBox$, $ABox$, and $ActBox$ of an online shopping scenario.

Semantics of DDL are detailed in [27]. For space reasons, we just introduce the main reasoning strategies in what follows.

A model for D_{ALCO} is considered as a pair M= (I,W), where I =(Δ^I, $.^I$) is an ALCO-interpretation and W is a set of ALCO-interpretations, seen as possible worlds. I consists of a nonempty domain Δ^I and a mapping $.^I$ that assigns each atomic concept to a subset of Δ^I, an individual name to an element of Δ^I, and each role to a subset of $\Delta^I * \Delta^I$.

The satisfaction of a set of formulas F in (I,W), written as (I,W)$\models$ F, is defined in TableII:

1)(I,W)$\models$ C(p) iff $\mathrm{p}^I \in \mathrm{C}^{I(w)}$;
2)(I,W)$\models$ R(p,q) iff (p^I,q^I)$\in$ $\mathrm{R}^{I(W)}$;
3)(I,W)$\models \neg\Phi$ iff (I,W) $\not\models \Phi$;
4)(I,W)$\models \Phi \vee \Psi$ iff (I,W)$\models \Phi\vee$ (I,W)$\models \Psi$;
5)(I,W)$\models \Phi \wedge \Psi$ iff (I,W)$\models \Phi\wedge$ (I,W)$\models \Psi$;
6)(I,W)$\models \Phi \rightarrow \Psi$ iff (I,W)$\models \Phi \Rightarrow$ (I,W)$\models \Psi$;
7)(I,W)$\models < \pi > \Phi$ $\exists$w'$\in$ W|((w,w')$\in \pi^I$ and (I,w')$\models \Phi$);
8)(I,W)$\models [\pi]\Phi$ iff $\forall$v$\in$W (w$\rightarrow_\pi$v $\Rightarrow$ v$\models \Phi$.

Table II: The Satisfaction of formula F in (I,W)

An action is executable in a possible world w satisfying its preconditions stated in Pre, and its execution reaches a possible world $\acute{w}$ satisfying the effects stated in Eff, with a minimal change w.r.t the previous world.

TBox	{OnlineShopping≡GoodsToPublishOnline∩Payment∩ Shipping,OnlineShopping∪GoodsToPublishOnline∪ Payment∪Shipping⊆Service, OnlineShopping⊆Shopping, ShippingDate⊆Date,Established∪Pay⊆PaymentState, customer-name∪goods-name⊆name, OrdReq∪PaymentMsg⊆Msg, PC∪SmartPhone⊆Aplliance, CreditCardInf⊆PrivateData, ConnectionState∪GoodsState∪ PaymentState⊆State, customer∪seller⊆Person}
ABox	{Web-user (customer), Msg(OrdReq),Person(seller), SendTo(Msg,Person), Site(WebSite),name(customer-name), name(goods-name),address (customer-address),PostCode(customer-PostCode),CreditCard(credit-card),ConnectionState(established), Msg(PaymentMsg), Dtae(ShippingDate)}
ActBox	{CheckGoodsStock (S_1), PaymentByCreditCard (S_2), PaymentByPayPal (S_3), Shipping (S_4), Delivery (S_5)}.

Table I: ABox, TBox, and ActBox of an online shopping process

B. Specification and Rewriting of User Queries

In our approach, a user query Q is modeled as a four-uplet $< I, O, A, QoS >$, where :

- I is a finite set of inputs of the query;
- O is a finite set of outputs of the query;
- A is a finite set of desired actions. Each action consists of a finite set of pre-conditions P and effects or post-conditions E.
- QoS is a set of quality of service constraints such that $QoS = \{(q_1, v_1, w_1), (q_2, v_2, w_2), ..., (q_k, v_k, w_k)\}$, where $q_i (i = 1, 2, ..., k)$ 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 query.

Given a query Q and a set of Web services, if the query Q is complex and there is no Web service that answers Q, we try to find a decomposition of Q into a set Q' of atomic sub-queries that can be answered by the existing services. The atomic services that respond to the sub-queries will be then collected and composed to provide a final answer for the user goal (original query).

