QoS Aware Semantic Web Service Composition Approach Considering Pre/Postconditions*

Peter Bartalos and Mária Bieliková
Institute of Informatics and Software Engineering, Faculty of Informatics
and Information Technologies, Slovak University of Technology in Bratislava
{bartalos,bielik}@fiit.stuba.sk

Abstract

Web service composition is a topic bringing several issues to be resolved. Our work deals with the effectiveness and scalability of service composition. During composition we consider QoS and pre-/post-conditions of single services to create a composite service satisfying the user needs the best. Regarding pre-/post-conditions we propose an approach to fast determination of which services produce results expected by the user, i.e. the post-condition of which services implicates the desired condition defined in the user goal. This paper presents also a renewed version of our approach to QoS aware service composition. We achieved a dramatic improvement in terms of composition time by performing a useful restriction on the service space.

1 Introduction

Web services present a topical research area with lot of attention. One part of this research aims to propose solutions to automatic composition of several web services into workflows bringing a utility which cannot be provided by single service. The desired result of such composition is described in the user query. The automatic web service composition showed to be a challenging task [8].

The research of service composition in last years tends to focus on issues related to QoS [2, 4, 7, 14], pre-/post-conditions [3, 9], user preferences (e.g. soft constraints) [1, 11], service selection considering complex dependencies between services [5, 13]. Our work deals with the effectiveness and scalability of *explorative service composition* aware of QoS and pre-/post-conditions. It is based on our previous work [3, 4]. Already the previous approach showed to have good performance as it took part at the *Web Services Challenge 2009* and was one of the best solutions. We had enhanced it by a process restricting the set of services which must be considered when looking for a solution. As a result we achieve improvement in terms of shorter composition time in more than one order of magnitude. Regarding QoS, we do not present any new method to evaluation of multi-dimensional quality of a composite service. Our approach selects the best solution considering the aggregated value of a given quality attribute. In the context of pre-/post-conditions we introduce a novel approach considering more expressive conditions with value restrictions.

2 Related work

In [4] and [7], two approaches which took a part at the Web Services Challenge 2009 are presented. Web Services Challenge is a competition aimed at developing software components and/or intelligent agents that have the ability to discover pertinent web services and also compose them to create higher-level functionality[1]. In 2009 it focused on automatic web service composition considering the QoS [8]. Both [4] and [7] proved to be scalable approaches. Even during the hardest data set at the competition, consisting from 15 000 web services, they were able to find a solution in acceptable time[2] (below 300 msecs). Both approaches realize preprocessing during which effective data structures are build. These are used during the user querying phase to quickly compose a desired composition. Approach presented in [4] benefits from a data structure based on a relational database and parallel process execution. Composition in [7] is effective due a filtering utilized to reduce the search space. The pruning removes services i) which have no inputs (cannot be executed) and ii) do not have optimal QoS.

*This work was partially supported by the grant VEGA 1/0508/09 and it is the partial result of the Research & Development Operational Programme for the project Support of Center of Excellence for Smart Technologies, Systems and Services, ITMS 26240120029, co-funded by the ERDF.

[1] http://ws-challenge.georgetown.edu/wsc09

[2] The time includes a call of the composition system from a client application, composition and result transformation to BPEL format, and realization of a callback from the composition system to submit the result to the client application.

The removing of unusable services is applied also in [4].

An other approach to QoS aware automatic web service composition dealing with scalability is described in [2]. Unlike to [4] and [7], this approach does not deal with a design of the structure of the composite service. As an input it already takes an abstract service, i.e. the aim is only to select concrete services, for each used service class, resulting in the best aggregated QoS. The two previous approaches find the best plan for each QoS characteristic separately. This approach uses a utility function to express the overall quality from the individual quality attribute values. The calculation of the utility function is based on *Simple additive weighting technique*. It involves scaling of the quality attribute values to allow uniform measurement independently on the unit and range of the given attribute. Then, weighting process follows to represent user priorities and preferences. In this context the approach deals with the scalability issues. It is based on a heuristic algorithm decomposing the original optimization problem into sub-problems which can be solved more effectively. The decomposition allows finding the best candidate for each service class separately, i.e. it is not required to check all possible service combinations. Before this, only two global parameters for each attribute must be calculated. After this, calculations are performed locally for each service class. The approach presents good scalability when finding near-to-optimal solution, according the number of service classes and candidates per class.

