A Constraint-Based Approach to Horizontal Web Service Composition

Ahlem Ben Hassine[1], Shigeo Matsubara[1,2], and Toru Ishida[1,3]

[1] Language Grid Project, National Institute of Information and Communications Technology
ahlem@nict.go.jp
[2] NTT Communication Science Laboratories, NTT Corporation
matsubara@cslab.kecl.ntt.co.jp
[3] Department of Social Informatics, Kyoto University
ishida@i.kyoto-u.ac.jp

Abstract. The task of automatically composing Web services involves two main composition processes, vertical and horizontal composition. Vertical composition consists of defining an appropriate combination of simple processes to perform a composition task. Horizontal composition process consists of determining the most appropriate Web service, from among a set of functionally equivalent ones for each component process. Several recent research efforts have dealt with the Web service composition problem. Nevertheless, most of them tackled only the vertical composition of Web services despite the growing trend towards functionally equivalent Web services. In an attempt to facilitate and streamline the process of horizontal composition of Web services while taking the above limitation into consideration, this work includes two main contributions. The first is a generic formalization of any Web service composition problem based on a constraint optimization problem (COP); this formalization is compatible to any Web service description language. The second contribution is an incremental user-intervention-based protocol to find the optimal composite Web service according to some predefined criteria at run-time. Our goal is *i*) to deal with many crucial natural features of Web services such as dynamic and distributed environment, uncertain and incomplete Web service information, etc; and *ii*) to allow human user intervention to enhance the solving process. Three approaches are described in this work, a centralized approach, a distributed approach and a multi-agent approach to deal with realistic domains.

1 Introduction

The great success of Web services, due especially to their richness of application made possible by open common standards, has led to their wide proliferation and a tremendous variety of Web services are now available. However, this proliferation has rendered the discovery, search and use of an appropriate Web services arduous. These tasks are increasingly complicated, especially while dealing with composite Web service to response to an ostensible long-term complex user's goal. The automatic Web service composition task consists of finding an appropriate combination of existing Web services to achieve a global goal. Solving this problem involves mixing and matching component Web services according to certain features. These features can be divided into two main groups:

I. Cruz et al. (Eds.): ISWC 2006, LNCS 4273, pp. 130–143, 2006.

- Features related to the user, including the user's constraints and preferences.
- Features related to Web services and which can be divided into two subgroups, *internal* and *external* features. *Internal* features include quality of service (QoS) attributes, and *external* features include existing restrictions on the connection of Web services, (e.g., a hotel room should be reserved for the ISWC2006 conference usually after booking the flight). *External* features are specified in the Web service ontology language, OWL-S [12], through a set of control constructs such as, *Sequence*, *Unordered*, *Choice*, etc.

However, there is usually a choice of many Web services for each subtask that has to be done to fulfill the main goal. We refer to these Web services as functionally equivalent Web services. In the sequel of this paper, as is generally done in the literature, we refer to each of subtasks making up the main goal as an *abstract* Web service and to each Web service able to perform a subtask as a *concrete* Web service. Solving a Web service composition problem means going through two types of composition process:

- *Vertical* composition, is aimed at finding the "best" combination of the *abstract* Web services, i.e., abstract workflow, for achieving the main goal while satisfying all existing interdependent restrictions.
- *Horizontal* composition, is aimed at finding the "best" *concrete* Web service, from among a set of available functionally equivalent Web services, i.e., executable workflow, to perform each *abstract* Web service. The quality of the response to the user's query (the composition task) considerably depends on the selected *concrete* Web services. The choice of a *concrete* Web service is dictated to functional (i.e., related to the inputs) and/or non-functional attributes (i.e., related to the quality of service attributes).

The main benefits from distinguishing between these two composition processes are: *i*) simplifying Web service composition problem to reduce it computational complexity, *ii*) avoiding any *horizontal* composition redundancy that may appear while searching for the "best" *orchestration* of *abstract* Web services, and mainly *iii*) ensuring more flexibility to the user intervention, i.e., user is able to modify/adjust the *abstract* workflow when needed.

The combination of Web services has attracted the interest of many researchers, amongst [9], [13], [8], [14], and several approaches have been reported. Most of these deal only with vertical composition, where only single concrete Web service is available for each abstract one. However, the tremendous growing number of functionally equivalent concrete Web services makes the search for an appropriate one, i.e., *horizontal* composition of concrete Web services, an NP-hard task [5]. This composition process has the following characteristics.

- Information is often incomplete and uncertain.
- The environment is naturally distributed and dynamic.
- Many (non)-functional features, inter-related restrictions and especially the preferences of the user may affect the quality of the response to a user's query.

Existing research efforts have tackled only some parts of the natural features of the Web service composition problem [1], [7], none have tried to deal with all of them. Also,

some complex real-world problems require some level of abstract interactions with the user to better search for a valid composite Web service. Finally, very few studies have considered the validity of the information concerning a concrete Web service during the composition process and none have dealt with this question of validity during the execution process. We have learned from all these works and we have focused our research on the requirements of the Web service composition problem that are derived from the natural features of the problem, search-based user interventions and the information validity during the composition and execution processes. Our main goal is to provide a means by which an optimal composite executable workflow can be created for a given set of sub-tasks with their inter-relation restrictions, i.e., an abstract workflow.