Formally, given a query Q a rewriting of Q is Q' such that $Q = Q'$. So that, each element in Q' refers to one or more service implementations. The rewriting process is completed if each sub-query in Q' refers to an atomic service.

Two relationships can be identified for the rewriting process:

- *Query containment*: A query Q' is contained in another query Q, denoted as $Q' \subseteq Q$, if each instance of Q' is a subset or equivalent to that of Q.
- *Query equivalence*: Two queries Q and Q' are equivalent (denoted as $Q = Q'$) if $Q \subseteq Q'$ and $Q \supseteq Q'$.

Using the DDL formalism, the set of query-inputs is considered as the initial state of the world, from which a service sequence is selected and a new state will be added to the world. If a sequence occurs and the target state set of the world that includes user requirements is achieved, this sequence in considered as the composed service that meets the user needs.

If we consider two states $u, v \in W$, where u designs the initial state formed over query inputs and v describes the final state after the query-rewriting process, each interpretation I of the query Q started from the state u, should reach the user goal in the state v through Q' using formula (1).

$$Q^{I,u} = \{(Q' \cup SQ_i)^{I,v}, 1 \leq i \leq n\} \quad (1)$$

Where: $SQ_i^{I,u} = (P_i, E_i)^{I,u} = \{(u, v) | \forall \Phi_i \in P_i(I, u) \models \Phi_i$, and $(I, v) = (I, u) \cup \{\Psi_i\} \forall \Psi_i \in E_i\}$;

This mechanism will be repeated until reaching the desired outputs and QoS: $(I, v) \models O \cap E \cap QoS$.

Table III shows a query Q for an online shopping transaction. This query Q is complex and no existing service in the $ActionBox$ can achieve the desired goal. Thus, Q can be rewritten as a conjunction set Q' of atomic sub-queries such as $Q' = CheckGoodsStockOnline \cap Payment \cap Delivery$ for example.

C. Service Specification

An atomic service is a tuple $S =< I, O, P, E, qos >$, where I, O are the input and output of the service S, respectively; P is a finite set of formulas in D_{ALCO} specifying the preconditions for the execution of S; E is a finite set of assertions or their negation in D_{ALCO}, which is the facts holding in the newly-reached world by the services execution, and qos is a set of provided quality criteria. The formulas in both P and E are conferred with well-defined semantics encoded in some $TBox$, which specifies the domain constraints in consideration. Table III gives an overview of the set of services that may be involved in the online shopping transaction.

Composite services are constructed from atomic services with the help of classic constructors in dynamic logics. Both atomic and composite services are services. Complex services are built up with formula (2).

$$\pi, \pi' \rightarrow \alpha | \Phi? | \pi \cup \pi' | \pi; \pi' | \pi^* | ((\Phi; \pi) \cup ((\neg\Phi)?; \pi')) | (\Phi?; \pi)^*; (\neg\Phi?) \quad (2)$$

where α, π, π' are atomic services, and Φ is a formula. Based on formulas and atomic services, complex services can be constructed with the help of four standard atomic services operators: the test $(\Phi?)$, sequence $(\pi; \pi')$, choice $(\pi \cup \pi')$, iteration operators (π^*), if formula Φ is verified then π is invoked, else invoking π' is an abbreviation of $(((\Phi; \pi) \cup ((\neg\Phi)?; \pi'))$, and if Φ do π is an abbreviation of $(\Phi?; \pi)^*; (\neg\Phi?)$.

