2011 35th IEEE Annual Computer Software and Applications Conference

An Intelligent Broker Approach to Semantics-based Service Composition

Yufeng Zhang
National Lab. for Parallel and Distributed Processing
Department of Computing Science
National Univ. of Defense Technology, Changsha, China
Email: yufengzhang@nudt.edu.cn

Hong Zhu
Department of Computing and Electronics
Oxford Brookes University
Oxford OX33 1HX, UK
Email: hzhu@brookes.ac.uk

***Abstract*—This paper proposes an intelligent broker approach to service composition and collaboration. The broker employs a planner to generate service composition plans according to service usage and workflow knowledge, dynamically searches for services according to the plan, then invokes and coordinates the executions of the selected services at runtime. A prototype called *I-Broker* has been implemented to support the approach, which can be instantiated by populating the knowledge-base with domain specific knowledge to form domain specific brokers. This paper also reports experiments that evaluate the scalability of the approach.**

***Keywords*-Service oriented computing; Service composition; Service broker; Planning;**

I. Introduction

Service composition is crucial to the success of service oriented computing. A great amount of effort has been reported in the literature. However, it is still one of the remaining most challenging issues. Existing approaches advanced in the literature include service orchestration and choreography through workflow definitions and models executed on workflow engines, and employment of AI planning systems. An alternative is to employ service brokers for services discovery, mediation and collaboration. As Sycara *et al*. argued [1], [2], broker is a promising approach because of its high flexibility and wide applicability.

However, existing web service brokers [1]–[4], have not delivered the promise, yet, because their functionalities are limited to the following:

- Interpreting the semantics of service queries and the registered capabilities of service providers;
- Searching for the service providers that matches a requester's query and sometime selecting the one with best track record of quality of services;
- Invoking the selected service provider on the requester's behalf and interacting with the provider if necessary to fulfil the query; and
- Returning query results to the requester.

Although a significant amount of reasoning is performed by such brokers to fulfil these functions, existing brokers are still not intelligent enough to deal with the complexity of dynamic service composition and collaboration. It is because a service request can rarely be fulfilled by one service provider directly. In such cases, a requested service needs to be decomposed into a number of subtasks and fulfilled by a number of different services.

Addressing this problem, this paper proposes a framework to enhance the power of service brokers with the capability of:

- decomposing requested services into a number of subtasks,
- searching for the best fit services for each subtask, and
- composing and coordinating these services in execution.

This is achieved by developing a planning technique. We have implemented a prototype framework called *I-Broker* and conducted a case study and experiments on its scalability.

The paper is organised as follows. Section 2 presents the proposed approach. Section 3 describes the prototype implementation of the framework. Section 4 briefly reports the experiments with the prototype system. Section 5 concludes the paper with a discussion of related work and future work.

II. The Proposed Approach

A. Overview

The proposed approach is based on Semantic Web Services. It is assumed that services are registered at a matchmaker with semantics descriptions in a given ontology. Service brokers are also services that are registered with matchmakers. They also invoke the matchmaker to fulfil its functionality.

Similar to other service brokers, our broker receives service requests represented in the form of XML with vocabulary defined in the ontology of the application domain. Once received a task, it searches in the semantic WS registry for services that are capable of fulfilling the task, where the capability is also expressed in the ontology. The matching between requested task and service capability is performed as ontology reasoning via invoking the semantic WS matchmaker [1].