This paper consists of two main parts. The first is a generic formalization of any Web service composition problem as a constraint optimization problem (COP) in which we try to express most of the Web service composition problem features in a simple and natural way. Our main purpose is to develop a common and robust means of expressing any Web service composition problem that ideally reflects realistic domains. The second contribution is a real-time interactive protocol to solve any Web service composition problem by overcoming most of the above encountered limitations. Although, there are various techniques for solving a COP, none of these integrate any user interaction issues. The constraint optimization problem formalism is especially promising for ideally describing any realistic Web service composition problem, because this problem is a combinatorial problem that can be represented by a set of variables connected by constraints. Three approaches are proposed in this paper, a centralized approach, a distributed approach and finally a multi-agent approach to reflect ideally realistic domains.

This paper is organized as follows. In Section 2, we give an overview of existing researches. In Section 3, we present the proposed formalization. In Section 4, we describe a real-world scenario. In Section 5, we describe the proposed algorithm. In Section 6, we discuss possibilities of extensions of the previous algorithm. In Section 7, we conclude the paper.

2 Related Work

Several solutions to the Web service composition problem have been reported including, integer programming (IP)-based techniques [2], [16], non-classical planning-based techniques and logic-based techniques [9], [11]. Recently, some researchers have suggested applying existing artificial intelligence (AI) optimization techniques, such as genetic algorithms (GA), mainly to include some Quality of Service attributes in the search process. Regarding IP-based proposed solutions [2], [16], authors assume linearity of the constraints and of the objective function. As for non-classical planning techniques, Sirin et al. proposed an HTN-planning based approach [13] to solve this problem. Their efforts were directed toward encoding the OWL-S Web service description as a SHOP2 planning problem, so that SHOP2 can be used to automatically generate a composite web service. McIlraith and Son [9] proposed an approach to building agent technology based on the notion of generic procedures and customizing user constraints. The authors claim that an augmented version of the logic programming language Golog provides a natural formalism for automatically composing services

on the semantic web. They suggested not to consider this problem as a simple planning, but as a customizations of reusable, high level generic procedures. Canfora et al. in [5] proposed to tackle QoS-aware composition problem using Genetic Algorithm (GA). This work deals with both vertical and horizontal compositions. However, to accomplish the Web service composition task, the Web service composition procedure may need to retrieve information from Web services while operating. Most studies have assumed that such information is static [9], [13], [5]. Other studies have required an interactive process with the user to get all the necessary information as inputs. Nevertheless, the static information assumption is not always valid, the information of various Web services may change (i.e., it may be "volatile information" [1]) either while the Web service composition procedure is operating or during execution of the composition process. Kuter et al. [7] present an extension of earlier non-classical planning-based research efforts to better cope with *volatile* information. This arises when the information-providing Web services do not return the needed information immediately after it is requested (or not at all). In addition, Au et al. [1] proposed two different approaches for translating static information into volatile information. They propose assigning a validity duration for each item of information received from information-providing services.

3 Constraint-Based Formalization of Horizontal Web Service Composition

The constraint satisfaction problem (CSP) framework is a key formalism for many combinatorial problems. The great success of this paradigm is due to its simplicity, its natural expressiveness of several real-world applications and especially the efficiency of existing underlying solvers. We therefore believe that CSP formalism allows a better and more generic representation of any Web service composition problem. Hence, we formalize the Web service composition problem as a *constraint optimization problem* (COP) in which we have two kinds of constraints: *hard* and *soft* constraints.

A *static* CSP is a triplet (X, D, C) composed of a finite set X of n variables, each of which takes a value in an associated finite domain D and a set C of e constraints between these n variables [10]. Solving a CSP consists of finding one or all complete assignments of values to variables that satisfy all the constraints. This formalism was extended to the COP to deal with applications where we need to optimize an objective function. A constraint optimization problem is a CSP that includes an objective function. The goal is to choose values for variables such that the given objective function is minimized or maximized.

We define a Web service composition problem as a COP by (X, D, C, $f(sl)$) where:

- $X=\{X_1, \ldots, X_n\}$ is the set of *abstract* Web services, each X_i being a complex variable represented by a pair ($X_i.in$, $X_i.out$) where
  - $X_i.in=\{in_{i1}, in_{i2}, \ldots, in_{ip}\}$ represents the set of p inputs of the *concrete* Web service, and
  - $X_i.out=\{out_{i1}, out_{i2}, \ldots, out_{iq}\}$ represents the set of q outputs of the *concrete* Web service.