IV. Web Services Composition

In our approach, We distinguish between Web service types and services instances. A *Web Service Type* is a set

Service-name	*Inputs*	*Outputs*	*Action*		*QoS*
			Preconditions	*Effects*	
Q	customer-name, customer-address, customer-PostCode, CreditCardInf	PaymentState(Pay), ShippingDate≤3	Electronic, OrdReq, SendTo (OrdReq, seller)	credit-card payment, shipping-date, goods costs	(data-privacy,"true", 0.4),(ShippingDate,≤3 days,0.2),(goods-cost,acceptable,0.3), (PaymentCost,minimal,0.1).
S_1	goods name, goods characteristics, goods cost	CheckStock(Goods)	appliance, ConnectionState (established), GoodsList	GoodsAvailableInStock	(goods-cost,"acceptable")
S_2	OrdReq, CreditCardInf	PaymentMsg	CreditCardState(valid), CreditCardSold≥ GoodsCost	PaymentState (EstablishedByCreditCard)	(privacy, "False"),(PaymentCost, 10$)
S_3	OrdReq	PaymentMsg	Sold≥ GoodsCost	PaymentState (EstablishedByPayPal)	(privacy, "True") ,(PaymentCost, 5$)
S_4	OrdReq, address, PostCode	ShippingDate	PaymentMsg (Ok)	GoodsState(delivered)	(DeliveryDate≤2 days), (ShippingCost=free)
S_5	OrdReq, address	DeliveryDate	PaymentMsg (Ok)	GoodsState(delivered)	(DeliveryDate≥4 days), (DeliveryCost=4$)

Table III: Sample set of DDL-based re-written services

of Web Service instances with similar functionality. A Web service type is semantically described by the inputs, outputs, preconditions and effects that capture the functionality offered by this type of services.

A *Web Service Instance* consists of its functionality, service type and its QoS properties. Web services instances are the actual services that can be invoked. A Web service instance is semantically described by the inputs, outputs, preconditions, effects, and the non-functional (QoS) attributes associated with this instance.

This separation between service types and instances have an important impact on the reduction of the search space when discovering candidate services for composition. Then, given a set of Web service types and the set of instances for each type, along with the specifications of a new service, a Web services composition is the process of creating an executable plan that stitches together the desired functionality from the existing services, while satisfying the QoS requirements. To enable this, we consider two composition stages, which are *Abstract* and *Concrete* compositions. The composition first proceeds to generate an abstract plan based on Web service types (Abstrat composition). This abstract plan is then concretized into an executable plan by selecting the appropriate Web service instances based on non-functional requirements (Concrete composition).

A. Abstract composition

The main reasoning tasks in DDL can be reduced to satisfiability checking of formulas. A formula ψ is satisfiable w.r.t a TBox T, if there is a dynamic interpretation that satisfies T and ψ. More precisely , an atomic action can be invoked only if formula (3) is satisfied.

$$I \wedge P \rightarrow O \wedge E \wedge QoS \quad (3)$$

Suppose that each user query and available service are designed by WS_R and WS_S respectively. The matching is given by the fact that the WS_R inputs, outputs, actions and QoS should be equivalent (e.g. they correspond to the same concept defined in the ontology and the near values of the QoS) or subsumed by that of the WS_S's concepts/values. So that, this matching process is satisfiable if and only in the cases where $WS_R \equiv WS_S$ or $WS_R \subseteq WS_S$. To deal with an optimal matching between the query and the atomic Web service concepts, the matching mechanism can be simply reduced to checking the unstasifiability of formula (4).

$$WS_R \cap \neg WS_S. \quad (4)$$

An abstract composite service plan P is composed of a set of service types and defined by $P = \{\pi_1, .., \pi_i\}$, where $1 \leq i \leq n$ and $\pi_i = AS_i$.

The determination of P consists in checking the satisfiability of the abstract model as indicated formula (5).

$$[(P_1 \cup ... \cup P_k)^*]\Pi \wedge S \rightarrow < Q > True \quad (5)$$

Where Π denotes the conjunction of the formula set of preconditions (P_i) of all service types ST_{ij} in the plan P_j, with $i \in N$ and $1 \leq j \leq k$ and $\Pi = \wedge_{i=1}^{n} Conj(p_i) \rightarrow < AS_{ij} > True$. S is a formula set that consists of a conjunction of member formulas S_i according to the $ABox$ that describes the initial state of the world: $S = Conj(S_i) \mid 1 \leq i \leq n$.