However, it differs from existing brokers in case when there is no good match found in the registry. Our broker further analyses the requested task according to a knowledge-base about how tasks can be fulfilled by performing a

0730-3157/11 $26.00 © 2011 IEEE
DOI 10.1109/COMPSAC.2011.11

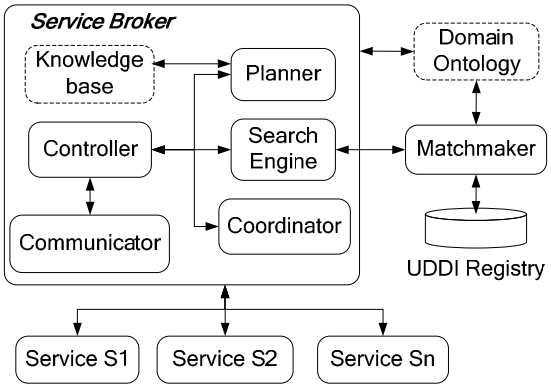

Figure 1. Architecture of the Intelligent Broker

number of subtasks. It decomposes the task into several subtasks if possible, and sets up an abstract service composition plan, which defines which kinds of subtasks and how they should be composed together to complete the original task. The broker then instantiates the abstract plan into an executable concrete composition plan by searching the registry to discover the services that are capable of fulfilling each subtask and selecting the best one if more than one candidates are available. The services selected are invoked and the information is passed between them via the broker. In this process, the broker may assemble data from various sources into messages and/or use them as parameters of service calls. Finally, the broker returns service results to the requester.

B. Architecture of Intelligent Service Brokers

Fig. 1 shows the overall structure of the broker, which contains the following components.

1) Communicator: It provides an interface to submit service requests in the form of tasks and receive results of the service invocation, if the task is performed successfully. It transforms messages into the internal representations for further processing. The results are also assembled into SOAP messages as responses to the requester.

2) Knowledge-base: It contains codified knowledge on how a task can be fulfilled by a number of subtasks. Each type of tasks is defined by a set of parameters. There are two kinds of parameters: descriptive parameters and functional parameters. The former describes the functionality of the task, such as the activity of the task, the execution environment of the task, and so on. The latter gives the data to be transformed by the task, including input and output data. The values of these parameters are concepts defined in the ontology of the application domain. The knowledge is represented in the form of rules:

$$T(p_1, \cdots, p_n) \Rightarrow T'_1(p_{1,1}, \cdots, p_{1,n_1}); \cdots; T'_k(p_{k,1}, \cdots, p_{k,n_k})$$

where T is a task and $p_1, \cdots, p_n$ are its parameters. It means that the task T can be decomposed into k subtasks T'_i with n_i parameters $p_{i,1}, \cdots, p_{i,n_i}$, $i = 1, 2, \cdots k$.

It is required that the parameter $p_{i,j}$ of subtask T'_i is constructed from $p_1, \cdots, p_n$ and the output parameters of its previous subtasks, i.e. a subset of $\{p_{x,y} | x < i, y \leq n_x\}$. This means that the subtasks can be executed in the order as they occur in the rule. The value of a parameter will be passed from one to the next according to the parameters dependency between subtasks.

It is also required that each of the output parameters of task T is constructed from the set of output parameters of subtasks T'_i $(i = 1, \cdots, k)$. This is to ensure that task T is realized by the subtasks in the rule.

Therefore, a rule is not only a logic decomposition of a task into several subtasks, but also an expression of the workflow and the collaborations between various kinds of services to complete a specific kind of task. Moreover, from computational point of view, these rules also provide heuristic rules for narrowing the search space for generating service composition plans.

3) Planner: It analyses the requested service and searches the registry to determine if it can be fulfilled by services existing in the registry directly. If not, it searches the knowledge-base to find how it can be decomposed into subtasks and generates abstract service composition plans. For a given requested service, there may be multiple plans that can fulfil it. In particular, it compares the requested service with the rules in the knowledge-base. When a service request matches the task on the left-hand-side of '$\Rightarrow$' of a rule, i.e., the parameters of the requested service match the parameters of the task, the rule is applicable. The parameters of the corresponding subtasks of the rule will then be instantiated with the corresponding values assigned to the parameters of the task. An abstract composition plan is thus generated.