- D=$\{D_1, \ldots, D_n\}$ is the set of domains, each D_i representing possible *concrete* Web services that fulfill the task of the corresponding *abstract* Web service.
  D_i=$\{s_{ij}(s_{ij}.in, s_{ij}.out) \mid s_{ij}.in \subseteq X_i.in$ AND $X_i.out \subseteq s_{ij}.out\}$
- C=$C_S \cup C_H$
  - C_S represents the soft constraints related to the preferences of the user and to some Quality of Service attributes. For each soft constraint $C_{Si} \in C_S$ we assign a penalty $\rho_{C_{Si}} \in [0, 1]$. This penalty reflects the degree of unsatisfiability of the soft constraint C_{Si}.
  - C_H represents the hard constraints related to the inter-*abstract* Web services relations, the OWL-S defined control constructs[1], and the preconditions of each *concrete* Web service. For each hard constraint $C_{Hi} \in C_H$ we assign a weight $\perp$ (i.e. it should be imperatively satisfied). It is noteworthy that C_H may include also some *hard* constraints specified by the user, these hard constraints can be *relaxed* upon request whenever no solution is found for the problem.
- For each *concrete* Web service we assign a weight to express the degree of user preference, $w_{s_{ij}} \in [0,1]$. Weights are automatically accorded to the values of variables in a dynamic way with respect to the goal.
- $f(sl)$ is the objective function to optimize, $f(\text{sl}) = \otimes_{s_{ij} \in sl}$(*user's preferences*, *penalty over soft constraints*, *Quality of Service attributes*, *probability of information expiration*), and *sl* is a solution of the problem defined by the instantiation of all the variables of the problem. In this work, we focus on optimizing both *i*) the user's preferences toward selected *concrete* Web services denoted by φ (*sl*) and *ii*) the penalty over soft constraints denoted by $\psi(sl)$. The Quality of Service attributes and the probability of information expiration will be tackled in our future work.

Solving a Web service composition problem consists of finding a "good" assignment $sl^* \in Sol$:=$D_1 \times \ldots \times D_n$ of the variables in X such that all the hard constraints are satisfied while the objective function $f(sl)$ is optimized according to Eq. 1.

$$f(sl^*) = arg \max_{sl \in Sol} \otimes(\varphi(sl), \psi(sl)) \tag{1}$$

In this paper, we maximize the summation of the user preferences for all *concrete* Web services involved in the solution *sl* and minimize the summation of the penalties associated to all soft constraints[2] according to Eq. 2.

$$f(sl^*) = arg \max_{sl \in Sol} \Big(\sum_{s_{ij} \in sl} w_{s_{ij}} - \sum_{C_{Si} \in C_S} \rho_{C_{Si}} \Big) \tag{2}$$

Since the solution might not be only a sequence of *concrete* Web services, i.e., it may include concurrent *concrete* Web services, we use "," to indicate the sequential execution and "$\|$" to indicate concurrent execution. This information is useful in the execution process. The obtained solution will have a structure such as, *sl*=$\{s_{1i}, \{s_{2j} \| s_{3k}\}, s_{4h}, \ldots, s_{nm}\}$. This problem is considered to be a dynamic problem since the set of *abstract*

[1] Our formalization for the OWL-S control constructs will be described below in more detail.

[2] To allow more flexible and wider expression, we do not restrict the objective function to any kind of function.

Web services (the set of variables) is not fixed; i.e., an *abstract* Web service can be divided into other *abstract* Web services if there is no available *concrete* Web services to perform the required task. In addition, the set of values in the domain of each variable (the set of possible *concrete* Web services) is not fixed. *Concrete* Web services can be added/removed to/from the system.

In the Web services composition problem, several control constructs connecting Web services can be used. The main ones, defined in the OWL-S description, can be divided into four groups and we describe our formalization for these four groups below.

- Ordered, which involves the SEQUENCE control construct, can be expressed using a hard constraint. Each pair of *abstract* Web services linked by a sequence control construct are involved in the same $C_{Sequence}$ constraint.
- Concurrency involves the SPLIT, SPLIT+JOIN, and UNORDERED control constructs. The natural aspect of the following proposed agent-based approach (Section 5) allows the formalization of this control construct in a natural way. Note that only "JOIN" will be associated with a C_{Join} constraint. SPLIT and UNORDERED will be modeled using an "empty" constraint C_{empty}, that represents a universal constraint. This constraint will be used to propagate information about parallel execution to concerned variables in the following proposed protocol.
- Choice involves IF-THEN-ELSE and CHOICE control constructs. For each set of *abstract* Web services (two or more) related by the IF-THEN-ELSE or CHOICE control construct, the corresponding variables are merged into the same global variable (X_j for example), and their domains are combined and ranked according to the preference of the user. For example a set of *m abstract* Web services ($\{t_1, t_2, \ldots, t_m\}$) related by the "CHOICE" control construct, we combine them into a global variable (X_k for example) and *rank* their domains. For their preconditions, we assign a sub-constraint to each condition $\{C_{cond1}, C_{cond2}, \ldots, C_{condm}\}$ and create a global constraint $C_{Choice}=\cup_i C_{condi}$. At any time we are sure that only one condition will be satisfied since $\cap_i C_{condi}=\emptyset$.
- LOOP, neither the CSP formalism nor any of its extensions can handle iterative processing. It will be considered in our future work.