The validity of formula (6) requires the unsatisfiability of its negation.

$$F1 : (P_1 \cup ... \cup P_k)^*]\Pi \wedge Conj(s) \wedge \neg < Q > True \quad (6)$$

The dependency relationships between each two consecutive service types AS_1 and AS_2 in the plan can be simply checked by the unsatisfiability of formula (7), which means that invoking AS_2 sequentially after executing AS_1 requires that the inputs and preconditions of AS_2 have to be equivalent.

$$F : I2 \wedge P2 \wedge \neg(O1 \wedge E1) \quad (7)$$

Then, formula (8) is satisfiable only if whenever this sequence of services is executed; the formula F must be

unsatisfiable.

$F2 : [AS1; AS2]\neg F$ (8)

Finally, the abstract plan is given by algorithm 1:

Algorithm 1: Abstract Plan Modeling Algorithm

Input: User-query Q; set of service types $AS_i \in AS$
Output: Abstract service composition plan meeting user requirements **or** nil as failure;

```
begin
    Q'=QueryRewriting (Q); Plan=∅;
    for each sub-query (SQ_i)∈Q' do
        for each AS_i, AS_{i-1} ∈ AS do
            if SQ_i ⊆AS_i and AS_i ≠ AS_{i-1} and F is
            unsatisfiable then
                P= {P∪AS_i}
    Initialize queue QueOfPlans with P;
    while QueOfPlans is non-empty do
        for each P_k= headOf QueOfPlans do
            if F1 is unsatisfiable then
                if for each AS_i, AS_{i+1} ∈ P_k do
                    F2 is satisfiable
                then
                    return P_k as a successful plan;
                    headOf QueOfPlans= P_{k+1};
            remove P_k from QueOfPlans;
            headOf QueOfPlans= P_{k+1};
        return nil
End
```

B. Concrete Composition

In the concrete composition, the plan generated by the abstract composition stage is considered as a template for the composite service, which in conjunction with the QoS parameters specifications, drives the process of matching each service type to a corresponding service instance. Usually, for each service type in the plan, a set of alternative Web services instances with similar functionality is available, and that these Web services instances have different QoS parameters. This leads to the general optimization problem of how to select services instances for each type so that the overall QoS requirements of the composition are satisfied and optimal.

We consider four QoS properties of Web service. These propeties are:

1) Price $(q_{pr}(s))$ of a service s is the fee that its requesters have to pay for invoking it.
2) Duration $(q_{du}(s))$ of a service s measures the expected delay between the moment when a query is sent and the moment when the result is received.
3) Availability $(q_{av}(s))$ of a service s is the probability that it is accessible.
4) 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;

To generate the composition plan that fulfills a end user QoS requirement, we adopt the Simple Additive Weighting (SAW) approach in Multiple Criteria Decision Making (MCDM)[28]. This approach consists in two phases which are:

- Scaling phase: A QoS criteria could be either positive or negative. As shown in Table IV, some QoS values such as availability and reputation need to be maximized, whereas other values such as the execution time and costs have to be minimized. To cope with such issue, a scaling phase based on normalizing the value of each QoS criterion is introduced.
- Weighting phase: In this phase, a new parameter $w_k \in [0, 1]$ is associated to each normalized value of the QoS criterion. This parameter presents the user preference.

QoS criteria	Local constraints
$q_{pr}(s)$	$\min \{q_{pr}(CS_i)_{i\in\{1,..,n\}}\}$
$q_{du}(s)$	$\min \{q_{du}(CS_i)_{i\in\{1,..,n\}}\}$
$q_{rep}(s)$	$\max \{q_{rep}(CS_i)_{i\in\{1,..,n\}}\}$
$q_{av}(s)$	$\max \{q_{av}(CS_i)_{i\in\{1,..,n\}}\}$

