Fast and Scalable Semantic Web Service Composition Approach Considering Complex 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

The process of semantic web service composition arranges several web services into one composite service to realize complex workflows with an exploitation of semantics. This paper proposes an approach to automatic semantic web service composition. Its advantage is good scalability regarding the complexity of user constraints and pre/postconditions. Based on these conditions it propagates the value restrictions constraint from the user goal through the overall composite service. The resulting plan depicts all the possible branches of the workflow leading to a goal. This includes the automatic generation of the conditions deciding which branch will be chosen during the execution. Finally, our approach exploits available data which can be used as input parameters for web services. If these are not offered, it searches for a web service producing them.

1 Introduction

Semantic web services present a topical research area aimed at exploiting semantic annotation of web service descriptions [14]. One of the most studied topic in this area is the automation of the semantic web service composition aiming at arranging several services into one complex service to be able to realize more complicated workflows. The composition is studied in different contexts to achieve practical applicability [5, 9].

Even though a lot of work has been done in the field of semantic web service composition, there still remain problems. These are mainly related to insufficient automation of composition and execution, and scalability of approaches. Our approach to semantic web service composition deals with a semantic description of pre/postconditions of web services and a goal we intend to achieve using the service composition. The main contribution is related to the composition algorithm taking into consideration complex pre/postconditions. It finds all possible branches of the workflow leading to the specified user goal. The conditions affecting which branch will be taken during the execution are created automatically. We perform a huge amount of computation in the preprocessing phase, which allows for fast response to user queries. Moreover, already during composition, we automatically identify data to be presumably used as inputs during execution. To do this, data available in the operating environment (dedicated data repository) or values from the given goal are used. This way we prepare the workflow for execution. In the whole process we exploit the semantics related to the data.

Our work in this area is motivated by the *SEMCO-WS* national research project [10]. Its aim is to expand and refine several approaches developed in project Knowledge Based Workflow System for Grid Applications – *K-Wf Grid* (`http://www.kwfgrid.eu/`). The aim of this European project was the automated composition of workflows of grid services using semantic support and comfortable user interfaces for complex grid middleware.

The main goal of *SEMCO-WS* considering this paper is improving the composition and execution of workflows. This includes the selection of inputs for web services. The overall architecture of the developed system includes a repository of semantically annotated data, which may be used as inputs. Our approach is based on performing semantic querying over this repository to find suitable data for composition.

The domain of *SEMCO-WS* is crisis management. Our pilot application refines the architecture of an existing, complex commercially used crisis management and decision support system for the radiological, environmental, hydrological as well as seismological emergencies. The system allows users to simulate fictive emergency scenarios at the moment of the accident or during the long-term actions implemented months or years after an accident.

*This work was partially supported by the Slovak Research and Development Agency under the contract No. APVV-0391-06, "Semantic Composition of Web and Grid Services".

2 Related Work

Several methods used in semantic web service composition based mostly on AI planning techniques were recently proposed. These include state space search, graph based planning, HTN (Hierarchical Task Network) planning, approaches based on logical programming and others [11]. A lot of approaches also employ pre/postconditions and consider them during service chaining. There exist several formalisms to describe them. The most known are OWL-S `http://www.daml.org/services/owl-s/1.0/owl-s.html`, WSDL-S `http://www.w3.org/Submission/WSDL-S/` and WSML `http://www.wsmo.org/wsml/`. They differ in the complexity and expressivity of their construction elements. In each case the semantic annotation binds the elements of the web service to domain terms described by ontologies. From all the elements it focuses on the I/O of the web services and the conditions under which the web service can be invoked and the conditions which will hold after its execution. OWL-S provides a dedicated part in its structure for the description of pre/postconditions (*process:hasPrecondition* for precondition and *process:hasEffect* for postcondition), yet does not specify how to describe them specifically and allows different languages to be used.

In [6], the authors present a method working with the OWL-S service descriptions. The pre/postconditions are defined as a list of predicates which hold before/after execution. There is an implicit conjunction between them, i.e. all the predicates must hold. The predicates cannot be combined using other logical operations such as disjunction. The composition is based on forward chaining and is speeded up using heuristics based on relaxed graph. From the description of the pre/postconditions and the application of relaxed graph based heuristics, the approach presented in [13] seems to be similar to [6]. From the papers, it is not clear how these approaches scale when a huge number of services is used, e.g. thousands.