In [6] the authors present a composition approach handling so called user constraints. These represent value restriction constraints to input, output or local parameters of services. The approach has a differing composition problem definition as in [3] and [9]. It seems that the problem neglects the design of the composite service's structure, i.e. the set of used services and the control/data flow are partially known (similarly as in [2]) and the related issues are not a part of the problem solving. For each service the user may define value restrictions over the services' parameters. Thus, the user should know which services are used in the composition. The approach takes care about satisfying these constraints during service execution. Before execution, it is checked if the input data hold the constraints to input parameters. Constraints to output parameters are checked after execution. If these do not hold, the user is informed about the failure, or a back-track mechanism is realized. The user may define also multi-service restrictions. Such constraint is checked for first services. After it is executed the value of the constraint is updated and used for checking the next services. In opposite to [3] and [9], this approach takes care only about conditions defined by the user. It does not deal with the pre-/post-conditions of services during chaining.

In the approach proposed in [9] the USDL language is used to specify the formal semantics of services [10]. The pre-/post-conditions are expressed as atomic statements combined with conjunction, disjunction, and negation. The approach focuses also on other important features of effective composition. One of them is the ability of incremental updates. In dynamic world the set of composed services is changing. Services may be added/removed. The composition system should react to these changes quickly. This usually can not be achieved if the change requires a complete reconstruction of the internal service repository. It is desired to handle the change by adding new data to the repository affecting only a local part of it.

Our previous approach to pre-/post-condition aware automatic web service composition is described in [3]. It uses OWL-S to depict the semantics of services. The conditions are described using SWRL (Semantic Web Rule Language) and its extension to first order logic (`http://www.daml.org/2004/11/fol/proposal`). From the expressivity of pre-/post-conditions point of view, this approach is equivalent to [9]. The aim of the work is a scalable, pre-/post-condition aware composition. The approach bases on an effective service representation in an internal data structure having two parts. One part is stored in a relational database (RDB) and the second in operational memory. The RDB is used to select services directly satisfying the user goal (final services in the composition) and services having all inputs provided in the user query (initial services). The capabilities of RDB are used to quick selection of these services. Moreover, RDB takes care also about finding services having post-condition satisfying the condition defined in the user goal. This is achieved by a normalization of the formulae expressing the conditions to disjunctive normal form (DNF). The overall reasoning required to decide which services have satisfying post-condition is realized using the capabilities of RDB management systems and a piece of program. Similarly to [9], also this approach is able to react to changes in the service set quickly, by performing only a local change in the internal service registry.

3 Service composition preliminaries

The aim of semantic web services is to provide functionality, which can produce required data and effects in a widely accessible, easily discoverable form (by machines). Semantic web service composition enhance to potential of single services. Its aim is to combine several services to supply more complex needs.

Web service composition problem is defined as follows: given a query describing the goal and providing some inputs, design a workflow (depicting the data- and control-flow) from available services such that if it is executed, it produces the required goal. In this context, both the query and services are described at the semantic level. The required/provided data presented in the query and the I/O

of services have defined meaning. This is done by their grounding to concepts in the ontology. To enhance the opportunities we deal also with conditions representing the desired effects in the goal, conditions which must hold before service execution, or hold after service execution. These are defined using predicates forming a logical formula. The condition may have defined semantics by grounding the predicates to properties in the ontology and grounding the predicate arguments to concepts in the ontology. However, in this case we can use only binary predicates. The predicate arguments are variables corresponding to the I/O or represent local, ground variables. In the following, we formalize the service composition problem.

Definition 1 ***(Repository of Services):*** *Repository R is a set of available services.*

Definition 2 ***(Condition):*** *Condition C is a 2-tuple $C = (Str, Val)$. Str is a logical formula consisting from n-ary ground predicates combined using logical operators $\wedge, \vee, \neg$. It depicts the structure of the condition. Val is a list of value restrictions of the variables appearing as the arguments of the predicates. These variables are ground to concepts in the ontology.*

Definition 3 ***(Condition implication):*** *Let denote the set of sub-formulae representing the conjunctions of the formula Str transformed to DNF, as Str_{DNF}. Condition $C1 = (Str1, Val1)$ implicates condition $C2 = (Str2, Val2)$, denoted as $C1 \Rrightarrow C2$, iff the following holds:*