Table IV: Local constraints formulas for QoS criteria

Each instance score is calculated using the formula: $Score(CS_i) = \sum_{k=1}^{r} (q'_{i,k} * w_k)$. Where $q'_{i,k}$ are normalized values of a decision matrix elements calculated as follows:

$$q'_{i,k} = \begin{cases} \frac{v_{i,k}}{V_k^{max}}, if(V_k^{max} - V_k^{min}) \neq 0 (a) \\ \frac{V_k^{min}}{v_{i,k}}, if(V_k^{max} - V_k^{min}) \neq 0 (b) \\ 1, \qquad if(V_k^{max} - V_k^{min}) = 0. \end{cases}$$

Where each Score(CS_i) denotes the normalized value of r QoS criteria (q_k) associated with candidate instance CS_i, $v_{i,k}$ is the current value while V_k^{max} and V_k^{min} denote respectively the maximum and minimum values of QoS criteria q_k among all the candidate instances. A matrix M is built, in which each row M_i corresponds to an instance CS_i while each column corresponds to a quality criteria q_k.

Once the matrix M was calculated, for each service type included in the abstract plan, the instance with high score is selected. Then, the selected instances are aggregated to determine the overall QoS of the composite service with the aggregation formulas presented in table V. We assume that QoS measures considered here are quantitative, the QoS values are static, e.g., the mean value of each QoS measure. Dynamic QoS is beyond the scope of this paper.

If there is no matching with the global constraints, we can select other instances near to the optimal instances. If no alternative instance is available, we automatically replace the plan P_i with a plan P_{i+1} such that:

Criterion	Function
Reputation	$A_{rep} = 1/n \sum_{i=1}^{n} q_{rep}(si)$
Price	$A_{pr} = 1/n \sum_{i=1}^{n} q_{pr}(si)$
Duration	$A_{du} = 1/n \sum_{i=1}^{n} q_{du}(si)$
Availability	$A_{av} = \prod_{i=1}^{n} q_{av}(si)$

Table V: Aggregation functions for computing composition QoS

$((\Psi; P_i) \cup ((\neg\Psi)?; P_{i+1})), 1 \leq i \leq K$

Let us now come back to our online shopping scenario to illustrate the selection process. We consider three services types and their corresponding instances as shown in table VI.

Service Type (AS)	Service Instance (CS)	q_{pr}	q_{du}	q_{rep}	q_{av}
AS_1	S_1	12	120	0.98	0.95
	S_2	15	70	0.5	0.4
	S_3	16	80	0.9	0.6
AS_2	S_4	10	100	0.88	0.91
	S_5	13	88	0.5	0.4
	S_6	15	110	0.9	0.6
AS_3	S_7	17	90	0.78	0.67
	S_8	16	50	0.7	0.6
	S_9	35	110	0.4	0.3

Table VI: QoS values for the online shopping scenario

We suppose also a query that contains the following inputs and user preferences:

- **Inputs:** OrdReq: iPad - 4G Frequencies 2100mhz; customer-name: Daniela John; customer-address: Roma, Italy; customer-PostCode:00198; CreditCardNb: 876545678; CreditCardPWD: 2344321.
- **Preferences:**(q_{pr},40,0.4), (q_{du},300, 0.3), (q_{rep},0.6,0.1), (q_{av},0.5, 0.2).

According to formula (7), the abstract plan of the goal service can be described as

$$P =< CheckGoodsAvailability, Payment, Delivery >$$

Service types $CheckGoodsAvailability$,$Payment$, and $Delivery$ are denoted by AS_1, AS_2, and AS_3 respectively. The results of the SAW selection technique are presented in table VII.

The instance with the highest score is taken among the set of instances of each service type. As shown in table VII, for the first service type $S_1 \prec S_3 \prec S_1$, for the second type $S_6 \prec S_5 \prec S_4$ and for the third type $S_9 \prec S_7 \prec S_8$. Thus, $P =< S_1, S_4, S_8 >$ is seleted as the optimal execution plan. Then, to verify the global score of the composite service resulting from P, we apply the aggregation functions described above and obtain $A_{pr} = 38$; $A_{du} = 270$; $A_{rep} = 0.6$; $A_{av} = (-0.738)$.