4) Search Engine: Given an abstract service composition plan, the search engine calls the matchmaker of semantic WS registry to find appropriate services for each subtask. The search request is constructed according to the description of the subtasks generated by the task planner as in the abstract service composition plan and submitted to the matchmaker. The search result returned by the matchmaker may include multiple candidates. Each of them is tagged with a score that represents its fitness to the searched capability. The higher the score is, the more suitable to fulfil the subtask, and hence the higher priority to be selected to perform the corresponding subtask. Through selecting service candidates to fulfil the subtasks, the abstract service composition plan is instantiated into an executable concrete service composition plan. To achieve a higher flexibility and fault tolerance, the candidates that have lower scores but above a threshold are also preserved. When the selected service fails later on in the invocation, the candidate with the second highest score,

if any, is then selected.

5) Coordinator: It is responsible for the invocation of selected services for the subtasks with appropriate parameters and for the coordination of these services. In particular, the results of previous services are re-assembled as parameters for the following services according to their dependency.

6) Controller: It controls the execution of the broker. Once it receives a request of a task from the communicator, it invokes the planner to generate a set of abstract service composition plans according to the knowledge-base. Then, it selects a plan and invokes the service search engine to search for appropriate services registered in the matchmaker for each subtask in the selected abstract composition plan. If succeed, one service will be selected for each subtask. Subsequently, it informs the coordinator to invoke and coordinate the selected services according to the concrete composition plan.

C. Control Process

Roughly speaking, the broker attempts to fulfil a service request through the following five stages:

1) *Planning*: In this stage the broker generates an abstract service composition plan;
2) *Searching*: In this stage the broker searches for appropriate services for each subtask;
3) *Invocation*: In this stage the selected services are invoked in the order that parameters of the services depends on each other;
4) *Collecting*: In this stage the broker collects results of service invocations and passes data between them;
5) *Delivery*: The final stage is to return the results collected from the services to the user.

This process is not always successful straightforwardly. Therefore, we devised a backtracking mechanism to ensure all possible compositions are tried before giving up. In particular, if any of the steps in the attempt to fulfil a task fails, the controller backtracks to an earlier step and tries an alternative. Fig. 2 shows the control process of the broker and how failures trigger backtracking.