1. *There is a conjunction in $Str1_{DNF}$ implying some conjunction in $Str2_{DNF}$, i.e. $\exists str1 \in Str1_{DNF}, \exists str2 \in Str2_{DNF}$ such that $\models str1 \Rightarrow str2$.*
2. *Each variable $v1$ appearing in $str1$ subsumes the corresponding variable $v2$ in $str2$, i.e. $v1 \sqsubseteq v2$.*
3. *If the variable of $str2$ has defined a value restriction, this is not violated by the value restriction of the corresponding variable of $str1$.*

Definition 4 ***(Service):*** *Service S is a 4-tuple $S = (I, O, Pre, Post)$. I/O is a list of inputs/outputs, i.e. variables ground to concepts in the ontology. Pre is a condition which must hold before service execution. $Post$ is a condition which holds after service execution. Services are the elements of service repository R.*

Definition 5 ***(Query):*** *Query Q is a 3-tuple $Q = (I', O', Conds)$. I' is a list of provided inputs. O' is a list of required outputs. Both I' and O' consists of variables ground to the concepts in the ontology. $Conds$ is a list of conditions which are required to hold after execution of service composition. Each condition corresponds to one required output.*

Definition 6 ***(Service composition):*** *Service composition is a directed acyclic graph $G = (V, E)$, where $\forall S \in V : S \in R$. The following conditions must hold:*

1. *Each service $S_m = (I_m, O_m, Pre_m, Post_m)$ in the composition must have provided each input $i \in I_m$. The input is provided in the query, or as an output of ancestor service $S_n = (I_n, O_n, Pre_n, Post_n)$. Each input i must be subsumed by its provider ip.*
   $\forall S_m \in V, \forall i \in I_m : \exists ip \in \bigcup_n O_n \cup I'$ such that $ip \sqsubseteq i$.
2. *For each chained service pair S_{anc}, S_{succ}, it must hold that the postcondition of the ancestor service implicates the precondition of the successor service, i.e. $\forall e \in E, e = (S_{anc}, S_{succ}) : Post_{anc} \Rrightarrow Pre_{succ}$.*
3. *Each required output from O' is provided by some service S_m, i.e. $\forall o' \in O' : \exists o \in \bigcup_m O_m$ where $o \sqsubseteq o'$. If there is defined a value restriction for o', this is not violated by the value restriction of o.*
4. *Each required condition C' from $Conds$ corresponds to some required output o'. C' is implicated by the postcondition $Post$ of each service S producing the required output o', i.e. $\forall C' \in Conds$ it holds that if S produces o such that $o \sqsubseteq o'$, then $Post \Rrightarrow C'$.*

The first basic step in the service composition is to decide which services can produce the required data and achieve a desired state. The first issue is simpler. Here, the task is only to decide if the given output of the potential services subsumes the output required in goal. The second issue requires evaluation of a logical formula, i.e if a post-condition of a potential service implicates a given condition in goal. The post-condition of service in general prescribes several alternative conditions which may hold after execution. These correspond to the respective conjunctions of the post-condition transformed to DNF. Thus, the evaluation whether the post-condition of the service implicates the required goal condition can be done only after its execution. The only situation when execution is not required is when each conjunction of the post-condition implicates some conjunction of the goal condition. The implication holds only in the case that certain conjunction of post-condition is the same or an extension of the conjunction of goal condition (it may contain additional predicates). Moreover, there has to be a unique mapping between the variables appearing in the conjunctions.

Also in the context of value restrictions, the execution is required in general. The values of the variables appearing in post-condition, which correspond to some outputs, are known after execution. Hence, it does not make sense to have a concrete value restriction in the description of the service's postcondition. This would mean that we know the value of the output before execution. In some cases,

the character of the service may allow to define a minimal/maximal value of the output.

In general, the evaluation whether some conjunction implicates an other one, is a problem of high complexity. This problem can be transformed to subgraph isomorphism problem. A conjunction is transformed to a labeled graph, where each node corresponds to one variable. There is a directed, labeled edge between two nodes iff there is a predicate between the variables corresponding to these nodes. Then, the evaluation of the implication is equivalent to evaluation if there is a subgraph isomorphism between the graphs. It is known that subgraph isomorphism problem is NP-Complete. Hence, the known algorithms perform in exponential time. However, there exists an algorithm running efficiently in practical scenarios [12].