From the scalability point of view, more promising results are presented in [8, 12]. Similarly to our approach they perform composition in insignificant time (in msecs) even if the number of services rises to thousands. Both of them realize preprocessing to decrease the computation required during composition. In the approach proposed in [8] the USDL language is used to specify the formal semantics of services [1]. This OWL based language uses WordNet as a common basis for understanding the meaning of services. If we neglect the fact that in our approach we can also use existential and universal quantification in the formulae describing pre/postconditions, USDL has the same expressivity of conditions. In [7] the authors extend the notion of composition presented in [8] to handle non-sequential conditional composition. Our workflow representation uses a similar notion. The approach described in [12] does not consider any pre/postconditions. It chains services based only on semantic matching of I/O considering also subsumption.

In [4] the authors present a composition approach handling so called user constraints. These represent value restriction contraints to input, output or local parameters of services using the KIF language. To bind the constraints to service parameters the OWL-S extension is used. For us, it is not clear why the authors do not use the *process:hasPrecondition* and *process:hasEffect* parts of the OWL-S. Their approach seems to be equivalent to the usage of these parts of the language. Our approach based on OWL-S offers the same possibility to express the user constraints and the pre/postconditions of web services.

3 Approach Overview

Our approach is based on using semantics throughout the whole process of service composition and brings the following aspects:

- **Complex pre/postconditions.** We propose a solution to complex web services pre/postcondition description based on existing standards and approaches. It allows expressing sufficiently complicated statements by combining predicates with first order logic operators.
- **User goal description.** The definition of the goal is represented as a pair: concept type and constraint. The concept type is a concept from the ontology used for semantic annotation of web services. The constraint is defined as a first order logic formula, similarly to the pre/postconditions. The predicate's arguments in the formulae may be restricted to concrete values.
- **Value restriction propagation.** We propose an approach propagating the value restrictions from the goal through the chained web services in the plan to the inputs. As a result, the corresponding input parameters are bound to concrete values without the need of other specification (more details in section 4.1).
- **Automatic decision on alternatives of the composed plan.** We employ decision blocks to define the alternatives of the composed plan. The conditions serve for the respective branch selection during the execution of the workflow and are defined in an automatic manner.
- **Data awareness.** The composition algorithm takes the available data into consideration. If the available data satisfy the given constraint, they are used as input parameters for the respective web services. Otherwise the planning continues by searching for a web service providing the required data.

3.1 Complex pre- and post-conditions

To depict the semantics of web services, we use OWL-S. As there is no common consensus which language should be used, we present our proposition to complex pre/postcondition description in OWL-S. It is based on SWRL (Semantic Web Rule Language, `http://www.w3.org/Submission/SWRL/`) and its extension. We use its atomic statement to express simple conditions (unary and binary predicates). At the semantic level they are modeled as properties between OWL variables (of defined type – concept), individuals or data values. The variables represent I/O parameters or local variables.

OWL-S allows creating any number of pre/postconditions for one web service. The interpretation is that each precondition must hold before execution and each postcondition will hold after execution, i.e. there is an implicit conjunction. The creation of more complex statements is required to increase the expressivity of web service pre/postcondition annotation. We use a proposition for SWRL extension to first order logic (`http://www.daml.org/2004/11/fol/proposal`) for this purpose. It allows creating more complex formulae using negation, conjunction, disjunction, etc.

Figure 1 depicts an example of two pairs of services which can be chained because they have compatible I/O and pre/postconditions. Services S_1, S_2 have the same output of type *Model* at the semantic level, which is consumed by service S_3. The *Model* represents a meteorological model of a region, during a certain time interval, given as input to services S_1, S_2. Based on it, service S_3 computes which areas of the region, to which the model corresponds to, are endangered by a storm.

The pre/postcondition of service S_3 expresses that the region corresponding to the model given as input and the region corresponding to the resulting alert are the same. It also expresses that the model must have high precision. Hence, if we search for a web service providing data to service *S3*, we require a service resulting in the *Model* and satisfying the condition that the model corresponds to a certain region and has high precision.

Service *S1* obviously satisfies this condition. The mapping of the symbols in the pre/postcondition is the following. The *O* of S_1 is mapped to *I* of S_3 and the *R* of S_1 is mapped to the *R* of S_3. Now consider service S_2. Its postcondition expresses that its output model corresponds to a certain region and is of low or high precision (imagine that the precision depends on the region). Hence, it also satisfies the precondition of S_3 in the case that it computes a model with high precision.