4 Real-World Scenario

Consider a situation where a person living in France wants to organize a trip to Japan to have laser eye-surgery. After the surgery, he will have to make appointments with his ophthalmologist in France for post-operative examinations. This task involves several interrelated subtasks as shown in Figure 1(a):

- t_1 = Withdraw money from the bank to pay for the plane fare, surgery, accommodation, and treatment,
- t_2 = Make an appointment with the doctor, get the address of the clinic and determine the price of the surgery,
- t_3 = Reserve a flight,
- t_4 = Reserve accommodation, which involves,

- t_{4-1} = Reserve accommodation in a nearby hotel if the price is less than or equal to US$100 per night,
- t_{4-2} = Reserve accommodation at a hostel if the cost of a hotel exceeds US$100 per night,

– t_5 = Make an appointment with his ophthalmologist for an examination one week after returning to France.

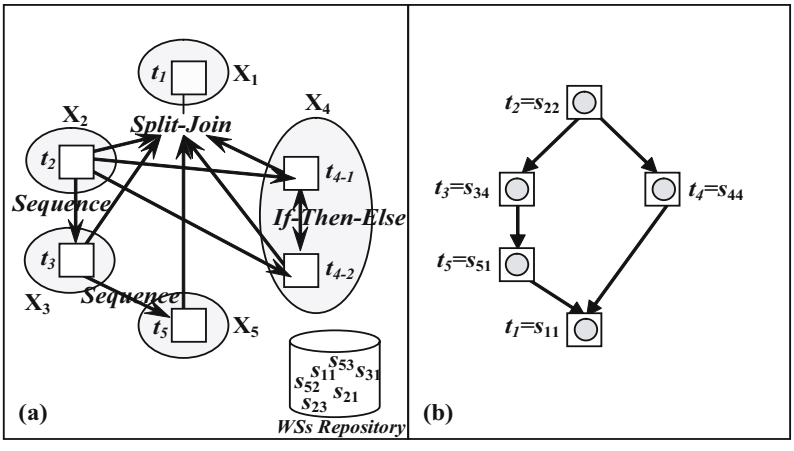

Fig. 1. (a) The set of tasks for the example with their pairwise control constructs, (b) The corresponding executable workflow solution for the problem

This problem can be formalized as follow:

– X={X_1, X_2, X_3, X_4, X_5}, where each X_i=(X_i.*in*; X_i.*out*) corresponds to one of the above tasks (Figure 1(a)).
  - X_1 corresponds to the task of withdrawing the required amount of money; X_1.*in*={Id, Password, Amount}; X_1.*out*={RemainAmount};
  - X_2 corresponds to the task of making an appointment for the surgery; X_2.*in*={Disease, Date}; X_2.*out*={ClinicName, Place, Confirmation, Price};
  - X_3 corresponds to the task of booking a flight; X_3.*in*={Destination, Date, PatientName}; X_3.*out*={FlightNumber, Price};
  - X_4 corresponds to the two tasks to reserve accommodation in either a hotel or a hostel depending to the cost. Recall that in our formalization we combine into the same variable the tasks implied in the same CHOICE relation. In this example t_{4-1} and t_{4-2} are involved in the same IF-THEN-ELSE control construct, so we combine them into X_4; X_4.*in*={Name, Place, Date, NightsNumber, MaxPrice}; X_4.*out*={Hotel/hostelName, Address, Price};
  - X_5.*in*={DoctorName, PatientName, Date, TreatmentType}; X_5.*out*={Confirmation, Price};

- D={D_1, D_2, D_3, D_4, D_5}, where:
  D_1={s_{11}}, D_2={s_{21}, s_{22}, s_{23}}, D_3={$s_{31}, s_{32}, s_{33}, s_{34}, s_{35}$}[3], D_4={$s_{41}, s_{42}, s_{43}, s_{44}$},
  D_5={$s_{51}, s_{52}, s_{53}, s_{54}$},
- C=$C_S \cup C_H$, where
  - C_H including
    - X_1.Id $\neq$ *nil*;
    - X_1.Amount $\geq$ X_2.Price + X_3.Price + X_4.Price + X_5.Price
    - X_2.Date $<$ X_3.Date;
    - X_3.Date $<$ X_4.Date;
    - X_4.Price $\leq$ US$100;
    - X_4.Date + X_4.NightsNumber+7 $<$ X_5.Date;
  - C_S including
    - *Distance*(X_4.Place, X_2.Place)$\leq$10km[4].
- For each $s_{ij} \in D_i$, we assign a weight $w_{s_{ij}}$ to express the degree of preferences of the user *PrefUser*(D_j),
  PrefUser(D_1)={1}, *PrefUser*(D_2)={0.26, 0.73, 0.58}, *PrefUser*(D_3)={0.53, 0.61, 0.35, 0.82, 0.12}, *PrefUser*(D_4)={0.33, 0.71, 0.63, 0.84}, *PrefUser*(D_5)={0.87, 0.25, 0.59, 0.66}.
  These degrees of preferences are subjective values and depend on the user.
- The main objective is to find the best combination *sl* of the above *abstract* Web services and assign the most appropriate *concrete* Web services such that *sl* maximizes the objective function *f*(*sl*) defined in Section 3 Eq. 2. Note that for simplicity, we assume inter-independence between the values of the different domains. We will consider dependence issues in future work.