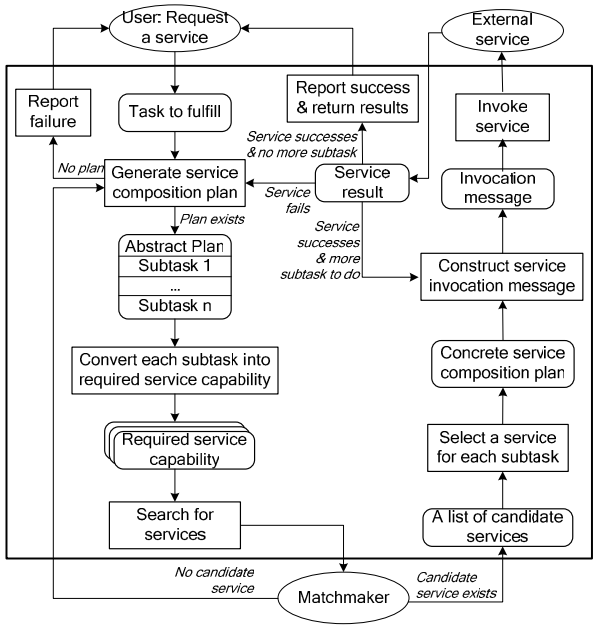

Figure 2. Broker's Control Process

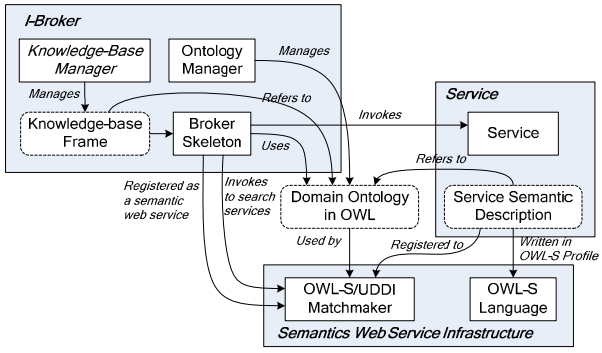

Figure 3. Relation between I-Broker and Semantic WS

D. The Prototype I-Broker

We have implemented a prototype framework called I-Broker based on Semantic WS. Fig. 3 shows the relationship between the components of the framework and the facilities provided by Semantic WS infrastructure. We assume that the semantic information of WS is described in the form of service profiles in OWL-S. These services are registered to an OWL-S/UDDI Matchmaker, which provides semantic-based service search facility.

The framework implements the components in the broker architecture presented in Section 2.2. These components form a skeleton of a service broker. When the knowledge-base is populated with rules about the workflow in a particular application domain, the skeleton is instantiated into a functional broker. The knowledge-base is represented in XML that consists of a number of rules. The broker itself is a semantic WS. It can be registered to the matchmaker, and searched for and invoked by users.

E. Running Example

As a running example, we have populated the knowledge-base with the knowledge of software testing process and built a broker specialized in software testing. Details of the running example are omitted for the sake of space, which can be found in [5].

III. Experimental Evaluation

This section briefly reports our experiments with the framework. The objectives of the experiments are to evaluate the scalability of the proposed approach.

A. Design of the experiments

There are three types of objects in the experiments.

- *Services*: They are registered to the semantic WS registry. Their capabilities are represented in the form of service profiles. The test broker searches for services by submitting requests to the Matchmaker.
- *Rules* in knowledge-base: They are the knowledge of workflow in the application domain and the usages of specific services. They are represented in the form of XML files and stored locally within the broker as the knowledge-bases.
- *Service requests*: They are the service requests submitted to the brokers and represented in the format of XML using the ontology.

The data mutation technique [6] is applied in the preparation of the objects for the experiments to overcome the difficulties due to their structural complexity. In particular, for each type of objects, we first select a set of real ones, which are called the seeds in data mutation technique. Then, we apply a set of data mutation operators systematically to each seed to generate a set of mutants of the real ones so that the mutants are of subtle differences from the seeds.

We designed a group of mutation operators that are applicable to service profiles, which are used in service capability registration, and the descriptions of tasks and subtasks of rules in the knowledge base. Each operator changes the value of one parameter in the seed according to the ontology. The particular ontology used in the experiment is STOWS, which stands for Software Testing Ontology for Web Service [7].

Let x be any of the parameters in service profiles. The data mutation operators are defined as follows.

- *RxF*: Replace the x parameter in the profile, which is a class in the ontology, by its father class in the ontology;
- *RxS*: Replace the x parameter in the profile by one of its subclasses in the ontology;
- *RxB*: Replace the x parameter in the profile by one of its brother classes in the ontology;
- *RxN*: Replace the x parameter in the profile by a class in the ontology that has no relation to the parameter.

For instance, if the service classification of a profile is *TestCaseGeneration*, applying *RSB* operator to this parameter, we get several profiles whose service classifications are *TestCaseExecution* and *TestResultValidation*, etc., provided that they are the sibling classes of *TestCaseGeneration* in the ontology.

Table I gives the set of services used as the seeds for generating the service type of objects. Table II gives the number of seeds and mutants of each type of subjects used in the experiment.

B. Main Results of the Experiments

We have conducted the following 3 experiments to evaluate the broker's scalability.