Now we formalize when the postcondition of one service satisfies the precondition of another service, i.e. when two services are compatible and can be chained. To simplify the explanation, we express the predicates of the conditions as follows: *region* by symbol *R*, *hasPrecision* by symbol *PL* if the second argument is *lowPrecision*, *hasPrecision* by symbol *PH* if the second argument is *highPrecision*.

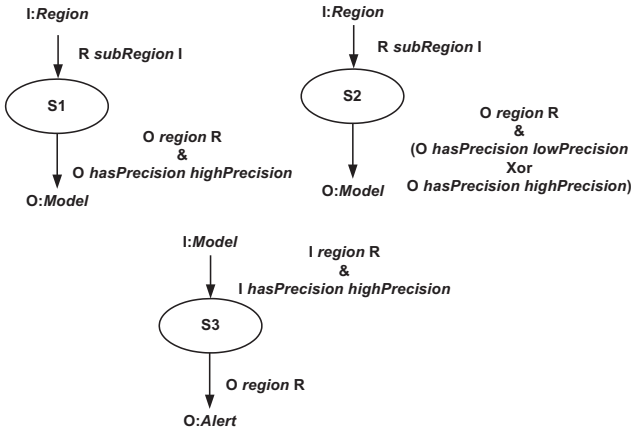

Figure 1. Web service chaining example.

The fact that two web services are compatible and can be chained means that there exists such a combination of predicate values in the pre/postcondition formulae that they are evaluated as true (we consider the same values for the mapped symbols in pre/postcondition). Specifically, we must check each predicate value combination where the postcondition formula is true and check if for some combination also the precondition formula is true. Additionally, the ancestor service's output must subsume the successor service's input from the semantic type point of view.

From the planning point of view we can chain two services if they are compatible. Reasoning is required to evaluate web service compatibility and can be performed by evaluating whether the postcondition formula implies the precondition formula. We consider only those predicate value combinations where the postcondition is true. Let us denote $Pre(S)$ the precondition formula of service S (analogically postcondition). The question if two services S_{anc} (ancestor service) and S_{succ} (successor service) can be chained is easy to answer using truth tables of $Pre(S_{succ})$, $Post(S_{anc})$, and $Post(S_{anc}) \Rightarrow Pre(S_{succ})$. Taking services S_1 and S_3 we check the combination *R*, *PH* which is the only one where $Post(S_1)$ is true, see Table 1. This table holds also for $Pre(S_3)$ because they are the same. When deciding if S_1 and S_3 are compatible, we evaluate $Post(S_1) \Rightarrow Pre(S_3)$ which is $R \wedge PH \Rightarrow R \wedge PH$, see Table 3 (unconsidered predicate value combinations are omitted). It contains only one true value and no false values. This means that these services are strongly compatible, i.e. the ancestor service results in a postcondition satisfying the successor service's precondition in each case.

If the postcondition formula contains also symbols

Table 1. $Post(S_1)$, $Pre(S_3)$

	R	**¬ R**
PH	1	0
¬ PH	0	0

Table 2. $Post(S_2)$

		R	**¬R**
PL	**PH**	0	0
PL	**¬ PH**	1	0
¬ PL	**PH**	1	0
¬ PL	**¬ PH**	0	0

Table 3. $Post(S_1) \Rightarrow Pre(S_3)$

	R	**¬ R**
PH	1	
¬ PH		

Table 4. $Post(S_2) \Rightarrow Pre(S_3)$

		R	**¬ R**
PL	**PH**		
PL	**¬ PH**	0	
¬ PL	**PH**	1	
¬ PL	**¬ PH**		

(predicates) which are not contained in the precondition, we have to consider each extended expression. In our example this means that for service S_2 and S_3 we have to consider also *R*, *PH*, *PL* and *R*, *PH*, *¬PL*, however the interesting symbols are *R*, *PH*, see Table 2 depicting $Post(S_2)$. Table 4 depicts $Post(S_2) \Rightarrow Pre(S_3)$ which is $R \wedge ((\neg PL \wedge PH) \vee (PL \wedge \neg PH)) \Rightarrow R \wedge PH$. It contains one true value and one false value. This denotes that the services are weakly compatible. The case when service S_2 can provide data for service S_3 is if the execution of S_2 results in the condition *R*, *PH*, *¬PL*, i.e. not in each case.