Assume that *Distance*(s_{21}, s_{44})= 13km, *Distance*(s_{22}, s_{44})= 11km and *Distance*(s_{23}, s_{44})=20km, and the penalty over this soft constraint, *Distance*(X_4.Place, X_2.Place)$\leq$10km decreases as the *distance* converges to 10km, then $\rho_{Distance(s_{22},s_{44})} < \rho_{Distance(s_{21},s_{44})} < \rho_{Distance(s_{23},s_{44})}$. The most preferred solution for this problem is, *sl*={s_{22}, {{s_{34}, s_{51}} $\| s_{44}$}, s_{11}} (Figure 1(b)) with φ(*sl*)= 0.73+0.82+0.84+0.87+1=4.26.

5 Constraint Optimization Problem Interactive Algorithm for Solving the Web Service Composition Problem

The overall objective of our approach is to generate the *best* executable workflow (according to the aforementioned criteria) within a feasible time. Several constraint optimization problem algorithms can be applied to solve this problem, but none allows the intervention of the human user during the search process. In the following, we propose an algorithm (Algorithm 1) that allows human interaction with the system to enhance the solving process.

For each variable X_j[5] we first determine a set of candidate *concrete* Web services, $Cand_{X_j}$ for its *abstract* Web service that satisfies all the hard constraints $C_{Hl} \in C_H$ (Algorithm 1 line 4), and then we *rank* $Cand_{X_j}$ according to the objective function defined

[3] For example, Air France Web service, Lufthansa Web service, etc.
[4] *Distance*(*x*, *y*) is a function that returns the distance between two places.
[5] The variables are ordered according to the input *abstract* workflow.

in Section 3. This ranked set is used to guide the selection of the next variable X_{j+1} in the search process. For X_{j+1} we proceed first by applying *join* operation to the received list $Cand_{X_j}$ and the current one $Cand_{X_{j+1}}$, i.e., $Cand_{X_j} \bowtie Cand_{X_{j+1}}$ (Algorithm 1 line 12). The obtained sub-solutions are then *filtered* (Algorithm 1 line 12) according to the set of existing hard constraints. Finally, the resulting set of sub-solutions is ranked according to the objective function for optimization. If the set of candidates $Cand_{X_j}$ is large, to avoid explosion in the join operation, we select a fixed number of the most preferred *concrete* Web services for each variable, (i.e., a subset of candidates), and try to propagate these to the next variable. Whenever this subset does not lead to a complete solution, we backtrack and then seek a solution using the remaining candidates. The order of the values in the candidate set is established to avoid missing any solution. The obtained sets of sub-solutions are propagated to the next variable (Algorithm 1 line 16) and the same dynamic resumes until the instantiation of all the *abstract* Web services. If the set of candidate Web services becomes empty (i.e., none of the available Web services satisfies the hard constraints), or the set of sub-solutions resulting from the join and filter operations becomes empty and no more *backtrack* can be performed, the user is asked to relax some of his/her constraints (Algorithm 1 line 23). However, if the relaxed user's constraints involve the first instantiated variable in the search tree then the search process is performed from scratch. It is noteworthy that three issues are possible in this algorithm, *i*) Ask user intervention whenever a local failure is detected, which may reduce the number of backtracks, *ii*) Ask user intervention only when a global failure is detected, no more backtracks can be performed, *iii*) keep trace of the explored search tree to be able to point directly to the concerned variable by user relaxation and pursue the solving process and avoid some computational redundancy.

In addition, whenever we need any information concerning any *concrete* Web services, a request-message is sent to an information-providing Web service to get the necessary information along with both its validity duration and the maximum time required to execute the underlying Web service. The agent should maintain this time so that it can detect the information expiration and perform right decision (Algorithm 1 line 20). To deal with the main characteristic of this real-world problem, the dynamic environment, we maintain the validity of necessary information during the solving and execution processes, *totalTime*. *totalTime* should be less than the minimum validity time required for any Web service information. We use the following denotation:

- $T_{plan}(sl)$: necessary time needed to provide a plan *sl*,
- $t_{exe}(s_i)$: needed time to execute one *concrete* Web service,
- $t_{val}(inf_j)$: estimated time before the expiration of solicited information inf_j.