The situation in conjunction implication evaluation in practical examples is easier that in subgraph isomorphism. Several characteristics of conjunctions, which are fast to evaluate, can be used to quickly claim that the implication does not hold. Speculating practical examples, the problem is not NP-hard. Neglecting everything else, if there is at least one pair of mapped variables in the conjunctions, then the problem can be solved in polynomial time. In the context of service composition, there is always at least one variable in the goal conjunctions which is mapped implicitly, e.g. the variable representing the output parameter.

4 Method of effective composition

Our approach (see Fig. 1) is based on a lot of preprocessing done before responding to user queries. During it we create data structures which are used to quickly answer the query. The most important is that we evaluate which services can be chained, i.e. which services produce data and have a post-condition required by the other services. This can be done without a knowledge of any query which will be processed. The fast evaluation of the implication between two conditions is based on encoding of their different characteristics. The encoding of services' pre-/post-conditions and the evaluation if the post-condition of the service producing required data for another service implicates its precondition are done during preprocessing. The problem still remaining is to i) select services producing the required outputs and condition (final services in the workflow), Fig. 1-a) and ii) evaluate which services can be used, since they have provided inputs and how they interconnect (design of the data-/control-flow), Fig. 1-b). The latter is significantly affected also by the selection of the service combination with the best aggregated QoS. This is computed from the respective QoS values of used services based on aggregation rules, just as defined in [14]. Preprocessing may be used to precalculate the overall QoS characteristic of a single service from its individual quality attributes based on a utility function.

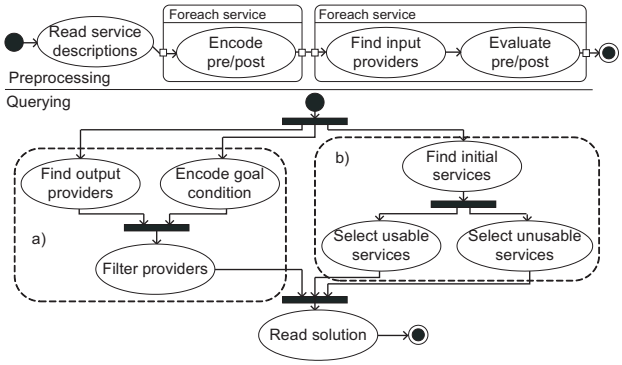

Figure 1. Overview of composition process.

4.1 Finding services producing goal

To find the services directly producing the required goal (final services in the workflow) a two step process is performed, Fig. 1-a). First, we find services providing the required outputs. Second, we filter these services based on post-conditions. Instead of searching for a service producing desired post-condition in the whole service registry, we only check whether the service providing the required outputs satisfies the condition restriction.

The search for the services providing the required outputs is done in $O(1)$ time for each required output, i.e. in $O(|O'|)$ at all. To achieve this, during preprocessing we create a hash table storing lists of services producing certain concept. The concept is used as a key in the table. The respective list includes also services producing an output subsuming the concept used as the key.

The evaluation if $Post \Rrightarrow C'$, where $C' = (Str', Val'), C' \in Conds$ and $Post = (Str, Val)$ is realized as a search for a mapping between variables from $str' \in Str'_{DNF}$ to variables from $str \in Str_{DNF}$. To faster the mapping, we encode several characteristics of the conjunctions and their elements. The selection of characteristics is based on our analyzes of what can be relatively easy to calculate and strongly specific for a conjunction and variables appearing in it. The characteristics are encoded as a unique number having such a property that, any two subjects of encoding having the same code are equivalent based on the properties which are considered in that encoding. The overview of encodings is presented in Tab.1. Due the lack of space and straightforwardness of the process, we do not provide algorithmic description of the encoding. In the next $CODE(subject)$ denotes a given encoding of subject. The uniqueness of the encoding is a key to make unique mapping of variables. If more than one subject is encoded to the same code, it is useless to faster the implication evaluation process. The encodings related to post-conditions of services are made during preprocessing and stored in hash tables (we symbolize them by postfix "HT" after the code

Table 1. Encodings overview.

Code symbol	Encoding subject	Encoded properties
WITH	variable	appearance in predicates concept
WITHOUT	variable	appearance in predicates
CONJ	conjunction	for all variables: appearance in predicates concept
UNQCON	variable	concept

symbol). In these, the key is a code and the value is a subject of encoding. The encodings of goal-conditions can be realized only during composition. During evaluation, we take unique codes related to Str'_{DNF} and check if the hash table storing the corresponding code of Str_{DNF} contains such code. The check is done in $O(1)$ time.