Now we generalize the explanation of the compatibility of two services S_{anc} and S_{succ}. Let denote the set of formulae representing the respective conjunctions of the pre/postcondition DNF of service S as $PreC(S)$ and $PostC(S)$. The three cases of service compatibility are:

1) Services S_{anc} and S_{succ} are not compatible if:
$\forall c_{pre} \in PreC(S_{succ}), \nexists c_{post} \in PostC(S_{anc}) :$
$c_{post} \Rightarrow c_{pre}$ is true.

2) Services S_{anc} and S_{succ} are weakly compatible if:
$\exists c_{pre} \in PreC(S_{succ}), \exists c_{post} \in PostC(S_{anc}) :$
$c_{post} \Rightarrow c_{pre}$ is true.

3) Services S_{anc} and S_{succ} are strongly compatible if:
$\forall c_{pre} \in PreC(S_{succ}), \exists c_{post} \in PostC(S_{anc}) :$
$c_{post} \Rightarrow c_{pre}$ is true.

3.2 Creating representation of the planning domain for fast composition

Our approach to fast web service composition is based on preprocessing. During it we create a data structure used to quickly answer two queries. The first query (Q_1) serves to select services satisfying a given user goal. The second query (Q_2) selects services providing data required as input for a given service. These two queries are used to perform fast web service composition satisfying pre/postconditions. In the following, we present the used data structure and the process populating this data structure with data considering the given web services set.

Our data structure is based on a relational repository having two parts, see a scheme in Figure 2. The first stores data about the pre/postconditions (*Part 1a*) and I/O (*Part 1b*) of services. After *Part 1* is populated, we query it to evaluate which services can be chained. The results are stored in *Part 2*. During realization of Q_1, *Part 1* is used. To answer Q_2 only *Part 2* is employed, which also contains data about the mapping between symbols in pre/postconditions of the chained services.

In the previous section we mentioned that to decide if two services can be chained, reasoning is employed. Our approach realizes this reasoning using a programmed function and the querying mechanism of relational databases. Before the pre/postconditions are stored in our data structure, we transform them into disjunctive normal form (DNF). They are stored in the relational database in tables *Predicate, Conjunction, Condition*.

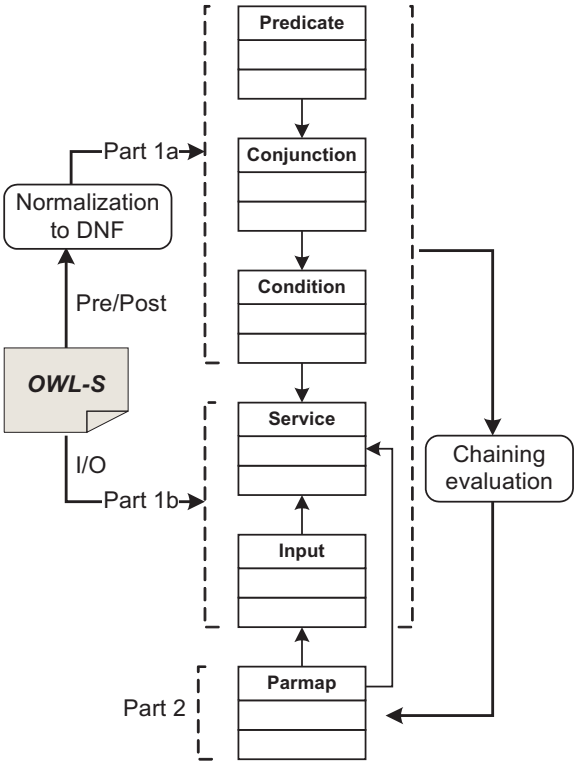

Figure 2. Data structure creation.

From the precondition point of view, it is important that it is enough to satisfy one conjunction from the all from which the DNF representing the precondition consists of. This is because if one (or more) conjunctions in DNF are evaluated as true, the overall DNF is true. Hence, the respective conjunctions represent alternatives, from which it is enough to satisfy one, to fulfill the precondition.

From the postcondition point of view, the conjunctions from which the DNF representing it consists of, correspond to the respective cases in which the execution of the web services can result in. In other words, after the service is executed, at least one conjunction is evaluated as true.