Table I
SEED SERVICES USED IN THE EXPERIMENT

Name	Description
CASCAT	Generate test cases from algebraic specifications
Test Translator	Translate test cases from CASCAT format into Test Executor format
Test Executor	Execute tests for a numeric calculator WS
Klee	Symbolic execution of C code
Magic	Check component's conformance to specification
XML Comparator	Compare XML files
Java NCSS	Metrics for Java program
Findbugs	Find bugs in Java program by static analysis
PMD	Find potential bugs in Java by static analysis
WSDL Test Gen	Generate test cases from WSDL
WS Test Executor	Execute tests generated by WSDL Test Gen

Table II
NUMBERS OF SEEDS AND MUTANTS

	#Seeds	#Mutants	Total
Service	11	460	471
Rule	40	2049	2089

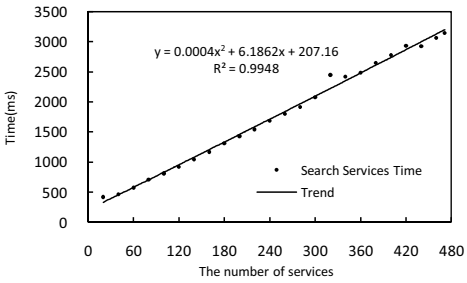

Figure 4. Search for Services in Registries of Different Sizes

1) Scalability w.r.t. the number of services: In this experiment, we fix the knowledge-base and a set of service requests, but vary the number of services registered to the registry (referred to as registry size in the sequel) to study how the number affects the performance of the broker.

In particular, the registry size varies from 20 to 471. Each registry contains 11 seeds plus a subset of their mutants selected at random. Given a set of such service descriptions, we form a registration state of the system. To alleviate the fluctuation of the system performance brought by the internet connection, we carried out the experiment in a relatively stable internet environment. Moreover, in each state of the system, the broker is run repeatedly for 30 times. The average execution time is calculated.

The experiments results show that the average search time increases with the number of services in the registry, but in almost a linear manner; see Fig. 4. According to this trend, the broker will only take a few seconds to search for a service even if there are thousands of services available. Thus, it is scalable with respect to the registry size.

2) Scalability w.r.t. the size of knowledge-base: In this experiment, we fix the set of registered service and service

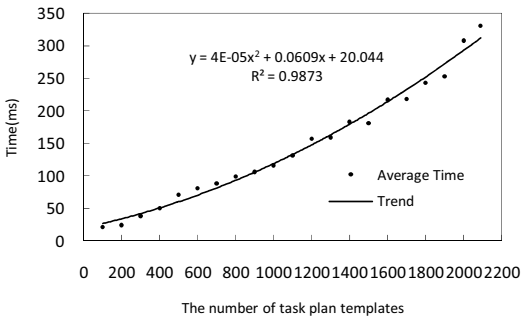

Figure 5. Time Spent on Producing Plans

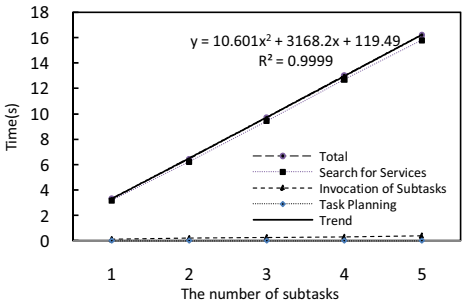

Figure 6. Processing Tasks of Different Complexities

requests, but vary the number of rules in the knowledge-base (referred to as knowledge size) to study how the number affects the performance of the broker. In particular, the knowledge size varies from 100 to 2089. Each time the knowledge-base contains the seed rules plus a number of randomly selected mutants of the seed rules. Because the broker tries the rules in the knowledge-base sequentially, we place the mutants on the top of the real ones to ensure that the longest execution path that the planner will exercise. With each such populated knowledge-base, we execute repeatedly the broker on each service request for 20 times and the average execution time is calculated.

The results of the experiment show that when the size of knowledge-base increases, the time spent on producing service composition plans increases in a quadratic polynomial rate; see Fig. 5. We believe that in reality, the knowledge-base can hardly reach the scale of 2000 rules. Even though, the time spent on producing a service composition plan is only about 300 mini-seconds. Therefore, it is scalable with respect to the size of the knowledge-base.