Alg. 1 depicts more details about implication evaluation. For all $str' \in Str'_{DNF}$ we check if $CONJ(str')$ is present in $CONJHT$ storing codes of all $str \in Str_{DNF}$ (lines 2 to 3). If it is, Str_{DNF} contains the same conjunction as the given str'. In this case we only have to map variables and check if the value restrictions are not violated. This is done in linear time regarding the number of variables.

Algorithm 1 Evaluate implication

```
1: for all str' ∈ Str'_DNF do
2:   if CONJ(str') found in CONJHT then
3:     map vars and check value restrictions
4:   for all str ∈ Str_DNF do
5:     for all var in str' having WITH code do
6:       if WITH(var) found in WITHHT then
7:         map vars and check value restriction
8:     for all var in str' having WITHOUT code do
9:       if WITHOUT(var) found in WITHOUTHT
         AND subsume ok then
10:        map vars and check value restriction
11:    for all var in str' having UNQCON code do
12:      if UNQCON(var) found in UNQCONHT AND
         appearance ok then
13:        map vars and check value restriction
14:    while new mapping found do
15:      for all predicate in str' with one mappedvar do
16:        for all suitablepredicate in str do
17:          potentialvar = unmapped in
             suitablepredicate
18:          check appearance
19:        if only one potentialvar with appearance ok then
20:          map vars and check value restriction
```

If there is no equivalent str for any str', we try to find unique mapping between variables. First, we map variables having unique $WITH$ code (lines 4 to 7). For all variable var in str' having a $WITH$ code, we check if $WITHHT$ contains a variable under key $WITH(var)$. If it does, we map var to the value of $WITHHT$ under key $WITH(var)$ and check the value restriction. The mapping based on $WITHOUT$ code is similar to $WITH$ (lines 8 to 10). The difference is that $WITHOUT$ code does not include the concept type of the variable. The benefit is, that we find corresponding variables also in the case that the variable in str is not ground to the same concept as the variable from str', but to concept subsuming it. Of course, in this case we must check if subsumption holds, in contrast to mapping based on $WITH$ code.

For each variable still being unmapped, we check (lines 11 to 13) if there is some variable in str for which it holds that: i) it is ground to the same concept as the given variable in str', ii) no other variable is ground to that concept. These two conditions hold, if there is some variable in the str having the same $UNQCON$ code as the variable from str'. If we find such variable, we have to check if it appears in the same predicates. This is done in linear time regarding the number of predicates where the variable from str' appears. If appearance is as required, value restriction is checked.

Even if we apply all the coding mechanisms to find mapping between variables, some of them may still remain unmapped. For these we start an iterative process. In each iteration, at least one variable should be mapped. If this is not true, there is no unique mapping between the variables. In each iteration (lines 14 to 20), we look for a predicate in str' having one unmapped variable. Then, we find the same predicates in str. We filter them to get only those where the mapped variable is mapped to the mapped variable in str'. All unmapped variables in remaining predicates of str become potential variables for mapping. Note that in most cases there will be only one potential variable. For each potential variable we check its appearance in predicates where the unmapped variable from str' appears. If there is only one potential variable appearing in the required predicates, we found a new unique mapping. Otherwise, we look for an other predicate in str' having only one mapped variable.

4.2 Creating workflow structure

This section explains the process of finding services which are used in the composition to produce data for the final services, if these are not provided in the query. In general, this set in not empty. The empty set refers to a situation that all final services have inputs provided in the user query. Usually, this set consists of several interconnected services.

Our approach is based on two processes, Fig. 1-b). The first selects services which can be used because they have provided inputs. The second selects services which cannot be used because they do not have provided all inputs. The latter is not necessary to find a composition. It is used only

to faster the *select usable* process, which is necessary.

The selection of usable services starts with a selection of the initial services. Analogically to selection of final services, it is done in $O(1)$ time for each input provided in the user query, i.e. in $O(|I'|)$ time at all. Each service having provided inputs is added into list *inputProvidedServices*. This list is the input for a process presented in Alg. 2.

Usable services are selected in a loop. In each iteration we process the first service from *inputProvidedServices*. For this service we check each provider of each input. If the provider is usable and it improves the aggregated QoS of *service*, we update the link to the best provider of the respective input. If for each input we found at least one provider, the $service$ is usable. In this case, we update its aggregated QoS. Then, we traverse all its successors. Each successor which was not processed, or the *service* improves its aggregated QoS is added into *inputProvidedServices* to be processed.