To decide whether two services are compatible, we need to find at least one conjunction in the precondition DNF, which is covered by at least one conjunction of the postcondition DNF. Covered means that there exists a mapping between the symbols of the conjunctions that the conjunction from the postcondition implies the conjunction from the precondition. If for each conjunction of the precondition there is no covering conjunction of the postcondition, the services are not compatible. If each conjunction of the postcondition covers at least one conjunction of the precondition, the services are strongly compatible. If at least one conjunction of the postcondition covers at least one conjunction of the precondition, the services are weakly compatible.

The decision process whether two services can be chained is realized as follows. For each input parameter of each web service (stored in table *Input*), we search for services providing the required data (output parameter is stored within table *Service*) and satisfies the precondition. The compatibility of two services is caught in table *Parmap*. It stores all possible input and output parameter pairs, and the mapping of symbols in pre/postcondition of the corresponding services. This is exploited during composition to quickly propagate the value restrictions. Table *Parmap* contains compatible I/O pairs considering also subsumption between them.

To get the list of compatible services for the respective input parameter, we perform the query Q_1 over *Part 1* of the relational database. The query is dynamically constructed based on the given conjunction of the precondition. It exploits the logical AND operator and the capabilities of the relational database systems' to evaluate logical expressions. Because we already work with normalized formulae and at this point only with conjunction, we do not need any other operator. The overall reasoning required to decide if two services are compatible based on their pre/postcondition is moved to a programmed function and the relational database querying system. We do not employ any specific logical reasoner.

The overall filling of the database is realized as preprocessing. Our aim is to move all possible computation required to compose web services into preprocessing. Based on the presented data structure stored in the relational database we realize fast web service composition. One important property of our data structure is that if a new web service is available, the information about it can be added incrementally and no complete rebuilding of the database is required, which is also true in the case of web service removal.

4 Composition algorithm

In this section we present our two phase web service composition algorithm. First, query Q_1 is used to select services satisfying the given goal. In the second phase, we perform composition based on back chaining for each service with unsatisfied inputs. Here we would again use Q_1 because there is no difference between searching for services resulting in the user goal or a required input data with the corresponding constraint. The only difference is that we cannot search for services satisfying the user goal before we know it, i.e. we cannot preprocess this. In the case of input parameters, we performed preprocessing resulting in *Part 2* of our data structure. Hence, we do not realize the demanding, dynamically constructed Q_1 but we only select its results using fast Q_2 with static structure.

The main idea of the back chaining (second phase of composition) is depicted in Algorithm 1. For each input parameter and corresponding constraint it first checks if there is a value restriction for it arising from the user constraint. If not, the available data repository is checked whether it contains the required data. If not, we search for web services satisfying the input and its constraint, using Q_2. After the proper services are selected, we check if they are already used in the composition. If this is true and also the constraint related to that service call suits the current constraint, we chain these services. This leads to parallelism or choice between the current service and the service for which the service call was planned in some previous planning step. Otherwise we plan a new call of that service.

From the performance point of view, it is important that the demanding Q_1 is realized during the composition only once, in first phase. During the service chaining, we realize only the fast Q_2. Hence, we speed up the process.

4.1 Value restriction propagation

This section explains how we realize value restriction propagation from the user goal to the services used in the composition. After the services resulting in the user goal are selected, we also examine the mapping between the symbols in the user constraint and the postcondition formula. All the value restrictions in the user goal constraint are propagated to the respective service. If we restricted some pred-

Algorithm 1 SatisfyInputs: Input: *service, constraint:*

```
for all input of service do
  if isValueRestricted(input, constraint) then
    continue;
  else if isDataInRepository(input, constraint) then
    bindDataToInput();
  else if S ← getSatisfyingServices(input, constraint) then
    for all satisfyingService in S do
      if plannedService ← isAlreadyPlanned(satisfyingService)
      then
        if suitConstraints(plannedService.constraint, constraint)
        then
          chain(service, usedService);
        end if
      else
        newService ← satisfyingService;
        newConstraint ← propagateConstraint(newService, constraint);
        chain(service, newService);
      end if
    end for
  end if
end for
```

icate's argument in the user constraint to a specific value, based on the symbol mapping we also restrict the corresponding argument in the postcondition. If such a symbol appears also in the precondition of the service, the propagation is repeated also when other services are chained with it. If some symbol in the successor service's precondition formula is restricted to some value, we restrict also the corresponding symbol in the ancestor service's postcondition formula. The mapping between the symbols in the pre/postcondition of the chained services is retrieved from table *Parmap*.