3) Scalability w.r.t. to task complexity: In this experiment, we fix the set of services and knowledge-base, but vary the service requests to study how the complexity of service requests affects the performance of the broker. In particular, we use the original knowledge-base and the whole set of 471 seed and mutant services. The service requests were classified into 5 subsets. Each subset contains test requests of the same complexity. Here, the complexity is measured by the number of different kinds of subtasks that the service request should be decomposed into. Again, the broker is executed repeatedly on each service request for 30 times and the average execution time of the broker is calculated. Fig. 6 shows the average total execution time for processing service requests of different complexities. A quadratic polynomial figure fits very well the curve with $R^2 = 0.9999$.

It is worth noting that, the variation range of task complexities is usually very small, say up to 10 different kinds of subtasks. Therefore, in the extreme cases, the broker can process a very complicated request within a minute. On average, a test request usually has 2 or 3 different kinds of subtasks. In such cases, the broker only takes less than 10 seconds. Therefore, it is scalable with respect to the complexity of requests.

IV. Conclusion

This paper presented a broker-based approach to service composition. A prototype framework is reported. It is capable of generating service composition plans dynamically by decomposing service requests into a number of subtasks according to the knowledge of workflow in the application domain and the services existing in a semantic web service registry. Experiments with the broker have been conducted to evaluate its scalability, which clearly show that the broker is capable of processing service requests with satisfactory performance when a large number of services registered and the knowledge-base used to generate service composition plans are populated. Thus, it is scalable for practical uses.

A. Comparison with Related work

A great amount of research effort has been reported in the literature on automatic service composition [8]–[10]. Existing works fall into two types, the workflow approaches and planning approaches. The current achievements in the former approach include standardized workflow definition languages and implementation of workflow execution engines such as BPEL4WS, and OWL-S, etc. Due to its flexibility and computation power, planning approaches have attracted increasing interests in recent years. Among the most well-known are those adapting Golog [11], SWORD [12], AIMO [13], SHOP2 [14], PORSCE II [15], OWLS-XPlan [16], etc. However, existing planning approaches suffer from the scalability problem. In general, planning is NP-complete. As shown in [17], the time needed to convert OWL-S service descriptions into planning domain definitions is exponential to the number of services. Many AI planners failed to generate plans for problems of low and mid range complexity. Moreover, AI planners tend to generate unnecessarily long composition plans [17].

Our approach differs from existing workflow based approaches in that we treat workflow as domain knowledge and use it to generate service plan dynamically, rather

than encoded statically. Such knowledge is represented in the form of task decomposition rules, which is similar to the input to HTN (Hierarchical task network) planners. However, our approach is different from existing HTN based WS planning approaches, such as SHOP2 and SWORD, where the knowledge of workflow is simply converted from OWL-S and plans are mostly generated by a brutal force of inference. Our rules are at a higher level of abstraction and more general. Planning is completed in two stages. At first, we generate an abstract plan of the activities to be performed without determining the specific services that carry out the actions. Then, the registry is searched to make a concrete plan that has the full details of the services to be invoked. This significantly reduced the search space of planning. Our preliminary experiments show that this is scalable and efficient. Our approach is also different from the planning approach by recognising the need of collaborations among the services. We provide facilities to support such collaborations as the workflow approaches do, which is neglected by planning approaches. This is particularly important for complicated services.

One of our main contributions is that we encapsulate the domain knowledge of workflow and the capability of planning, service discovery, invocation and collaboration into brokers. We enable such brokers to collaborate with each other in the same way as with other services. This is actually a multi-agent problem solving architecture rather than a centralised facility. Consequently, the search space of planning is decomposed into subspaces according to the application domains. This is one of the reasons why our approach is scalable while existing planning approach is not.