Algorithm 2 Find usable services

```
1:  while inputProvidedServices.size > 0 do
2:      service = inputProvidedServices.removeFirst()
3:      for all input provider of service do
4:          if provider is usable then
5:              update best QoS providers                          A
6:      if all inputs provided then
7:          update aggregated QoS
8:          for all successors of service do
9:              if successor is unprocessed OR service improves successor's aggregated QoS then
10:                 inputProvidedServices.add(successor)           B
```

The problem with Alg. 2 is that it wastes a lot of time by processing services for which it will not find providers for all inputs. To avoid this, we propose a process restricting the set of services which must be processed by Alg. 2. The restriction is based on the idea that several services have at least one input which is not provided by any other service. These services are usable only if those inputs are provided in the user query. We call them user data dependent services. The list of these is created during preprocessing.

The selection of unusable services starts by evaluation which user data dependent services have not provided inputs in the user query. These services are unusable. Moreover, each service having at least one input for which the only provider is an unusable service is unusable too. The selection of these is depicted in Alg. 3.

Alg. 3 starts with a list of services for which it is not evaluated if they are unusable. These services are processed in a loop until there is a chance that new unusable service may be found. This is in the case when some service was selected as unusable, what means that some of its successors may become unusable too. After the overall process finishes, all user data dependent services or their successors, for which some input is not provided, are selected as unusable. There may be still services which are not selected as unusable even if they are unusable. These are those which are not user data dependent, but some of their inputs have no usable providers.

The *select unusable services* process affects the selection of usable services in such a way that the unusable services are not processed, i.e. they are not added into *inputProvidedServices*. It is important that the selection of usable and unusable services may run in parallel.

Algorithm 3 Find unusable services

```
1: unusable found = true
2: while unusable found do
3:     unusable found = false
4:     for all service in undecidedServices do
5:         if some input depends on unusable then
6:             service is unusable
7:             unusable found = true
```

5 Evaluation

The evaluation of our approach is split into evaluation of i) process selecting last services in the workflow taking into account also the pre-/post-conditions, ii) process creating the structure of the composite service. All experiments were realized using a Java implementation of our composition system. The computations were run on a machine with two 64-bit Quad-Core AMD Opteron(tm) 8354 2.2Ghz processors with 64GB RAM.

The experiments were realized using data sets generated by a third party tool used to create data sets for *Web services Challenge 2009* (`http://ws-challenge.georgetown.edu/wsc09/software.html`). We generated test sets consisting from 10 000 to 100 000 services. For each test set, the solution requires at least 50 services to compose. The number of concepts in the ontology is from 30 000 to 190 000. To allow others to evaluate their composition system on same data, we made the test sets available at `http://semco.fiit.stuba.sk/compositiontestsets/`.

5.1 Implication of conditions

The generator tool from WS-Challenge does not create pre-/post-conditions, nor goal condition. Due this, we developed our own condition generator extending the data sets from WS-Challenge. It generates random conditions and value restrictions. It takes as parameters the maximal size of Str and each str of the generated condition. We can generate the test sets in such a way, that the evaluation of

the implication is done based on one of the proposed encodings, or the iterative process. This is because we wanted to evaluate each mechanism separately.

Fig. 2 depicts results of the experiments with conditions of different average sizes of Str and str. The size of Str' is in each case $|Str|/4$ and $|Conds| = 4$. As it is obvious, most of the time is spent by encoding. Our deeper analysis showed that the slowness is caused by the transformation of the goal condition into DNF. After encoding, the evaluation is performed relatively fast. There are minor differences between the applications of different encodings. This is why we depict only the maximum over evaluation times of different encoding application. The evaluation using the iterative search process performs slightly slower than if encoding may be used to map variables. This is in compliance with our theoretical analysis.

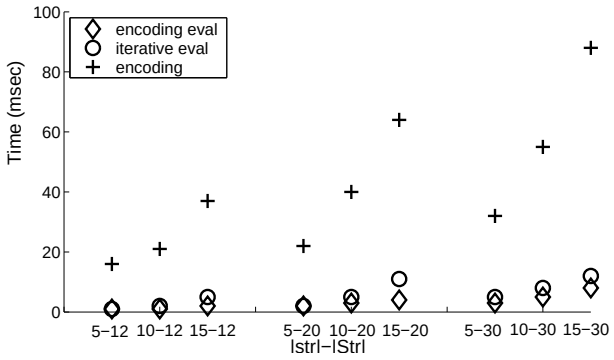

Figure 2. Implication evaluation.