AS	CS	q'_{pr}	q'_{du}	q'_{rep}	q'_{av}	Score
AS_1	S_1	1	0.58	1	1	0.97
	S_2	0.8	1	0.51	0.42	0.75
	S_3	0.75	0.87	0.91	0.63	0.77
AS_2	S_4	1	0.88	0.97	1	0.95
	S_5	0.76	1	0.55	0.43	0.74
	S_6	0.66	0.8	1	0.65	0.73
AS_3	S_7	0.94	0.55	1	1	0.83
	S_8	1	1	0.89	0.89	0.96
	S_9	0.45	0.45	0.51	0.44	0.44

Table VII: Instance Score Calculation

V. Conclusion

In this paper, we have presented a semantic user requirements oriented framework to compose Web services. This framework distinguishes between abstract and concrete compositions. Composition at the abstract level is reduced to formulas satisfiability checking in DDL language to generate an abstract plan based on Web service types. This abstract plan is then concretized into an executable plan by selecting the appropriate Web service instances based on QoS constraints using the SAW approach in MCDM. Currently, we are working on the implementation of a composition planning architecture that contains the different components of the framework (planner, Selection engine...). Then, we plan to consider a real dataset such as the one presented in [29] and compare our approach with the RFC [29], and the two-phase [30] approaches.

Acknowledgment

The work reported in this paper is supported by King Abdul Aziz City for Sciences and Technology (KACST), Riyadh, Kingdom of Saudi Arabia, under grant number 11-INF1776-08.

References

[1] Limin Chen, Hong Hu, and Zhongzhi Shi. Web service composition as satisfiability checking in dynamic description logics. *Grid and Cloud Computing, International Conference on*, 0:55–60, 2009.

[2] Guohua Shen, Zhiqiu Huang, Xiaodong Zhu, and Jun Yang. Reasoning about web services with dynamic description logics. In Mark Burgin, Masud H. Chowdhury, Chan H. Ham, Simone A. Ludwig, Weilian Su, and Sumanth Yenduri, editors, *CSIE (6)*, pages 106–110. IEEE Computer Society, 2009.

[3] Ion Constantinescu, Boi Faltings, and Walter Binder. Large scale, type-compatible service composition. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '04, pages 506–, Washington, DC, USA, 2004. IEEE Computer Society.

[4] Wei Liu Yu Yue Du Bao Qi Guo Chun Yan Qiang Xu. A fast algorithm for web service composition based on dynamic description logic. In *Information Technology Journal, 9*, 1150-1157. Asian Network for Scientific Information, 2011.

[5] Zhixiong Jiang, Leqiu Qian, Xin Pen, and Shisheng Zhu. Dynamic description logic for describing semantic web services. In *Computer and Computational Sciences, 2007. IMSCCS 2007. Second International Multi-Symposiums on*, pages 212 –219, aug. 2007.

[6] Wenjia Niu, Zhongzhi Shi, Changlin Wan, Liang Chang, and Hui Peng. A ddl-based model for web service composition in context-aware environment. In *ICWS*, pages 787–788, 2008.

[7] Rachel Pottinger and Alon Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2001.

[8] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, December 2001.

[9] Michal Mrissa and Mohand-Said Hacid. Combining configuration and query rewriting for Web service composition. Technical Report RR-LIRIS-2009-045, LIRIS UMR 5205 CNRS/INSA de Lyon/Universit Claude Bernard Lyon 1/Universit Lumire Lyon 2/cole Centrale de Lyon, December 2009.

[10] Odysseas G. Tsatalos, Marvin H. Solomon, and Yannis E. Ioannidis. The gmap: a versatile tool for physical data independence. *The VLDB Journal*, 5(2):101–118, April 1996.

[11] Lerina Aversano, Gerardo Canfora, and Anna Ciampi. An algorithm for web service discovery through their composition. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '04, pages 332–, Washington, DC, USA, 2004. IEEE Computer Society.