B. Future work

For future work, we are considering alternative implementations of the framework to improve the performance and capability of the I-broker prototype. It is desirable to enable the users to write rules in a notation at a higher level of abstraction rather than as XML files. We are also developing a knowledge-base manager to support the writing, updating and testing the knowledge-base. A more powerful and complicated rule language is also under research. Existing service brokers in the literature often have the functionality of keeping the track record of QoS of WS. Such functionality can be easily incorporated into our framework.

ACKNOWLEDGEMENT

The work reported in this paper is partly supported by the National Basic Research Program of China (Grant No. 2011CB302603) and the National Natural Science Foundation of China (Grant No. 60725206).

REFERENCES

[1] K. Sycara, M. Paolucci, A.Ankolekar, and N. Srinivasan, "Automated discovery, interaction and composition of semantic web services," *J. Web Semantics*, vol. 1, no. 1, pp. 27–46, 2003.

[2] K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan, "Dynamic discovery and coordination of agent-based semantic web services," *IEEE Internet Computing*, vol. 8, no. 3, pp. 66–73, May/June 2004.

[3] T. Koponen and T. Virtanen, "A service discovery: a service broker approach," in *Proc. of ICSS'04*, 2004, p. 7.

[4] X. Bai, S. Lee, R. Liu, W. Tsai, and Y. Chen, "Collaborative web services monitoring with active service broker," in *Proc. of COMPSAC'08*, 2008, pp. 84–91.

[5] H. Zhu and Y. Zhang, "Collaborative testing of web services," *IEEE Transactions on Service Computing*, In press, available at http://cms.brookes.ac.uk/staff/HongZhu/Publications/TR-DOCE-AFM-2010-02.pdf.

[6] L. Shan and H. Zhu, "Generating structurally complex test cases by data mutation: A case study of testing an automated modeling tool," *The Computer Journal. Aug. 2009; Vol. 52, No. 5:*, vol. 52, no. 5, pp. 571–588, 2009.

[7] Y. Zhang and H. Zhu, "Ontology for service oriented testing of web services," in *Proc. of SOSE*, 2008, pp. 129–134.

[8] J. Rao and X. Su, "A survey of automated web service composition methods," in *Proc of SWSWPC'04*, 2004, pp. 43–54.

[9] S. Dustdar and W. Schreiner, "A survey on web services composition," *Int'l Journal of Web and Grid Services*, vol. 1, no. 1, pp. 1–30, August 2005.

[10] M. ter Beek, A. Bucchiarone, and S. Gnesi, "Web service composition approaches: From industrial standards to formal methods," in *Proc. of ICIW'07*, 2007, p. 15.

[11] S. Mcilraith and T. C. Son, "Adapting GOLOG for composition of semantic web services," in *Proc. of ICKRR'02*, 2002, pp. 482–493.

[12] S. R. Ponnekanti and A. Fox, "SWORD: A devel-oper toolkit for web service composition," in *Proc. of WWW'02*, Honolulu, HI, USA, 2002, pp. 83–107.

[13] S. Tabataei, W. kadir, and S. Ibrahim, "Automatic discovery and composition of semantics web services using AI planning and web service modeling ontology," *Int'l J. of Web Service Practices*, vol. 4, no. 1, pp. 1–10, 2009.

[14] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN planning for web service composition using SHOP2," *Journal of Web Semantics*, vol. 1, no. 4, pp. 377–396, 2004.

[15] O. Hatzi, D. Vrakas, N. Bassiliades, D. Anagnostopou-los, and I. Vlahavas, "The PORSCE II framework: Using AI planning for automated semantic web service composition," *Knowledge Engineering Reviews*, 2010.

[16] M. Klusch and A. Gerber, "Semantic web service composition planning with OWLS-XPlan," in *Proc of SASW'05*, 2005, pp. 52–66.

[17] ——, "Evaluation of service composition planning with OWLS-XPlan," in *Proc. of WI-IATW'06*, 2006, pp. 117–120.