5.2 Workflow design

The aim of evaluating the workflow design phase of the composition is to show i) dramatic improvement of the composition time resulting from the usage of unusable services selection, ii) efficiency and scalability of our composition approach. The results achieved by our composition system are summarized in Tab. 2. The results for composition time present significant improvement against the approach without service space restriction (Without). In both cases, when selection of unusable services runs in parallel (NewP), and in sequence before select usable services process (NewS), the time is reduced in more than one order of magnitude, see also Fig. 3. To clarify the reason of the improvement, we measured also how much times did the execution of *select usable* services process cross the code at lines 5 (marked as A), and 12 (marked as B), see also Fig. 4. The decrease of the crosses explains the improvement of composition time. The parallel version presents an improvement from 15 to 46 times in terms of composition time and adequate improvement in terms of crossing lines A, B, see Fig. 5.

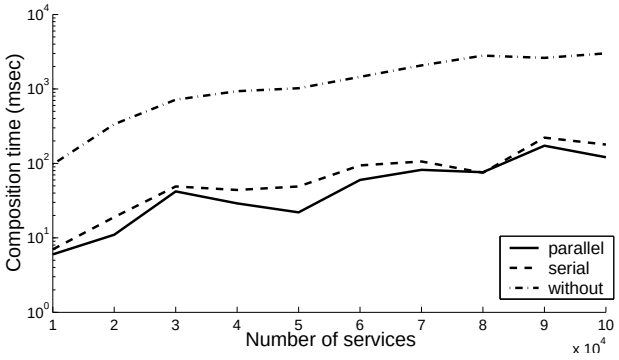

Figure 3. Composition time.

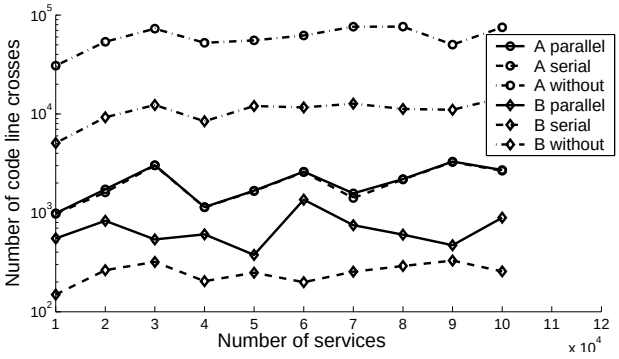

Figure 4. Code line crosses.

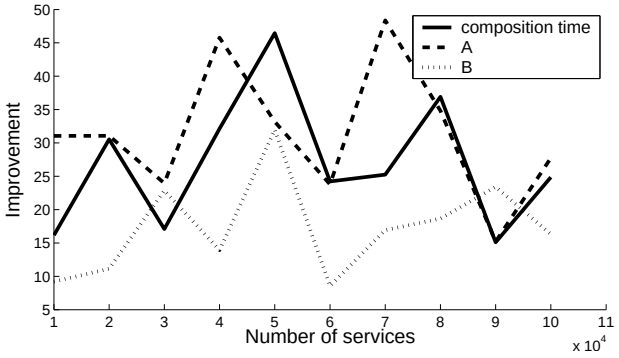

Figure 5. Improvements.

6 Conclusions and future work

This paper presents a composition approach focusing on problems related to efficiency and scalability. At conceptual level, it takes into consideration pre-/post-conditions of services and finds a solution having the best QoS. As far as we know, there is no other work dealing with scalability regarding the complexity of the user goal condition and pre-/post-conditions. Our current approach is able to evaluate implication even if the complexity of the conditions is not trivial.

Table 2. Experimental results.

Services	Composition time (msec)			Number of code line crosses						Improvement		
	NewP	NewS	Without	NewP A	NewS A	Without A	NewP B	NewS B	Without B	CT	A	B
10 000	6	7	97	991	976	30767	552	149	5079	16.2	31.0	9.2
20 000	11	19	336	1728	1611	53686	831	263	9249	30.5	31.1	11.1
30 000	42	49	718	3041	3018	72825	539	319	12325	17.1	23.9	22.9
40 000	29	44	932	1144	1136	52368	606	204	8438	32.1	45.8	13.9
50 000	22	49	1022	1674	1661	55542	376	248	12023	46.5	33.2	32.0
60 000	60	94	1454	2613	2581	62142	1361	199	11645	24.2	23.8	8.6
70 000	82	106	2070	1577	1413	76288	751	254	12713	25.2	48.4	16.9
80 000	76	75	2806	2194	2174	76390	602	290	11230	36.9	34.8	18.7
90 000	173	222	2613	3299	3262	50183	471	329	11025	15.1	15.2	23.4
100 000	121	179	3009	2711	2667	75202	895	256	14589	24.9	27.7	16.3