If we restrict to some value a symbol representing some input parameter, we do not search for another source providing the corresponding data and continue by satisfying other inputs. During the workflow execution the restricted inputs are set to the value arising from the restriction. An example of service chaining and value restriction propagation is provided in section 5.2.

4.2 Alternative and parallel branches in the composition

During composition we may be in need of planing a service that already is used in the plan. The most obvious case is when we have two or more services which are equivalent, i.e. they have the same functionality and hence also the semantic description. The difference may be for example in their QoS. If we look for the input data for some services and realize that there is already another service in the plan providing the required data, we chain the services together. During workflow execution, the output will be consumed in parallel or alternatively by these services.

The pre/postconditions of the chained services decide if the connection between the ancestor and successor services will be parallel or alternative. If two or more services connect to the same service and they satisfy the same conjunction of the ancestor service's postcondition, there is a parallel between them. If they satisfy different conjunctions, there is an alternative between them. In this case a decision block is created. The condition controlling the decision is created automatically. This is done based on the matching conjunctions of the pre/postcondition. The execution engine decides to continue the execution taking such a branch which fits the condition that was the result of the ancestor service invocation (note that at this moment we already know the values of the predicates in the postcondition, i.e. we know which conjunctions resulted in true value). If there are more than one fitting branches, other rules are applied. During workflow creation we can set priorities for each branch and the execution engine takes the branch with the highest priority. Alternatively, we ask a user to decide which branch to take. The involvement of the user in the composition process is discussed in our previous work [2].

5 Evaluation

5.1 Performance evaluation

To evaluate the performance of our approach, we have implemented a test service generator. Using it, we create a user goal and a set of services which are composed to satisfy it. The services are generated in such a way that there is only one composition leading to the goal, i.e. one workflow is possible. Each service is used in the workflow exactly once. The generation is based on the following parameters: web service count (*WS*), mean input parameter count (*MIN*), disjunction size (*DS*), conjunction size (*CS*), choice and decision count (*CDC*). *DS* and *CS* specify the character of the user constraints and pre/postconditions, normalized to DNF. *DS* expresses the size of the DNF; *DC* expresses the size of each respective conjunction of the DNF. Hence, the number of predicates (*NP*) in the DNF is $DS \times CS$. We employ random generators to create workflows of varying structure.

In our experiments we used workflows consisting from 100 to 200 000 services and the DNF with 4 to 250 predicates. From the experiments the following conclusions can be made. The composition time is independent on the following parameters: *MIN*, *CDC*, and the number of values restricted in DNF. The composition time scales linearly depending on the following parameters: *WS*, *DS*, and *CS*. This is in compliance with the results of our theoretical analyses of the composition algorithm's complexity.

Due the space constraints, we present only a representative sample of all our results. The computations for *WS*

from 1 000 to 10 000 were realized on an Intel T5870 2GHz Dual core, 2GB RAM with 256MB memory limitation for JVM. The computations for higher *WS* were realized on an Athlon 64 X2 4200+ 2.2GHz Dual core, 8GB RAM with 1.5GB memory limitation for JVM. For each workflow parameter combination, we generated from 200 to 1 000 workflows. Hence, each result is the average mean of at least 200 runs. The graph in Figure 3 presents results for workflows consisting from 1 000, 10 000 and 100 000 services. The *x*-axis specifies the values of *DS* and *CS*. The *y*-axis represents composition time which is normalized to fit in the graph – the composition time for *WS* = 10 000 is divided by 10 and for *WS* = 100 000 by 100 (this way we get relative values related to *WS* = 1 000). As we can see from Figure 3, the composition time is more affected by *NP* for lower *WS*. This is because during composition we always perform one query selecting the services resulting in a goal (this is independent on *WS* and affected only by *NP*). After this, we realize service chaining strongly depending on *WS*. Hence, as *WS* increases the *NP* becomes less influential. From the same reason, the relative composition times from Figure 3 are lower for higher *WS*.