Naturally, the validity of information is usually considered as uncertain. Hence, for each validity time a *probability of information alteration* $p_{alt}(inf_i)$ can be associated with to the underlying information inf_i. We will consider this probability of information alteration in our future work. The maximal time T_{plan} required to provide a solution is defined by Eq. 3.

$$T_{plan}(sl) < \min_{\forall\ s_i\ \in\ sl;} t_{val}(inf_j) - \sum_{s_j \in sl} t_{exe}(s_j); \tag{3}$$

Algorithm 1. User-intervenstion-based algorithm for Web service composition

***WSCSolver*(*i*, *setSubSol*, *totalTime*, *checkedValues*)**

1:. **if** $i > \|\mathrm{X}\|$ **then**
2:. return *setSubSol*;
3:. **end if**
4:. $Cand_X[i] \leftarrow \{s_{ik} \in \mathrm{D}_i \mid s_{ik}$ *satisfies* all the $\mathrm{C}_H\} \setminus$ *checkedValues*[*i*];
5:. **if** *information required for any* $s_{ij} \in Cand_X[i]$ **then**
6:. Collect necessary information; Update t_{val}, t_{exe} and *totalTime*;
7:. **end if**
8:. *Rank* $Cand_X[i]$ according to $w_{s_{ij}}$ and $\rho_{C_{Sj}}$ and while checking t_{val}, t_{exe} and *totalTime*;
9:. *subSol* $\leftarrow \emptyset$;
10:. **while** *subSol* = $\emptyset$ **do**
11:. *subCand* $\leftarrow$ subset of the $Cand_X[i]$; add(*checkedValues*[*i*], *subCand*);
12:. *subSol* $\leftarrow$ *setSubSol* $\bowtie$ *subCand*; Filter and Rank *subSol* according to f(*subSol*);
13:. **end while**
14:. **if** *subSol* $\neq \emptyset$ **then**
15:. add(*setSubSol*, *subSol*);
16:. return ***WSCSolver***(*i*+1, *setSubSol*, *totalTime*, *checkedValues*);
17:. **else**
18:. **if** $i > 1$ **then**
19:. reset to $\emptyset$ all *checkedValues*[*j*] for $j > i$;
20:. Update *totalTime*; Update *setSubSol*;
21:. return ***WSCSolver***(*i*-1, *setSubSol*, *totalTime*, *checkedValues*);
22:. **else**
23:. *RelaxedConst* $\leftarrow$ ask User to relax constraints involving X_k where $k < i$;
24:. Update(C_H, C_S, *RelaxedConst*);
25:. $i \leftarrow j$ such that $\forall\ \mathrm{X}_k$ *involved in* C_l and $\mathrm{C}_l \in$ *RelaxedConst*, $\mathrm{X}_j \prec_{lo} \mathrm{X}_k$;
26:. Update *setSubSol*;
27:. return ***WSCSolver***(*i*+1, *setSubSol*, *totalTime*, *checkedValues*);
28:. **end if**
29:. **end if**

Each sub-solution based on expired information will be temporarily discarded but kept for use in case the agent cannot find any possible solution. This measurement is an efficient way to cope with Web services with effects characterized mainly by their volatile information because it allows a forward estimation of the validity of information during both the composition process and the execution process.

6 Extended Algorithms

6.1 Web Service Composition Problem Distributed Algorithm

The main limitation of the previous algorithm is that it cannot be easily adapted to any alteration in the environment. Whenever a user decides to relax some of his/her constraints, and these constraints involve already invoked variable, especially the first one in the search tree, the search for a solution will be performed from scratch. However,

distributed approaches can be easily adapted to the user intervention. In this solution the same algorithm will be split on among set of homogeneous entities. Each entity will be responsible of one variable and the same algorithm will be performed in parallel by this set of entities. In case of conflicts, i.e., no solution can be generated and no *backtrack* can be performed, the system will ask the user to relax some constraints. The concerned entity will update its view, generate new candidates and exchange them with other concerned entities. The process resumes until either a solution for the problem is generated or its insolubility, even with all possible relaxations, is proven. Nevertheless, this distributed solution might be inefficient for some real-world scenarios where we need to access a specialized Web service. A specialized Web service maintains information about a set of Web services; for example, HotelsByCity.com maintains information about several hotels' Web services. The information concerning involved Web services is considered private, which makes it difficult to gather Web services needed information on same site and process them. Hence, we believe that extending the above algorithm to a multi-agent system is more effective for realistic domains.

6.2 Multi-agent System for Web Service Composition Problem

The underlying multi-agent architecture consists of three kinds of agents, *abstract* Web service agents, one or more Information-providing agents and an Interface agent. The Interface agent is added to the system to inform the user of the result. Each agent A_i maintains total validity time for all selected Web services, $valTime^{A_i}$. This information is locally maintained by each agent and updated each time a requested information is received. All the agents will cooperate together via sending point-to-point messages to accomplish their global goal. We assume that messages are received in the order in which they are sent. The delivery time for each message is finite. The agents are ordered, according to the input *abstract* workflow, from higher priority agents to lower priority ones so that each constraint will be checked by only one agent. For each successive two subtasks t_i and t_j such that $t_i < t_j$, their corresponding agents will be ordered as follows: $A_i \prec_{lo} A_j$, and the agent A_i (*resp.* A_j) is called *Parent* (*resp. Children*) for A_j (*resp.* A_i). The ordered links between agents, from the *Parents* to their *Children*, represent the inter-agent hard constraints between the corresponding *abstract* Web service; i.e., these relations represent OWL-S control constructs (sequence, choice, ordered, etc.) and/or hard/soft user constraints.

