# A Multi-Objective Evolutionary Algorithm to Optimize the Dynamic Composition of Semantic Web Services

Brahim Batouche[a,b] Yannick Naudet[a] Isabelle Jars[a] Thibaud Latour[a] Frédéric Guinand[b]

[a] Public Research Center Henri Tudor, Luxembourg
[b] University of Le Havre, France

*Abstract*

Optimization of dynamic composition of web service is helpful to answer user-request in a dynamic environment. However, it addresses multiple problems such as heterogeneity of service descriptions, request interpretation and decomposition, automatic service discovery. To keep the robust and best compositions of Web services, we consider them in a resolution process which uses the Multi-Objective Evolutionary Algorithm (MOEA). This paper presents a MOEA which aims to solve not only the problem of dynamic composition optimization but also to ensures optimum results even if the system undergoes dynamic changes in terms of data and service availability over execution time.

## 1 Introduction

Composition of web services is an efficient way to achieve user goal by creating an *ad hoc* service with a selection and sequencing of available elementary services. The composite service obtained as a result of the request bears some dependencies with the nature, characteristics and availability of elementary (atomic or composite) services. This requires a highly adaptive service composition system enable to dynamically create, invoke, and maintain the resulting composite service.
In most cases, many candidates composite services are consider as an answer to a given request. The set of candidates depends on the availability of a large number of existing services resulting from previous compositions. The existing services are characterized by different functional and non-functional parameters, such as, price, response time, availability, reliability. These parameters are used to match user constraints and objectives during the composition process. Considering non-functional aspects of the elementary services extends the scope of the target constraints leading the service composition. It considers not only the problem of finding one composition that fulfill the global user's functional requirement, but also the difficulty of finding the best selection of the most suitable ones in non-functional terms. Finding the most suitable dynamic composition of web services is equivalent to an optimization of the dynamic composition.

Optimizing the dynamic composition of web services addresses multiple problems like, heterogeneity of service descriptions, request interpretation and decomposition, automatic service discovery, composition modeling, and composition optimization. Adding semantics to web service descriptions is one possibility to enable or to facilitate (at least theoretically) automatic service discovery and composition. From this perspective, semantic web service languages like, OWL-S [1] provides elements to tackle service composition problems.

The purpose of this paper is to present a method to approach optimal composition of web services in a dynamic environment, in response to a user request expressed in both functional and non-functional objectives. The method ensures that the result remains optimum even if the system undergoes dynamic changes in terms of data and service availability over execution time.

The paper is organized as follows. First, it identifies the related worked and tries to underline lacks in this domain. Second, it describes the problem formalization used in this work. Third, the resolution approach is presented. It explained how to find automatically a set of composite services, and how to use a Multi Objective Evolutionary algorithm (MOEA). Fourth, the event processing is shortly analyzed. Finally, section 6 concludes on this work and discusses our future experimentations.

## 2 Related Works

In a previous paper [2], we have presented how to use the MOEA to optimize static compositions of web services. In this contribution, we focus on the dynamic aspect of the composition, *i.e.*, when atomic service conditions and status (availability) change over time at execution time. There is quite little literature describing the use of Genetic Algorithms in the context of web service composition. Main existing works focus on follows resolution points: the automatic composition of web services and the optimization of the web service composition. These resolution points can combine, i.e. made together, in the case where the optimization problem is mono-objectives [3][4].

Literature in the domain of the automatic composition of web services is quite important [4][5][6][7]. More specific to the Semantic Web service standards, Matthias and Andreas present an architecture for automatic web service composition, allowing fast and flexible composition of OWL-S services [5]. They propose interesting ideas to plan the composite service and to automate the service execution. However, the result provided by their system cannot be used as a search space for the optimization, because the search space contains the information about the input/output instances of the service, such as, in a travelling case, the departing city of the transportation, the price, etc.

The automatic composition algorithm proposed by Silva and Sinderen generates the composition graph matching a request [6]. The request is presented by the input/output (I/O) of the requested global composite service. The algorithm compares this I/O with the I/O of the existing atomic or previously composed services. Then, it either selects the service which matches the request I/O, or it checks the service which matches the request output only. If a matching service is found, it is selected and a node presenting this functionality is created. Finally, the algorithm looks for a service such as service input matches the request input and output matches the selected service input, and so on. The algorithm ends when it finds the service matching the request I/O or when all the nodes of the graph are covered, *i.e.*, when a set of sub-services are covered. The obtained composite graph does not represent the search space of the optimization. Also, the result does not provide the link to execute the service.

Lee *et al.* use the flooding algorithm [7] to automatically compose web services. Their algorithm provides sequences of functionality needed to answer a request. However, the solution does not provide a link to invoke the chain of services. We cannot consider the answer as the search space for the optimization.

In order to deal with the optimization of dynamic composition, we propose to define the optimization problem and simulate it as a classic optimization problem. This topic is discussed in details in section 4.2.

## 3 Problem Formalizations

We define *complex queries* as requests for which no single atomic service (that can fulfill the request's goals) exists. As a consequence, complex queries require necessarily service composition. A classical example of such a request is given hereunder.

> *"I want to travel from Paris to London by air between August 5 and August 7 and returning 10 days later. In the meantime, I want to stay in a 3-star hotel and to rent a car, the whole at best price and reputation rating".* Additionally, we can add *"total price should not exceed €3000 and service quality should be at least satisfactory".*

The whole request contains distinct parts that can be considered as sub-queries, as well as multiple constraints and objectives. This example request, which serves as our experimental case study, requires together services for travelling, hotel booking and car rental. Each requested service is characterized by the *functional parameter*: *input, output, precondition* and *effect*. It contains different *constraints*: starting and arrival locations (Paris-London), transportation type (by air), date of departure (between August 5 and August 7), length of stay (10 days), and the quality level of the hotel (3-star). It also includes price and reputation rating objectives, the latter being combined as reputation rating. Last requirements specified concerning the whole trip are also constraint type, respectively on total price and reputation rating. The latter constraints are not functional, *i.e.*, they do not restrict *what* the services do or are expected to do (the *intended functions* of the services). Differently, they restrict *how* the services are performed or are expected to be performed (the *incidental outcomes* of using the services). These are typically non-functional constraints. The objective parameters are in the same time the constraint parameters.

### 3.1 Request Formalization

Complex requests *R* defines the requested set of functionalities $\mathrm{F_i}$, $R$= $\{\mathrm{F_i}\}$. In our travel example, the request requires three functionalities: transportation, hotel booking and car rental. In addition, each functionality has input and output *I* and *O*, respectively.
Considering constraints and objectives, we can reformulate the request as a quadruple: $R = < I, O, C, B >$, where $\mathrm{I} = (\mathrm{i_1}, \dots, \mathrm{i_l})^{\mathrm{T}}$ is the set of input, $\mathrm{O} = (\mathrm{o_1}, \dots, \mathrm{o_m})^{\mathrm{T}}$ is the set of output, $\mathrm{C} = (\mathrm{c_1}, \dots, \mathrm{c_n})^{\mathrm{T}}$ is the set of constraints and $\mathrm{B} = (\mathrm{b_1}, \dots, \mathrm{b_k})^{\mathrm{T}}$ the set of objectives. Since we consider web services, *I* and *O* are the URI's of functional parameters, and *C* and *B* are URI's of functional or non functional parameters. The functional parameters of services are described in the ontology of domain pertaining to the service whereas the non-functional parameters of service are described in a specific ontology, such as QoS ontology [8][9].

Generally, the constraints are imposed on some properties of the composition or on parts of it. Differently, the objectives correspond to the maximization or minimization of such a property's value, e.g., minimizing a price, maximizing reliability, reputation or availability. Some properties, like a price, can be the subject of both a constraint and an objective. For others, like reputation, their use in only an objective is more evident. The composite service matching a request is composed according to these constraints and objectives, and optimization is performed accordingly. As a corollary to this, constraints and objectives influence the size of search space.

Finally, both constraints and objectives can be of three different kinds: (a) related to services functional or non functional characteristics within a composition, *e.g.*, being able to book a flight with possibility of cancellation and with low agency fares; (b) related to data manipulated by services, *e.g.*, the price of the hotel should not exceed a certain amount or/and should be minimized; and (c) related to the composition, *e.g.*, the total trip cost should not exceed a given amount or/and should be minimized. The latter is taken as a penalty during the optimization phase.

### 3.2 Service Composition Formalization

The set of composite services obtained as a result of composition algorithm matching a user request is represented by a multilayered composition graph, which constitutes the search space for the optimization method. We formalize the composition graph as a unidirectional multi-layered acyclic graph, $G =< N, V >$, where $N$ is the multilayered set of service or data nodes, including the begin and end nodes of the graph, the *begin node* starts the composition graph and its output represents the input $I$ within the request $R$. The *end node* ends the composition graph and it is empty. And $V$ is the set of weighted arcs. The arc weights correspond to an objective value of the destination node. The objective corresponds to a non-functional parameter of the service.

As illustrated in Figure 1, we distinguish three types of nodes: Informative Services (IS), Active Services (AS) and data. The IS providers information (data). After its execution, the database of the service remains unchanged. The AS performs actions corresponding to their functional characteristics. The data base of the service is modified upon execution of these services. There are significant differences between IS and AS regarding their usage and purpose in our system. Indeed, in our algorithm the *IS* will be executed at composition time, *i.e.*, during the search for the composition matching the request, whereas the *AS* will be executed at composition user runtime, *i.e.*, after the optimization has been performed, and also after the user has selected its preferred composition among the best compositions proposed by the system.

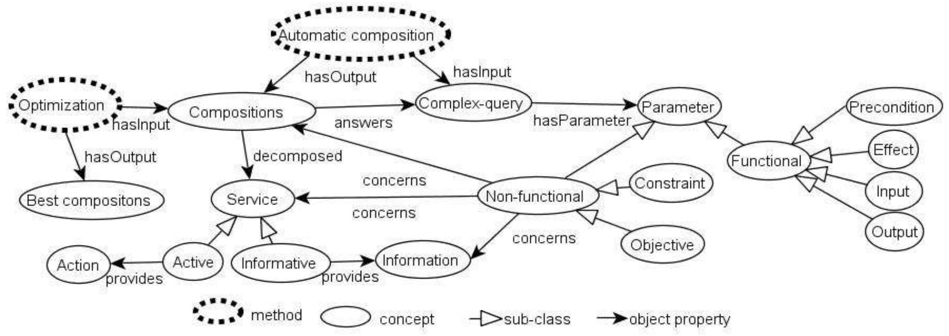


Figure 1. Conceptual description of services and parameters for dynamic service composition.

## 4 Resolution Approach

Our resolution approach is decomposed in two steps: (1) *automatic composition* consists to find a set of matching services, which represents the search space for the composition. (2) *Dynamic optimization* consists to select the best compositions at service execution time using the MOEA method.

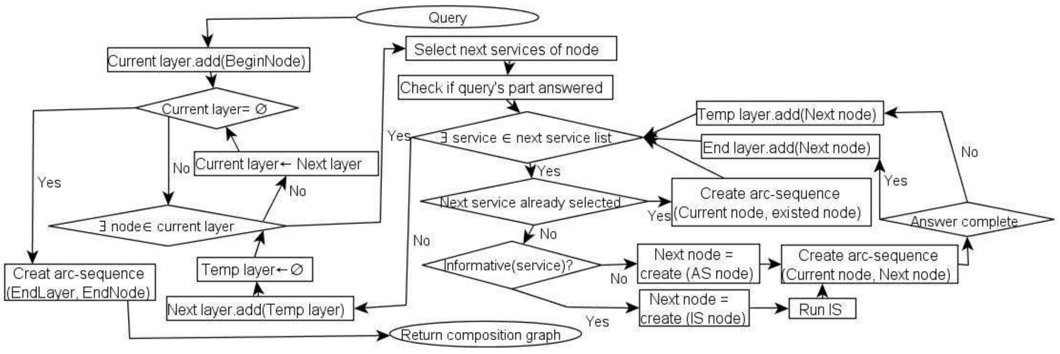
The request elements are used in a specific resolution step. The Input/output of the request are used to select the services, then the constraints related to services and data are used to filter out the irrelevant services and data obtained from the I/O-based pre selection. The composition-related constraints and all type of objectives are used during the optimization.

### 4.1 Algorithm Finding the Search Space of Composite Services

Our static composition algorithm providing the answer set is illustrated in Figure 2. This algorithm extends the flooding algorithm [7] and treats progressively the request to provide the search space of the composition. The terminology we use is defined as follow:

-The *current layer* is the set of nodes in the graph having a same depth level, currently being processed: $l_k=\{n_i\}$. Where $n_i$ have the same level of depth. Initially $l_k=\{$ *begin node*$\}$. One step of the algorithm corresponds to full covers of $l_k$. The node of $l_k$ being processed named *current node*. Initially, the *current node* "$n_i$" is the *begin node*

-The *temporary layer* is used to store the set of nodes following the current node, and not preceding *end node*. When precede *end node* are placed directly in the *end layer* of the graph.



Query
Current layer.add(BeginNode)
Current layer= ∅
Yes
No
Current layer← Next layer
No
∃ node∈ current layer
Creat arc-sequence (EndLayer, EndNode)
Temp layer←∅
Next layer.add(Temp layer)
Return composition graph
Select next services of node
Check if query's part answered
Yes
∃ service ∈ next service list
Yes
Next service already selected
No
No
Informative(service)?
No
Yes
Next node = create (AS node)
Next node = create (IS node)
Run IS
Temp layer.add(Next node)
End layer.add(Next node)
Create arc-sequence (Current node, existed node)
Yes
Create arc-sequence (Current node, Next node)
Answer complete
Yes
No

Figure 2. Schematic diagram of the service composition algorithm. *AS* stands for Active Services, and *IS* stands for Information Services.

The algorithm selects the service whose input matches a *current node* output, but if the *current node* outputs match one of the request outputs (a part of the requested functionality is covered), then the algorithm carries on by selecting the service which input matches another input of the request not yet covered by a matching service. The selected service is represented in the graph by the next node"$n_{i+1}$".

```
$If\ M(n_i.output, R.output_j) \leq \varepsilon\ then$
    $if\ M(R.input_{j+1}, n_{i+1}.input) \leq \varepsilon\ then\ select(n_{i+1}).$
$else\ if\ M(n_i.output, n_{i+1}.input) \leq \varepsilon\ then\ select(n_{i+1}).$
```

Where, $M$ the matchmaker functions, $\varepsilon$ the matching threshold.

When the selected service has already been selected previously, the corresponding node had already been created and added in the set of nodes *N* in *G*. Consequently, the next node will be selected from *N*. Otherwise, it will be created and an arc-sequence between the current node and the next node will be created. When the selected service is an *IS*, then the service will be invoked to obtained the information before creating the arc. When the data are obtained, the algorithm creates an arc sequence between the next node and the data nodes, then replace the next node by the data nodes.

The next node precedes the *end node* if all requested functionalities are covered. Otherwise, it is affected to the temporary layer, and then afterwards to the next layer. If all nodes of the current layer are covered, then the next layer replaces the current layer and so on until the next layer is empty. The algorithm terminated when this state is reached.

## 4.2 Selection of the Best Compositions

The selection of the best composition is equivalent to the optimization of the composition. This choice is driven by the definition of the optimization problem.

In simple case, the problem of web services composition optimization has already been addressed in terms of shortest path in a graph [10]. However, in a more general and realistic context, a composition of services has to fulfill multiple objectives such as cost minimization, time response minimization and consumer satisfaction maximization.

Addressing multi-objective optimization problems leads to search a tradeoff between non related objectives yielding a set of solutions. Therefore we define the optimization problem of the composition as a Multi-objective Shortest Path Problem (MSPP), which it is NP-hard. Therefore it is solved by approached optimization method, such as, the MOEA [11]. When the composition is dynamic, the problem can be defined as a problem of multicast multi-objectives shortest path [12], The tradeoff function for evaluating a service composition candidate (*sc)* is defined by the formula: $f(sc_i) = [f_{obj_1}, f_{obj_2}, \dots, f_{obj_k}]$, where $k$ is the number of objectives and $f_{obj_i}$ is the evaluation function for objective $i$ .

We define the Pareto front as $SC^* = \{sc_i^*\}$, with $sc_i^* = [obj_1^*, obj_2^*, \dots, obj_k^*]$ a non-dominated solution. Non-domination of solutions $sc_i^*$ in the front verified the condition: $\forall\, sc^* \in SC^*, \forall\, i \in \{1,2,\dots,k\}, \forall\, sc \in CSS, f_{obj_i}(sc^*) \succ f_{obj_i}(sc)$, where $\succ$ is the domination operator, i.e. $(\leq)$ in case of an objective to be minimized and $(\geq)$ in case of a maximization.

### 4.2.1 Multi-Caste Individual Coding and Initial Population

The individual coding is based into the optimization problem definition (multicast multi-objectives shortest path). However, some works, such as [13], presents the individual of genetic algorithm without based into the definition problem. According to our definition, the multicast individual coding considers all possible fail of service during the execution. For each fail, an alternative sub-composition to complete the initial composition is proposed. For this goal, we group all possible paths in the matrix, where each line presents a path (e.g. the path of first line is $S_1, S_3, S_5, S_9$ ).

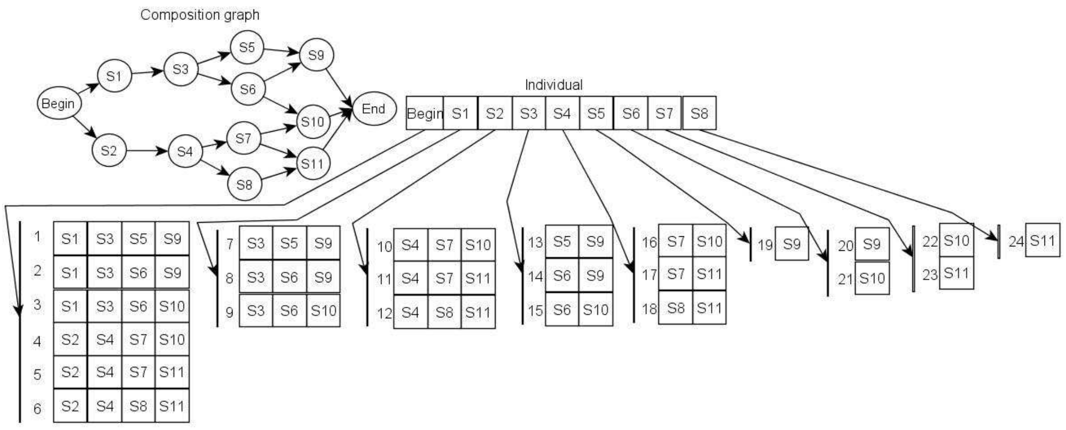


Figure 3. Multicaste individual coding

Figure 3 illustrates the multi caste coding, which presents all nodes in the composite graph. In our example, $Begin, S_1, \dots, S_8$, are the nodes represented individual. Services which precede the end node ($S_9$, $S_{10}$, $S_{11}$ in our example) are not presented because they are the last in the composition (i.e. the multi-caste member). The individual element value is the index of the matrix line (e.g. the element value of $Begin$, with the index *[1, 6])* and corresponds to the initial composition $CS^*$. $S_1$ can have the index value *[7, 9]*, $S_2$ the index value *[10, 12]…* and $S_8$ the index value *[24, 24]*. The initial population is chosen randomly among the paths of the composition graph. So, the evolutionary operation, crosser and mutation, respect the individual coding. The multi-cast individual presentation can also be simulated with another approached optimization method, such as the ant colony.

### 4.2.2 Multicast Evaluation

The multicast evaluation considers the fail probability $\gamma_i$ of the service "$S_i$" or the data "$D_i$". The service fail probability depends to the reliability or the availability of the service, which corresponds to a non functional parameter of $S_i$. The data fail probability is a constant and depends to the situation (e.g. if there are some perturbations in the transport then the information reliability is weak).

The importance level of the service $\theta_{S_i}$ depends on its position in the composition. Let's suppose a service composite with the sequence: $S_1, S_3, S_5, S_9$, with there $\gamma_i$ respectively equal to: 0.6, 0.5, 0.1, 0.9. According to the sequence, the service $S_3$ is more important than the service $S_5$, because the execution of $S_5$ depends to the execution of $S_3$. If $S_3$ fail then $S_5$ has no importance in the composition.
So, the importance costs of the service are respectively: $\theta_{S_1} = \frac{0.6+0.5+0.1+0.9}{\beta}$, …, and $\theta_{S_9} = \frac{0.9}{\beta}$. Generally: $\theta_{S_i} = \theta_{D_i} = \frac{\sum_i^r \gamma_i}{\beta}$, where *r* is the number of the services in the composition and $\beta = \sum \gamma_i = 2.1$.

Two values of $\theta_{S_i}$ for the service $S_i$ are possible, e.g., the sequences $S_1, S_3, S_6, S_{10}$ and $S_2, S_4, S_7, S_{10}$ are different but both contain the service $S_{10}$. Therefore, $S_{10}$ have values of $\theta_{S_{10}}$ which are calculated according to the path of the composition. The important cost will be multiplied by the objective value ( $\theta_{S_i}(Obj_1, \ \ldots, \ Obj_k)_{S_i}^T$) before evaluate the composition.

### 4.2.3 The Fitness

During the evolution phase the individuals (*ind*) are selected according to the value obtained by the evaluation function. The functions for the sequence structure without considers the evens possible, where evaluate the objectives price and reputation rating (*rr*) are: $f_{rr}(x) = \frac{1}{n}\sum_{i=1}^{n} \theta_{S_i} * rr_i$ and $f_{Price}(x) = \sum_{i=1,j=1}^{n,m} (\theta_{S_i} * pr_i + \theta_{D_j} * pr_j)$, where: *n* and *m* are respectively the number of services and data nodes in the composition and $x$ is the composition of services. The following evaluation function considers the evens possible, and assure the best alternatives composition for the events: $F_{rr}(ind) = \frac{1}{n}\sum_{i=1}^{n} \theta_{S_i} * f_{rr}(S_i)$ and $F_{Price}(ind) = \sum_{i=1,j=1}^{n,m} (\theta_{S_i} * f_{Price}(S_i) + \theta_{D_i} * f_{Price}(D_j))$, where $S_i, D_j \in ind$. The functions $F_{rr}$ and $F_{Price}$ are used during the optimization but the functions $f_{rr}$and $f_{price}$ give the real value without the possible events.

## 5 Events Processing

The service composite chosen, $cs^*$, from Pareto front represents the individual (e.g., if the individual sequence is "3-9-11-13-16-21-23-24", then the initial sequence of the composition is $S_1$, $S_3$, $S_6$, $S_{10}$ (see Figure 3)). When the service $S_6$ fails then the sub-sequence alternative of the composition corresponds to the index of predecessor, i.e., $S_3$ with the index equal to 13, which corresponds to the best sub-sequence $S_5$, $S_9$. So, the final sequence of the composition is: $S_1$, $S_3$, $S_5$, $S_9$. When the service $S_3$ fails, then we consider the index of $S_1$, which is equal to 9, and begin by the failed service $S_3$. To solve this problem, we consider the index of $S_1$ predecessor, which equal 3 and begin by $S_1$, $S_3$. In this case, we use the neighbor's solution in the Pareto front.

```
If ((S_i fail){
    If(S_i = Begin)
                    Propose neighbors;
    Else {Select sequence S_{i-1};
            While (S_i ∈ sequence(S_{i-1}) ∧ S_i ≠ Begin){
                i ← i − 1;
                Select sequence S_{i-1} ; }
        If(S_i = Begin)
        Propose neighbors; }}
```

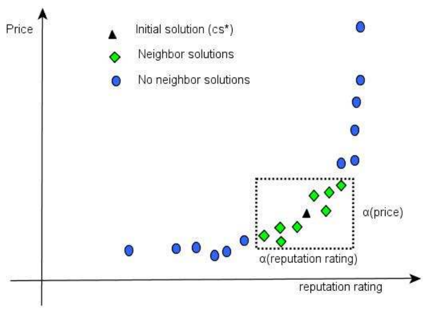


Figure 3. Composition with Pareto front

We define a neighbor according to the tolerance rate $\lambda$, where $\lambda \in [0, 1]$. The neighbors, service composites $cs$, satisfy the condition: $cs \in \psi,\ cs_{ob_1} \in \alpha_1 \wedge\ cs_{ob_2} \in \alpha_2 \wedge \dots \wedge cs_{ob_k} \in \alpha_k$. Where $\psi$ is the Pareto front, $\alpha_{ob_i} = [cs^*_{obj_i} - \varepsilon_i, cs^*_{obj_i} + \varepsilon_i]$ and $\varepsilon_i = \left|max_{obj_i} - min_{obj_i}\right| * \lambda$. When the set of neighbors is empty we increase $\lambda$ (see figure 3 and algorithm above).

## 6 Conclusion

This paper proposes a solution for the problem of dynamic composition optimization. The MOEA algorithms are used to provide a set of robust and best solutions (compositions of services). The solutions are distributed in the objectives space of the user request. The final choice is given to the end-user, who can select a solution according to the priorities he implicitly gives to objectives (knowing, of course, the available solutions).

As further researches, we'll try to generalize our method in order to be able to consider different types of composition structure such as: parallel or conditional. We'll also consider the precondition and effect in the selection of the service, which is an interesting area of research. The application of our method is currently discussed and we expect to apply it in the project WiSafeCar (http://wisafecar.gforge.uni.lu/) which focus on transport problems such as carpooling.

## 7 Acknowledgements

This work has been performed under the PhD grant TR-PhD-BFR07 funded by the Luxembourg National Research Fund (FNR).

## References

[1] Martin D., Burstein M. et al. OWL-S: Semantic Markup for Web Services. *W3C Member Submission*, 2004.

[2] Batouche, B., Naudet, Y., & Guinand F., Semantic Web Services Composition Optimized by Multi-Objective Evolutionary Algorithms, *ICIW: Internet and Web Applications and Services*, 2010.

[3] Alrifai M. and Risse T. Combining global optimization with local selection for efficient qos-aware service composition. *In 18th International World Wide Web Conference (WWW2009)*, April 2009.

[4] Sorin M. Iacob, ao Paulo A. Almeida, Jo and Maria E. Iacob. Optimized dynamic semantic composition of services. *In SAC '08 : Proceedings of the 2008 ACM symposium on Applied computing*, 2008. ACM.

[5] Matthias K and Andreas G, Semantic web service composition planning with OWLS-XPlan, 2005.

[6] Silva, E., and Sinderen, M. V. An Algorithm for Automatic Service Composition, *ICSOFT*, 2007.

[7] Oh, S., On, B., Larson, E. J., & Lee, D. BF: Web Services Discovery and Composition as Graph Search Problem, *e-Technology, e-Commerce, and e-Services, IEEE International Conference on*, 6-8, 2005.

[8] Kritikos K. and Plexousakis D. Semantic QoS Metric Matching. *ECOWS*. 2006.K. Kritikos and D. Plexousakis. Semantic qos metric matching. *In ECOWS: Proceedings of the European Conference on Web Services*, 2006.

[9] Gustavo F. Tondello and Frank Siqueira. The qos-mo ontology for semantic qos modeling. *In SAC '08 : Proceedings of the 2008 ACM symposium on Applied computing*, pages 2336_2340, 2008. ACM.

[10] X.X. Wang and H. Wang, "Web Services Selection and Composition based on the Routing Algorithm," *10th IEEE International Enterprise Distributed Object Computing Conference Workshops, pages 57_66*, 2006.

[11] Qiong Fan Fangguo He, Huan Qi. An evolutionary algorithm for the multi-objective shortest path problem. *International Conference on Intelligent Systems and Knowledge Engineering* , Oct. 15-16 2007.

[12] Randaccio L.S. and Atzori L., "Group multicast routing problem: A genetic algorithms based approach," *Comput. Netw*, vol. 51,2007.

[13] Wei-Chun Chang and Ching-Seh Wu and Chun Chang. Optimizing Dynamic Web Service Component Composition by Using Evolutionary Algorithms. *Web Intelligence*. 2005.