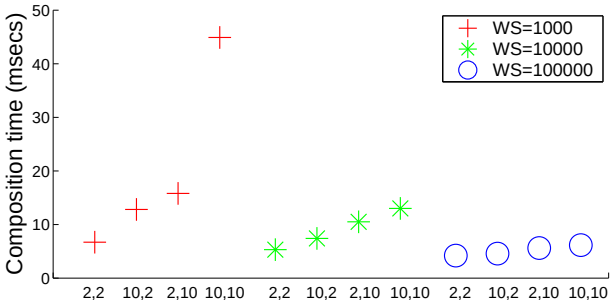

Figure 3. Experimental results.

5.2 Usefulness evaluation

In the following we concentrate on the evaluation of our approach from the usefulness point of view. In the context of project *SEMCO-WS* we are developing a multi-user workflow editor [3]. Its aim is to provide a tool supporting user collaboration during the creation and execution of workflows. At the client side, different users are allowed to manually edit the same workflow, see Figure 4. We integrated our web service composition tool into the editor to support automation of the editing process.

Here we provide an example of workflow from the crisis management domain which is composed automatically by our tool. In this use case we use web services developed in project *SEMCO-WS* and show how value restriction propagation is useful. The goal of our workflow is to get an alert for a specified region and term, in the context of a simulation of a contamination by a certain released substance. The goal is defined like this: Concept Type=*Alert*, Constraint= $(region(alert, 'Slovakia') \wedge simulationTermFrom(alert, '05/01/2009') \wedge simulationTermTo(alert, '10/01/2009')$.

Our automatic composition tool created a workflow shown in Figure 4. It contains services providing different geographical or meteorological data (*Landuse*, *DEM*, *Meteo*). These are inputs for a dispersion model service simulating the release of a given substance in the specified conditions (*Dispersion*). The conditions are provided as a *Scenario*. This is provided using a human task (activity in the workflow realized by human – provision of data in this case). Based on the dispersion model the dose rates are computed on the surface and atmosphere (*DoseRate*). Based on this, the alert service provides a list of cities where the dose rates exceed a safe limit (*Alert*).

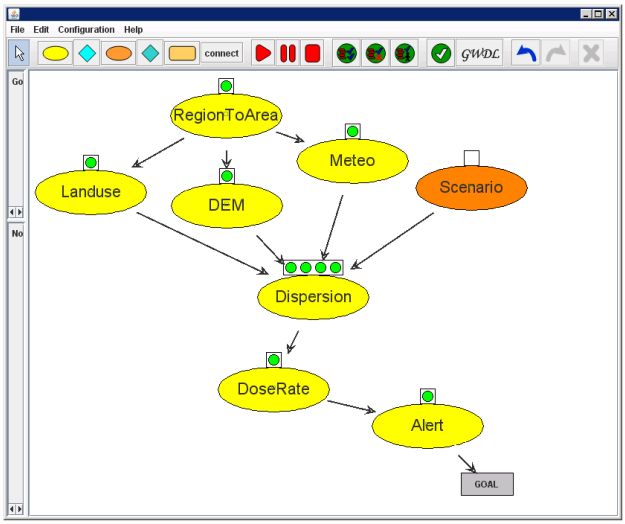

Figure 4. Multiuser workflow editor.

Services *Landuse*, *DEM*, *Meteo* require input data of type *Area*. This data specifies the area using the left lower and right upper corner of a covering rectangle. The corners are defined by latitude and longitude. The pre/postconditions of the services are designed in such a way, that this area corresponds to the region for which the alert is computed. The region defines the names of cities, districts, etc. This is contained as the value restriction in the goal (*Slovakia*). Based on the pre/postconditions, our value restriction propagation mechanism assigns the value *Slovakia* to the input of the *RegionToArea* conversion service. Hence, there is no need nor possibility to define this data in another way. Since the service has no other inputs, the corresponding branch of the workflow is complete and is ready for execution. The same mechanism sets also the input parameters for the *Meteo* service defining the term for

which the weather forecast is to be computed (*05/01/2009* and *10/01/2009*).

The output of the *RegionToArea* service is consumed by three services. This is possible because of the satisfied value restriction. After the execution of this service was planned for the first time, the value restriction constraint was set in such a way, that the input parameter's value is *Slovakia*. When the execution was planned for the second time, the composition approach examined that the execution of the service was already planned. Hence, it checks if the set value restrictions constraint satisfies also the restrictions propagated through the second branch. In the example, this was satisfied and the output of the *RegionToArea* service execution was bound to input the second time. This situation repeated also in the case of the third service. If the value restrictions constraint was not satisfied, a new invocation of the *RegionToArea* service would be planned.