Each agent A_i first reduces the set of candidate *concrete* Web services, $Cand_{X_j}$ for its *abstract* Web service by keeping only those that satisfy all the hard constraints (Algorithm 2, line 1), *ranks* it according to the user preferences (i.e., $w_{s_{jk}}$), and to the degree to which the soft intra-constraints are satisfied (Algorithm 2, line 15), selects subset of "best" candidates then sends it to its $Children^{A_i}$ (Algorithm 2, line 17). If the set of candidate Web services is empty, then the user is asked to relax some of his/her constraints. In addition, whenever the agent needs information concerning any *concrete* Web service, it sends a message (*RequestInformationFor:*) to the information-providing agent to get the necessary information along with its validity and the maximum time needed to execute the underlying Web service (Algorithm 2, line 10). The agent should retain this time so that it can detect information expiration and perform right decision, i.e., update the current solution when necessary. Each agent receiving needed infor-

Algorithm 2. ***Start*** message executed by each agent A_j.

Start

1:. Select $Cand_{X_j} \subseteq D_j$ / all intra-$C_H^{A_j}$ are satisfied;
2:. *listRequest* $\leftarrow \emptyset$;
3:. **while** $Cand_{X_j}=\emptyset$ **do**
4:. Ask user to relax some of his hard constraints;
5:. **end while**
6:. **for all** $s_k \in Cand_{X_j}$ **do**
7:. **if** inf_k required for s_k **then**
8:. *listRequest* $\leftarrow$ *listRequest* $\cup\ s_k$;
9:. **end if**
10:. *send*(*Information-providing*, *self*, *RequestInformationFor*:*listRequest*);
11:. **end for**
12:. **while** *listRequest* $\neq \emptyset$ **do**
13:. Wait; /**Information required for Web services**/
14:. **end while**
15:. Rank $Cand_{X_j}$ according to $w_{s_{jk}}$ and $\rho_{C_{Sl}}$;
16:. **for all** $A_i \in Children^{A_j}$ **do**
17:. *send*(A_i, *self*, *process*:$CandX_j$ within:$valTime^{A_j}$);
18:. **end for**

mation from the Information-providing agent first updates its dynamic knowledge, and then checks whether any of the information may expire before executing the workflow. If this is the case for any of the received information, the affected Web service, s_{jk} will be discarded from the set of possible candidates. Finally, the agent ranks the remaining candidates and sends them to its *Children* for further processing. Each agent A_i receiving a message to process candidate *concrete* Web services from its Parents or to process a set of sub-solutions proceeds by first performing a *join* operation on all received lists (Algorithm 3 line 1). The obtained sub-solutions are then filtered according to the set of existing hard constraints and then ranked according to the soft constraints and user preferences (Algorithm 3 line 3). If the set of sub-solutions is empty for the agent A_i, then a request is sent to parents to ask for more possible candidates in a predefined order to ensure the completeness of the proposed protocol (Algorithm 3, line 7). In case, all the possible candidates are processed and the set of possible solution is still empty, the concerned agent asks the user to relax some of his/her constraints related directly or indirectly to the variable X_i maintained by A_i. Thus, the appropriate agent will be invoked to first update its set of hard constraints and then define new candidates and send them again to the *Children* (Algorithm 3, line 14). The same process resumes until stable state is detected.

In real-world scenarios, the Web service composition problem is subject to many changes, defined on one side by the arrival of new Web services and on the other side by the inaccessibility of one or more Web services. For each new Web service, the appropriate agent will check whether this Web service can be included in the set of candidates. If this new Web service satisfies the hard constraints and increases $f(sl)$, it will be communicated to the *Children* to upgrade their set of sub-solutions, if possible.

Algorithm 3. ***Process-within*** message executed by each agent A_i.

Process:*list*^{A_h} **within:***t*

1:. *PossibleTuple*$^{A_i} \leftarrow$ *Cand*$_{X_i}$; *PossibleTuple*$^{A_i} \leftarrow$ *PossibleTuple*$^{A_i} \bowtie$ *list*^{A_h};
2:. **if** All *list*^{A_h} are received from *Parents*^{A_i} **then**
3:. Filter *PossibleTuple*^{A_i} such that $\forall$ *tuple*$^{X_i} \in$ *PossibleTuple*^{A_i}, *tuple*^{X_i} satisfies the inter-agent constraints ($C_H^{A_i}$) and optimize the predefined criteria (Section 3);
4:. update *totalTime*^{A_i};
5:. **if** *PossibleTuple*$^{A_i} = \emptyset$ **then**
6:. **if** Possible *backtrack* **then**
7:. *send Backtrack* message to Parents^{A_i} to ask for more candidates;
8:. **else**
9:. Ask user to relax some of his hard constraints related in/directly to X_i;
10:. **end if**
11:. **else**
12:. Rank *PossibleTuple*^{A_i} according to the criteria defined in Section 3;
13:. **for all** $A_j \in$ *Children*^{A_i} **do**
14:. *send*(A_j, *self*, *process*:*PossibleTuple*^{A_i} within:*valTime*^{A_j});
15:. **end for**
16:. **end if**
17:. **end if**

Otherwise, the new candidate will be ignored. As for each Web service that becomes inaccessible during the composition process, the appropriate agent should first check whether this Web service is included in the set of sent candidates. If this is not the case, the agent will only update its dynamic knowledge; if the inaccessible Web service has already been communicated to the *Children*, the agent should ask its *Children* temporarily not consider this Web service in case it is involved in their sub-solutions.