[12] Rama Akkiraju, Richard Goodwin, Prashant Doshi, and Sascha Roeder. A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI. In Subbarao Kambhampati and Craig A. Knoblock, editors, *IIWeb*, 2003.

[13] Thomas Weise, Steffen Bleul, Diana Comes, and Kurt Geihs. Different approaches to semantic web service composition. In *Proceedings of The Third International Conference on Internet and Web Applications and Services, ICIW 2008*. IEEE, IEEE Computer Society Press, June 9 2008.

[14] Ourania Hatzi, Dimitris Vrakas, Mara Nikolaidou, Nick Bassiliades, Dimosthenis Anagnostopoulos, and Ioannis Vlahavas. An integrated approach to automated semantic web service composition through planning. *IEEE Transactions on Services Computing*, 5(3):319–332, 2012.

[15] Naiwen Lin, Ugur Kuter, and Evren Sirin. Web service composition with user preferences. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pages 629–643, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] U. Bellur and H. Vadodaria. On extending semantic matchmaking to include preconditions and effects. In *Web Services, 2008. ICWS '08. IEEE International Conference on*, pages 120–128, Sept.

[17] Freddy Lecue and Nikolay Mehandjiev. Towards scalability of quality driven semantic web service composition. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 469–476, Washington, DC, USA, 2009. IEEE Computer Society.

[18] Dong-Hoon Shin, Kyong-Ho Lee, and Tatsuya Suda. Automated generation of composite web services based on functional semantics. *Web Semant.*, 7(4):332–343, December 2009.

[19] Ayse B. Bener, Volkan Ozadali, and Erdem Savas Ilhan. Semantic matchmaker with precondition and effect matching using swrl. *Expert Syst. Appl.*, 36(5):9371–9377, July 2009.

[20] Vishal S. Batra and Nipun Batra. Improving web service qos for wireless pervasive devices. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '05, pages 130–137, Washington, DC, USA, 2005. IEEE Computer Society.

[21] Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, and Ralf Steinmetz. Heuristics for qos-aware web service composition. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '06, pages 72–82, Washington, DC, USA, 2006. IEEE Computer Society.

[22] Jong Myoung Ko, Chang Ouk Kim, and Ick-Hyun Kwon. Quality-of-service oriented web service composition algorithm and planning architecture. *J. Syst. Softw.*, 81(11):2079–2090, November 2008.

[23] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. A framework for qos-aware binding and re-binding of composite web services. *J. Syst. Softw.*, 81(10):1754–1769, October 2008.

[24] Lianyong Qi, Ying Tang, Wanchun Dou, and Jinjun Chen. Combining local optimization and enumeration for qos-aware web service composition. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 34–41, July.

[25] Wei Jiang, C. Zhang, Zhenqiu Huang, Mingwen Chen, Songlin Hu, and Zhiyong Liu. Qsynth: A tool for qos-aware automatic service composition. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 42–49, July.

[26] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. In *Mechanizing Mathematical Reasoning*, pages 228–248, 2005.

[27] Liang Chang, Fen Lin, and Zhongzhi Shi. A dynamic description logic for representation and reasoning about actions. In *Proceedings of the 2nd international conference on Knowledge science, engineering and management*, KSEM'07, pages 115–127, Berlin, Heidelberg, 2007. Springer-Verlag.

[28] Y. Y. Haimes and R. E. Steuer. *Research and Practice in Multiple Criteria Decision Making, Lecture Notes in Economics and Mathematical Systems, vol. 487*. Springer-Verlag, Berlin, 552 pp., 2002.

[29] Joonho Kwon, Hyeonji Kim, Daewook Lee, and Sukho Lee. Redundant-free web services composition based on a two-phase algorithm. In *ICWS*, pages 361–368, 2008.

[30] Srividya Kona, Ajay Bansal, and Gopal Gupta. Automatic composition of semanticweb services. In *ICWS*, pages 150–158, 2007.