6 Conclusions and Future Work

We have presented an approach for web service composition aimed at creating workflows satisfying a given goal, which is defined using concept type and constraint described as first order logic formula. Based on the goal our approach creates a plan resulting in the corresponding data and the specified constraint. The plan contains all the possible alternative branches leading to the user goal. The condition determining which branch will be taken is evaluated during execution.

The composition algorithm performs until each sub-goal is satisfied, i.e. there is no web service which has not specified the source for its input data. The source can be defined as the output of web service invocation or as data available in the operating environment. It is not necessary to combine these two possibilities. During the composition, the value restrictions defined in the goal are back propagated based on the pre/postconditions of chained services. Our experiments showed good scalability of the planning approach with respect to the number of used service invocations, goal constraint and pre/postcondition formulae complexity.

Our future work deals with the problem of integrating service composition into applications, from software engineering point of view. We will study how service composition can be effectively applied to automate business process creation. This includes also the examination of the effects to the software architecture of the application.

References

[1] A. Bansal, S. Kona, L. Simon, and T. D. Hite. A universal service-semantics description language. In *ECOWS '05: Proc. of the Third European Conference on Web Services*, pages 214–225, Washington, DC, USA, 2005. IEEE CS.

[2] P. Bartalos and M. Bielikova. Enhancing semantic web services composition with user interaction. In *SCC '08: Proc. of the 2008 IEEE International Conference on Services Computing*, pages 503–506, Washington, DC, USA, 2008. IEEE CS.

[3] P. Bartalos, I. Kapustik, and V. Rozinajova. Visual support of workflow composition involving collaboration. In *GCCP '08: Proc. of the 2008 International Workshop on Grid Computing for Complex Problems*, pages 120–127, Bratislava, 2008. Institute of Informatics SAS.

[4] 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 International Conference on Web Services*, pages 228–237, Washington, DC, USA, 2008. IEEE CS.

[5] L. Hluchy, O. Habala, M. Babik, M. Laclavik, Z. Balogh, and E. Gatial. Knowledge-based platform for environmental risk management. In *ISPDC '07: Proc. of the Sixth International Symposium on Parallel and Distributed Computing*, pages 3–9, Washington, DC, USA, 2007.

[6] M. Klusch, A. Gerber, and M. Schmidt. Semantic web service composition planning with owls-xplan. In *AAAI Fall Symposium on Semantic Web and Agents*, Arlington VA, USA, 2005. AAAI Press.

[7] S. Kona, A. Bansal, M. B. Blake, and G. Gupta. Generalized semantics-based service composition. In *ICWS '08: Proc. of the 2008 IEEE International Conference on Web Services*, pages 219–227, Washington, DC, USA, 2008. IEEE CS.

[8] S. Kona, A. Bansal, and G. Gupta. Automatic composition of semanticweb services. *Web Services, IEEE International Conference on Web Services*, 0:150–158, 2007.

[9] J. Paralic and M. Paralic. Some approaches to text mining and their potential for semantic web applications. *Information and Organizational Sciences*, 31(1):157–170, 2007.

[10] M. Paralic, O. Habala, J. Paralic, and P. Bartalos. Semantic composition of web and grid services. In *Znalosti 2009*, pages 355–358, 2009.

[11] J. Peer. *Web Service Composition as AI Planning – a Survey*. University of St.Gallen, 2005.

[12] K. Ren, X. Liu, J. Chen, N. Xiao, J. Song, and W. Zhang. A qsql-based efficient planning algorithm for fully-automated service composition in dynamic service environments. In *SCC '08: Proc. of the 2008 IEEE International Conference on Services Computing*, pages 301–308, Washington, DC, USA, 2008. IEEE CS.

[13] A. Sirbu and J. Hoffmann. Towards scalable web service composition with partial matches. In *ICWS '08: Proc. of the 2008 IEEE International Conference on Web Services*, pages 29–36, Washington, DC, USA, 2008. IEEE CS.

[14] R. Studer, S. Grimm, and A. Abecker. *Semantic Web Services: Concepts, Technologies, and Applications*. Springer, 2007.