As our experiments showed, our approach suffers by the dependence on the difficulty to transform a logic formula into DNF. We were speculating and suppose that conditions required by a user in practical scenarios are more like conjunctions of relatively independent sub-formulae (independent in terms of distinct predicates and variables used in distinct sub-formulae). Instead of transforming the condition to DNF, we should use conjunctive normal form (CNF). Our preliminary experiments show that the transformation into CNF is fast if the input is a formula which is a conjunction at the first level. Our idea of encoding application and the algorithm to decide if an implication between two formulae holds, can be relatively easily adapted to work with CNF. Our future research evaluates this in details.

The research regarding QoS aware service composition is noticeably farther as research dealing with pre-/post-conditions. As this paper presents, it is feasible to compose services in acceptable time even if the size of the service repository rises to 100 000. We suppose that the research attention in future might move to other problems than efficiency of QoS aware composition. Since there is still no practical application of service composition, we should analyze what are the real scenarios where it could be applied and which problems have to be solved before it can be done.

We had adapted the proposed approach to effectively handle update requirements regarding the service set. We are able to quickly manage the adding/removal of service, or change of the QoS attributes. As a result, the user query is responded based on the actual situation without significant delay.

References

[1] S. Agarwal and S. Lamparter. User preference based automated selection of web service compositions. In *ICSOC Workshop on Dynamic Web Processes*, pages 1–12, 2005.

[2] M. Alrifai, T. Risse, P. Dolog, and W. Nejdl. A scalable approach for qos-based web service selection. In *Service-Oriented Computing ICSOC 2008 Workshops*, pages 190–199, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] P. Bartalos and M. Bielikova. Fast and scalable semantic web service composition approach considering complex pre/postconditions. In *WSCA '09: Proc. of the 2009 IEEE Congress on Services, Int. Workshop on Web Service Composition and Adaptation*, pages 414–421. IEEE CS, 2009.

[4] P. Bartalos and M. Bielikova. Semantic web service composition framework based on parallel processing. *IEEE Int. Conf. on E-Commerce Technology*, pages 495–498, 2009.

[5] P. Bertoli, R. Kazhamiakin, M. Paolucci, M. Pistore, H. Raik, and M. Wagner. Control flow requirements for automated service composition. *IEEE Int. Conf. on Web Services*, pages 17–24, 2009.

[6] Y. Gamha, N. Bennacer, G. V. Naquet, B. Ayeb, and L. B. Romdhane. A framework for the semantic composition of web services handling user constraints. In *ICWS '08: Proc. of the 2008 IEEE Int. Conf. on Web Services*, pages 228–237. IEEE CS, 2008.

[7] Z. Huang, W. Jiang, S. Hu, and Z. Liu. Effective pruning algorithm for qos-aware service composition. *IEEE Int. Conf. on E-Commerce Technology*, pages 519–522, 2009.

[8] S. Kona, A. Bansal, B. Blake, S. Bleul, and T. Weise. A quality of service-oriented web services challenge. *IEEE Int. Conf. on E-Commerce Technology*, pages 487–490, 2009.

[9] S. Kona, A. Bansal, B. Blake, and G. Gupta. Generalized semantics-based service composition. In *ICWS '08: Proc. of the 2008 IEEE Int. Conf. on Web Services*, pages 219–227. IEEE CS, 2008.

[10] S. Kona, A. Bansal, L. Simon, A. Mallya, G. Gupta, and T. D. Hite. Usdl: A service-semantics description language for automatic service discovery and composition. *Int. J. Web Service Research*, 6(1):20–48, 2009.

[11] N. Lin, U. Kuter, and E. Sirin. Web service composition with user preferences. In *ESWC*, pages 629–643, 2008.

[12] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of ACM*, 23(1):31–42, 1976.

[13] H. Q. Yu and S. Reiff-Marganiec. A backwards composition context based service selection approach for service composition. *IEEE Int. Conf. on Services Computing*, pages 419–426, 2009.

[14] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM.