QoS-Aware Service Selection for Abstract Workflows using Provenance Data and Fuzzy Constraint Satisfaction Modeling

Mahsa Naseri and Simone A. Ludwig
Department of Computer Science, University of Saskatchewan, Saskatoon, Canada
{naseri,ludwig}@cs.usask.ca

Abstract

Service-oriented applications are using services that most accurately meet their requirements; as a result Quality of Service (QoS)-based service selection mechanisms play an essential role in service-oriented architectures. In service-oriented environments such as the Grid, we usually define abstract workflows to enable the binding of services at runtime. Using service discovery techniques to discover services that functionally match tasks of the abstract workflow, enables to differentiate services in order to select the ones that better satisfy user QoS requirements. In this paper, we model the QoS-aware service selection for abstract workflows as a Constraint Satisfaction Problem (CSP) and demonstrate that soft CSPs like fuzzy CSP are an appropriate approach for these purposes. We provide evaluations of the model and discuss how effective the process of abstract workflow service selection can be when merging it with the degree of satisfaction. Furthermore, we exploit the hill climbing strategy using special heuristics to achieve a speedup in order to make the search process more effective and scalable.

Keywords: *QoS, Provenance, CSP, Fuzzy CSP, OWL, Service Selection.*

1. Introduction

Service-oriented environments have special characteristics than other computing environments due to the nature of being open, dynamic, and distributed. The main concern of service discovery is finding services that match functional requirements of users. In distributed environments such as the Grid there are usually many services that perform the same task. In these environments, non-functional properties of services are a great help for differentiating services from each other. These non-functional properties are most often referred to as quality of service.

On the other hand in these kinds of environments, we usually define workflows as sequences of tasks that, when put together in a specific order, can achieve a specific goal. It must be possible to describe workflows without specifying a binding of each task to a service, so that the bindings can be added at run-time. In order to convert these types of workflows (also referred to as abstract workflows), to concrete ones, semantic descriptions of services along with matchmaking techniques are used.

Having found matching concrete services for each abstract task, we can use QoS parameters to select the concrete services that also match the non-functional requirements of the user. These types of service selection are usually referred to as abstract workflow service selection. The QoS specifications of services are given by the service providers and are usually described by ontology languages.

One of the growing demands in distributed service-oriented environments is the need for tracking, recording and managing data sources. This has led to the generation of the concept of provenance in Service-oriented architectures (SOA). In these environments, keeping track of information like how and which resources are being used for the execution of a workflow is useful for later reasoning. This information along with the processing steps presents the provenance data. In the proposed architecture, we discuss that along with the QoS specifications described in service ontologies, provenance data can be exploited to get trustworthy values for QoS parameters of services.

Previously, some works [1,2,3] have discussed allowing service consumers to express their QoS requirements as soft constraints, and subsequently used fuzzy logic for single web service selection and composition processes. [1] addresses the single service selection as a fuzzy multi-attribute decision making approach and compares the performance of this algorithm with random and round robin selection policies. In [2] the authors exploit fuzzy logic and compute both functional and non-functional

weightings of QoS criteria. While most of these papers do not provide evaluations of the soft constraints, [3] studied the effect on the overall satisfaction caused by a transition from hard to soft constraints with few evaluations. In [12] the authors exploit fuzzy distributed constraint satisfaction problem (DisCSP) to compose web services according to their QoS values. In DisCSP, variables and constraints are distributed among independent and communicating agents. They consider having several service providers as agents, each with their own constraints on QoS levels of parameters. By mapping each QoS parameter into a variable and service providers' constraints to network constraint sets, they solve the case of maximizing the global satisfaction with fuzzified constraints by negotiations between agents. In the case of abstract workflow selection, several works [4,5,6,7] have tackled this issue proposing exact algorithms or heuristics to optimally determine the appropriate concrete services for each individual component invocation or over the whole composite request. [7] maps the service selection for workflows into a Multi-Objective Programming model (MOP). [6] models it as an optimization problem and adopts a genetic algorithm for solving it, while [4] models the service composition as a mixed integer linear approach, where both local constraints and global constraints can be specified.

None of the approaches above investigate procedures regarding abstract workflow service selection with soft constraints. On the other hand, the papers discussed exploit soft constraints for single services but did only provide few evaluations on the results of the approaches. All approaches do not consider all factors that have an effect on the service selection with soft constraints. The factors contributing are few in the case of single service selection, however, there are more factors involved in workflow service selection. As a result, the investigation of soft constraint-based approaches for abstract workflow service selection is necessary.

In this paper, we present a new approach for QoS-aware service selection for abstract workflows and discuss that this problem can be considered as a fuzzy Constraint Satisfaction Problem (CSP). Therefore, by solving the fuzzy CSP, we will find the concrete services that best match nonfunctional properties of each abstract service by satisfying user QoS requirements of the whole workflow. We also investigate the effect of changes on the factors that have an effect on the abstract workflow service selection and present detailed evaluations on the model. To investigate the scalability of the approach, the fuzzy CSP model uses the hill climbing search method including abstract workflow service selection heuristics.

The next sections of this paper are organized as follows: In Section 2 we present how the QoS-aware abstract to concrete service selection conversion can be modeled as a fuzzy CSP; in Section 3 we provide the implementation details of the model; in Section 4 a case study is presented; in Section 5 experimental results and discussion on the evaluations along with the hill climbing implementation of the model are given; and in the last section we present the conclusion.

2. Modeling Abstract Workflow Service Selection as a Fuzzy CSP

For distributed environments such as the Grid, where the workflows and the number of services that provide the same functionality are large, classical CSP modeling is not a good choice for this purpose. Exploiting the classic CSP approach for the abstract workflow service selection is suitable where the workflow is not too large and the number of concrete services matching each abstract one is small. In case of the Grid, it will result in a CSP with many variables and constraints. Furthermore, there might no sets of concrete services available that can fully satisfy the QoS requirements the user has specified. As a result, we propose using soft CSPs, in particular fuzzy CSPs [9], for these kinds of service selections.

In general, in soft CSP not all the given constraints need to be satisfied if all constraints cannot be met. One approach to deal with soft constraints is to generalize the notion of crisp constraints. A crisp constraint is characterized by the set of tuples for which the constraint holds. Hence, it is a natural extension that a fuzzy set characterizes a fuzzy constraint, i.e. different tuples satisfy the given constraint to a different degree.

In the case of service selection, this means that instead of fully satisfying the QoS requirements, they can be satisfied to some degree. The required degree can be specified explicitly by the user or can be implicitly defined within the algorithm. For example, the algorithm might be organized to change the degree automatically in case the execution time or memory usage exceeds a specified limit. In the evaluation section, we present how much this degree affects the performance of the algorithm.

To compute the degree of satisfaction for a sequence of concrete services, we exploit matrices S, the summation matrix, and R, the requirement matrix. For each row in S, which represents a QoS attribute, if

the value for that row in R is greater than or equal to the corresponding value in S then the satisfaction equals to 1, otherwise we calculate the ratio of satisfaction. We then compute the average satisfaction for all attributes and retrieve the overall achieved satisfaction. This approach can be improved by assigning a weight to each QoS attribute. The weights represent the importance of the attributes to the user. We can then compute the degree of satisfaction by using the weights to consider the importance of the attributes. In this model, backtracking will take place when the calculated degree is less than the required degree of satisfaction.

3. Architecture of the System

In service oriented environments we need to track and record data sources for later reasoning. This has led to the generation of the concept of provenance in SOA. Provenance data is referred to as the data that is recorded either during the execution of a workflow, which is referred to as execution provenance, or by a particular service reporting about itself or its provider. Provenance data enables users to trace and identify the individual services as well as their corresponding inputs and outputs that were involved in the production of a specific data result. This data is usually stored in a provenance store which is a combination of one or several databases.

In current semantic web systems, QoS specifications for each concrete service are described in OWL [10] ontologies. The ontological QoS specification of service providers are updated periodically while there might be many requests during each period. In case the QoS guarantees change during a period, the providers will not be able to satisfy the agreed-on thresholds. On the other hand, for service providers, in order to assure that they comply with the promised QoS, an agreement is usually made between the provider and consumer which is referred to as Service Level Agreement (SLA). Failing to meet SLAs could result in serious financial consequences for a provider. However, service providers may not be trustworthy enough to deliver the services based on the agreed-on QoS. Besides, the "Validity period" of the agreement might have come to an end and no agreement updates might have been made afterwards.

To overcome the inconsistencies between the guaranteed and delivered QoS values of services, we propose to retrieve QoS values from the provenance data and to exploit it along with the QoS specifications of services given by the OWL ontologies.

In the provenance store, we record information about the previous executions of different workflows to allow future reasoning. This data can include input/output datasets of services, start and end execution time, the status of execution of the whole process or each service etc. We can exploit the provenance store to retrieve trustworthy information about the QoS promises of services. For example, we can use the start and end time of the execution process of services to measure the QoS parameter "*time*". Similarly, we can keep track of the status of the execution of services or use the end time and get estimates for the "*reliability*" of services. Availability of a service would have a set of Boolean values gathered over a period and can be recorded by a client about a service. By counting the number of times the service was available and dividing this value by the total number the service was called for interaction, we can compute the QoS parameter "*availability*".

The QoS processor first extracts the values of the QoS attributes of concrete services from their service ontologies using the Protégé OWL API [11]. In the next step, for each concrete service the processor queries the provenance store and retrieves QoS values for some attributes such as execution time, reliability and availability. We can set the query to select the values that date back to a specific time; for example, a time after the last renewal of the attributes of ontologies or the agreements. To get the final values for these QoS attributes, we have two options. The first option is to take the average of values from both sources: provenance data and ontology. The second option, which we used in our implementation, is to assign a weight value to both sources. This weight value represents the importance or the trust we can put on the sources. Thus, in order to achieve the final value (V) for QoS parameter P for the service instance S_i, we multiply the weight value for the provenance data, i.e. w_1, by the value that we retrieve for that parameter of this service from the provenance data, added by the weight value the provenance data, added by the weight value for service ontologies, i.e. w_2, multiplied by the value we retrieve for the same parameter from the service's QoS ontologies. This statement is summarized in the following formula:

$$V_i = w_1 \cdot V_{1i} + w_2 \cdot V_{2i} \text{ and } \sum_{j=1}^{2} w_j = 1$$

where w_j represents the trust/weight value of resource j, V_i represents the final value for some QoS parameter for service i and V_{ji} represents the value retrieved for that parameter from resource j.

The matrix M is then built by the QoS processor

Table 1. QoS values for the sample workflow scenario.

Service Name		Time		Cost	Reliability	
		Provenance	Ontology	Ontology	Provenance	Ontology
Text_file_ Reader	Srv_1	11.5	12.5	10	0.85	0.75
	Srv_2	15	14.6	11	0.72	0.7
	Srv_3	17	16.5	8	0.8	0.85
Extract_TSV_ ChEBI	Srv_1	20.8	19.2	12	0.48	0.52
	Srv_2	13	12	9	0.64	0.6
	Srv_3	12	10	15	0.7	0.6
Remove_non_ MDL	Srv_1	15	14	8	0.4	0.4
	Srv_2	10.5	9.5	12	0.52	0.48
	Srv_3	13	15	10	0.3	0.4

for each abstract service and along with the requirement matrix is sent to the Fuzzy CSP solver. The result of the solver will be the list of concrete services that perfectly match the requested QoS values with regard to the requested degree of satisfaction.

4. A Case Study

Consider an abstract workflow scenario for a bioinformatics application. Chemical Entities of Biological Interest (ChEBI) [13] is a freely available dictionary of molecular entities focused on 'small' chemical compounds. ChEBI encompasses an ontological classification, whereby the relationships between molecular entities or classes of entities and their parents and/or children are specified. This workflow scenario loads a Tab Separated Value (TSV) file from the ChEBI database. The TSV is a very simple textual data format which allows tabular data to be exchanged between applications that use different internal data formats. After the extraction of the molecules from the TSV file all non MDL mol files are removed. MDL is a format for chemical table files. Later, the valid molecules are inserted into a database.

The workflow is composed of four services for reading the text files, extracting TSV files from the ChEBI database, removing non MDL files and inserting the rest of the molecules into a database.

Suppose that the service discovery process has discovered the first three matching concrete services for these four abstract services. The details of QoS values for the three of these services are given in Table 1. We assumed that for the user the trust weight on both QoS sources, i.e. provenance data and ontology descriptions, equals to 0.5.

The required degree of satisfaction is 0.85 and the QoS requirements for time, cost and reliability are 40, 30 and 0.8. Figure 1 shows an instance of a part of the execution process. In the third step, the satisfaction degree becomes 0.84 which is less than the required degree of satisfaction. Thus the algorithm chooses a different concrete service for Remove_non_MDL, thus the required degree of satisfaction is fulfilled. It then selects a concrete service for the last task of the workflow, guaranteeing that the overall satisfaction score is equal or greater than 0.85. Thus, the valid molecules are inserted into database. Having followed this procedure, a set of concrete services are selected for the abstract workflow in a way that the required QoS values are satisfied with 0.85 degree of satisfaction.

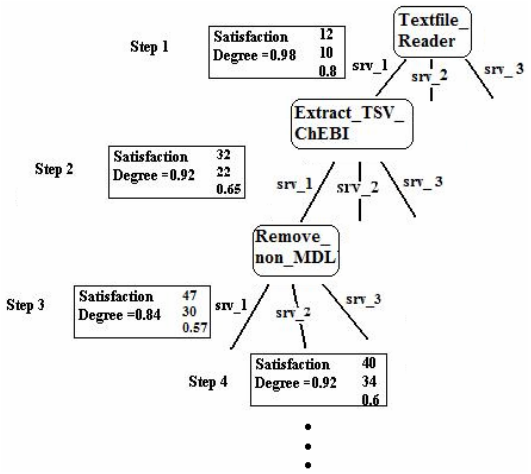

Figure 1. Keeping track of the execution steps.

5. Experimental Results

In this section, we provide evaluations of the presented approach. For the abstract workflow service selection there are some important factors that can affect the performance of the model, and investigating the degree of satisfaction with regard to these factors is very important. We have considered all the factors that affect the performance of the fuzzy CSP solver. In the following parts of this section, we will show and discuss how each factor can affect the execution time of the algorithm. The factors include: the number of abstract services, the number of concrete services, the

number of attributes and the degree of satisfaction. The number of abstract services presents the number of services the workflow is composed of, while the number of concrete services reflects the number of services that functionally match each abstract service in the workflow.

The application was implemented in Java and the experiments were run using an Intel 1.5 GHz dual CPU machine with 2 GB of RAM memory. To get reliable results, we executed the algorithm five times for different datasets. The datasets were generated by narrowing and broadening the range of the requested QoS attributes to make sure that the process has been tested for any kind of data and with different values for the factors.

5.1. Number of Abstract Services

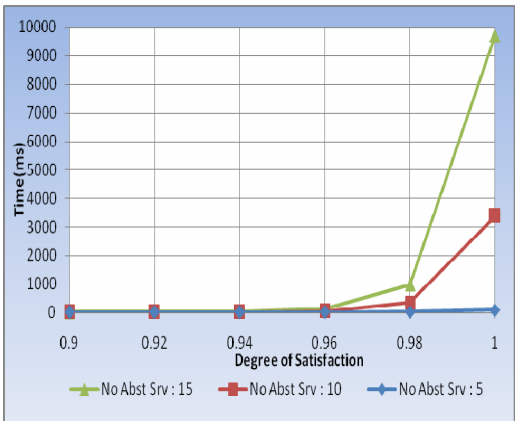

Figure 2. Effect of increasing the number of abstract services (No Abst Srv) for degrees of satisfaction 0.9 to 1.

The number of abstract services reflects the number of services the abstract workflow is composed of. It can be expected that the larger the workflow, the greater the execution time of the algorithm will be. We increased the number of abstract services along with the degree of satisfaction. Keeping the number of QoS attributes as well as the number of matching concrete services for each abstract service to 5, we repeated the evaluation with the number of abstract services ranging from 5 to 15 with increments of 5. For each set we ran the algorithm for degrees 0.7 to 1 with an increment of 0.1. We observed that there is a little change in the execution time when the required degree of satisfaction is below 0.9. The slope begins to change considerably from 0.9 to 1, especially for larger numbers of abstract services. It can be inferred from the experiment results that if a fully satisfying set of concrete services for the abstract workflow is available, an increase in the number of abstract services does not affect the performance of the algorithm much while the required degrees of satisfaction is below 0.9.

To investigate the behavior of the presented approach for the range 0.9 to 1, we evaluated the algorithm for this range with increments of 0.02 while the same number of concrete services and attributes were kept. Figure 2 shows the results of the evaluation. The notable variations can be observed from degree 0.96 onwards. The slopes of the lines continue to increase slowly until 0.98 and the major changes take place between 0.98 and 1. Larger numbers of abstract services have a greater effect on the slope and therefore on the execution time of the approach.

5.2. Number of Concrete Services

We evaluate the effect of the number of concrete services on the performance of the algorithm. It was explained earlier that the number of concrete services specifies the number of columns of the domain matrix, i.e. M, for each abstract service. For a good result, we kept the number of abstract services constant to 10 and the number of attributes to 5. We then increased the number of concrete services from 5 to 15 with increments of 5.

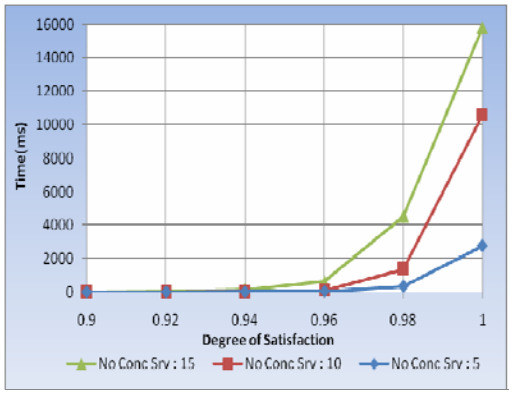

Figure 3. Effect of increasing the number of concrete services (No Conc Srv) for degrees of satisfaction 0.9 to 1.

Similar to the previous evaluation, we ran the algorithm for different datasets and for a degree of satisfaction range varying from 0.6 to 1 with increments of 0.1. Again as expected, increasing the number of concrete services will increase the execution time of the algorithm. Again, it is interesting to notice that like in the previous evaluation, not much variation can be observed in the range of 0.7 to 0.9, and the significant changes occur in the range of 0.9 and 1.

To investigate this further, we repeated the evaluations this time from 0.9 to 1 with increments of 0.02. The experiment was run for different datasets with 5, 10 and 15 concrete services. Figure 3 shows the results. We can see that as we increase the number of concrete services, the variations in the slopes of the curves can be observed from an earlier degree of satisfaction which is completely reasonable as increasing the number of concrete services makes the search process more time consuming, especially when the constraints can be hardly satisfied.

5.3. Number of Attributes

The number of QoS attributes for services is another factor that can affect the performance of the algorithm. These attributes consist of execution time, cost, reliability, etc. As discussed in Section 3, the number of attributes specifies the number of rows of the matrices. This implies that it should have a strong impact on the execution time of the algorithm. For this purpose, we kept the number of abstract services constant to 15, the number of concrete services to 10, and ran the algorithm with 3, 5 and 7 number of attributes while varying the degree of satisfaction from 0.8 to 1. The results are presented in Figure 4. The interesting point of this experiment is that as the number of attributes increases, the choices for partially satisfying the degree of satisfaction also increases. As a result in some of the experiments, we observed that for some degrees of satisfaction less than 1, the execution time was smaller for a larger number of attributes. This depends on the dataset and the resulting tree of possibilities which is being constructed by the fuzzy CSP and this can lead to a shorter execution time for a larger number of attributes.

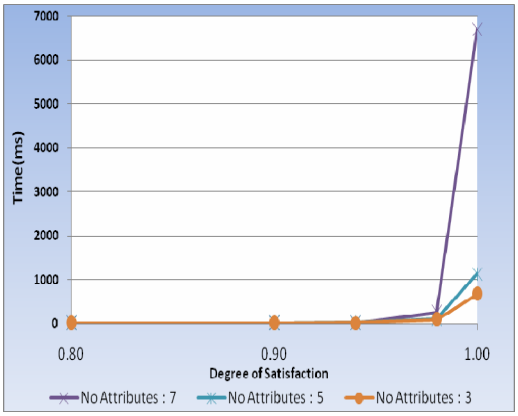

Figure 4. Effect of increasing the number of attributes for degrees of satisfaction 0.8 to 1.

5.4. Scalability

The fuzzy CSP algorithm implemented and evaluated so far was based on the classic backtracking and depth first search. This technique is complete but has the shortcoming of being time consuming and does not support scalability. For hard constrained service selection problems, and for workflows with a large number of services, evaluations of the implemented approach resulted in very high execution times. Therefore, in order to enable scalability of the model and investigate its behavior for large and strictly constrained abstract workflow service selections, we exploited the stochastic search method, hill climbing, and applied it to our fuzzy CSP model. To exploit the hill climbing method for solving the abstract workflow service selection with fuzzy CSP modeling, we came up with some heuristics to start with a good solution in the search space and improve the search faster.

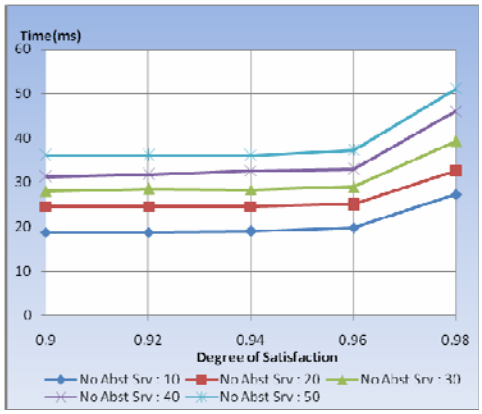

Figure 5. Scalability of the hill climbing algorithm for 10 to 50 numbers of abstract services.

The algorithm starts by computing the distance between the QoS parameters vector of each concrete service with the requirements vector. For each abstract service, the concrete service with the largest distance is selected. Therefore, the starting point in the search space is a suboptimal point which helps to retrieve the partial solution faster. The search process begins improvements by selecting the QoS parameter that has the lowest overall satisfaction. It then finds the abstract service, i.e. the variable, which had the lowest value for that parameter and tries to find and assign another concrete service to it in a way that improves the satisfaction degree. This procedure continues until no more abstract service assignment can be improved.

Experiments were done to evaluate the proposed algorithm for large workflows. The algorithm was repeated for 10 to 50 numbers of abstract workflows with increments of 10 and for degrees of satisfaction

0.9 to 0.98 with increments of 0.02. For each experiment the number of concrete services and the number of attributes were kept constantly to 5.

Figure 5 shows the result of the experiments. It can be seen that the approach is very fast in comparison to the backtracking approach. There is not a large variation in slopes for higher degrees of satisfaction while increasing the number of abstract services, and therefore the increase in the execution time remains relatively small.

6. Conclusion

In this paper, we put forward an approach to solve the service selection for abstract workflows using CSP. We discussed that none of the previous approaches which targeted this problem used soft constraints. We provided an abstract service selection model exploiting fuzzy CSP and outlined that fuzzy constraint satisfaction modeling is even better for this purpose because either it can find a partial solution in the case no fully satisfiable solution exists or it can result in a faster solution by decreasing the degree of satisfaction in cases when the QoS requirements are fully satisfiable. We also applied new heuristics and also solved the presented model using the hill climbing algorithm to observe whether fuzzy CSP for abstract workflow service selection will scale for real application scenarios.

The presented approach is also applicable for cases when the user tends to specify a range for each required QoS attribute, instead of a single value. Therefore, instead of comparing the values of the summation matrix with the requirement matrix, we check if the correspondent values in the summation matrix are within the range specified in the requirement matrix.

The evaluations showed the effectiveness of considering the degree of satisfaction for solving the service selection process. It was observed from the evaluations that for a service selection where the constraints can be fully satisfied, an increase in the number of factors does not affect the performance of the algorithm much, while the required degree is less than 0.95. On the other hand, for computing the degree of satisfaction for a set of abstract services, we may want to merge the importance of each attribute, its weight, into the calculation. Thus, in the case when one or more attributes are not particularly important for the requester, we might be able to save some time as the algorithm may need to search longer to satisfy these attributes completely.

7. References

[1] H. Tong, S. Zhang, “A Fuzzy Multi–Attribute Decision Making Algorithm for Web Services Selection based on QoS”, Proceedings of the 2006 IEEE Asia–Pacific Conference on Services Computing (APSCC’06), 2006.

[2] P. Wang, K. M. Chao, C. C. Lo, C. L. Huang, Y. Li, “A Fuzzy Model for Selection of QoS Aware Web Services”, Proceedings of the 2006 IEEE International Conference, 2006.

[3] S. Chung, O. Hafeez, M. De Cock, “Selection of Web Services with Imprecise QoS Constraints”, 2007.

[4] D. Ardagna and B. Pernici,“Global and Local QoS Guarantee in Web Service Selection”, In Proc. of Business Process Management Workshops, pp 32–46, Sept. 2005.

[5] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, “Heuristics for QoS-aware Web Service Composition”, In Proc. of Int’l Conf. on Web Services, Sept. 2006.

[6] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. “An Approach for QoS-aware Service Composition Based on Genetic Algorithms”, In Proc. of GECCO 2005, June 2005.

[7] Y. Qu, C. Lin, Y.Wang, and Z. Shan., “QoS-aware Composite Service Selection in Grids”, In Proc. of Int’l Conf. on Grid and Cooperative Computing, pages 458–465, Oct. 2006.

[8] E.P.K. Tsang, “Foundations of Constraint Satisfaction”, Academic Press, London and San Diego, 1993.

[9] Zs. Ruttkay, “Fuzzy Constraint Satisfaction”, Proceedings 1st IEEE Conference on Evolutionary Computing, Orlando, USA, 1994.

[10] OWL Web Ontology Language Reference, retrieved May 9, 2009, from http://www.w3.org/TR/owl-ref/, 2004.

[11] Protégé-owl api programmer's guide, retrieved January 21, 2009, retrieved May 9, 2009, from http://protege.stanford.edu/plugins/owl/api/guide.html, 2006.

[12] X. T. Nguyen, R., Kowalczyk, M. T. Phan, “Modeling and Solving QoS Composition Problem Using Fuzzy DiscCSP”, IEEE International Conference on Web Services, 2006.

[13] Chemical Entities of Biological Interest (ChEBI), retrieved May 9, 2009, from http://www.ebi.ac.uk/chebi/.