The stable state is progressively detected by all the *abstract* Web service agents [4]. The main idea is to define an *internal state* for each agent A_i. This state is set to *true* if and only if the internal states of all the children are *true* and agent A_i succeeds in finding an appropriate *concrete* Web service for its *abstract* one. The stable state will be detected by the children and progressively propagated to the parents. Each agent that has no parents, *Parents*$^{A_i} = \emptyset$, informs the Interface agent regarding the final state. The Interface agent communicates the result to the user.

7 Conclusion

The Web service composition problem is a challenging research issue because of the tremendous growth in the number of Web services available, the dynamic environment and changing user needs. In this paper, we have proposed a real-time interactive solution for the Web service composition problem. This problem consists of two main composition processes, vertical composition and horizontal composition and we have focused on the horizontal composition process. This work complements existing techniques dealing with vertical composition in that it exploits their abstract workflow to

determine the *best* executable one according to predefined optimality criteria. We have developed a protocol that overcomes the most ascertained limitations of the existing works and comply with most natural features of a realistic Web service composition problem such as the dynamism of the environment and the need to deal with volatile information during the composition and execution processes, etc. Three main approaches were proposed in this paper, the first is a user-intervention based-centralized approach, the second is a distributed version of the previous one that can be easily adapted to any environment's alterations and the third is a multi-agent approach to cope better with realistic domains where problem required information is maintained by specialized Web services. The multi-agent approach is currently under implementation and testing.

References

1. Au, T-C., Kuter, U. and Nau, D., Web Services Composition with Volatile Information. *In proc. ISWC'05*, pp. 52-66, 2005.
2. Aggarwal, R., Verma, K., Miller, J., and Milnor, W. Constraint Driven Web Service Composition in METEOR-S. *In proc. IEEE Int. Conf. on Services Computing*, pp.23-30, 2004.
3. Aversano, L., Canfora, G. Ciampi, A., An algorithm for web service discovery through their composition. *In proc. IEEE ICWS'04*, 2004.
4. Ben Hassine, A., and T.B. Ho, Asynchronous Constraint-based Approach - New Solution for any Constraint Problem. *In proc. AAMAS RSS'2006*, 2006.
5. Canfora, G., Penta, M.D., Esposito, R. and Villani, M.L., An Approach for QoS-aware Service Composition bsed on Genetic Algorithms. *In proc. ACM GECCO'05*, pp. 25-29, 2005.
6. Dechter, R. and Dechter, A., Belief Maintenance in Dynamic Constraint Networks. *In proc. 7th National Conf. on Artificial Intelligence, AAAI-88*, pp. 37-42, 1988.
7. Kuter, U., Sirin, E., Parsia, B., Nau, D. and Hendler, J., Information Gathering During Planning for Web Service Composition. *In proc. ISWC'04*, 2004.
8. Lin, M., Xie, J., Guo, H. and Wang, H., Solving Qos-driven Web Service Dynamic Composition as Fuzzy Constraint Satisfaction. *In proc. IEEE Int. Conf. on e-Technology, e-Commerce and e-service, EEE'05*, pp. 9-14, 2005.
9. McIlraith, S. and Son, T.C., Adapting Golog for Composition of Semantic Web Services. *KR-2002, France*, 2002.
10. Montanari, U., NetWorks of Constraints: Fundamental Properties and Applications to Picture Processing. *In Information Sciences*, Vol. 7, pp. 95-132, 1974.
11. Narayanan, S. and McIlraith, S., Simulation, Verification and automated Composition of Web Services. *In Proceeding 11th Int. Conf. WWW*, 2002.
12. OWL Services Coalition, OWL-S: Semantic markup for web services, *OWL-S White Paper http://www.daml.org/services/owl-s/1.0/owl-s.pdf*, 2003.
13. Sirin, E., Parsia, B., Wu, D., Hendler, J. and Nau, D., HTN Planning for Web Service Composition Using SHOP2. *In Journal of Web Semantic Vol. 1*, pp. 377-396, 2004.
14. Ishida, T., Language Grid: An Infrastructure for Intercultural Collaboration. Valued Constraint Satisfaction Problems: Hard and Easy Problems. *In IEEE/IPSJ Symposium on Applications and the Internet (SAINT-06)*, pp. 96-100, 2006.
15. Yokoo, M. Ishida. T, and Kuwabara, K. Distributed Constraints Satisfaction for DAI Problems. *In 10th Int. Workshop in Distributed Artificial Intelligence (DAI-90)*, 1990.
16. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., and Chang, H. QoS-aware middleware for web services composition. *IEEE Trans. Software Engineering, 30(5)*, 2004.