International Journal of Computer Applications (0975 – 8887)
Volume 13– No.8, January 2011

Algorithm for Web Service Composition using Multi-Agents

G. Vadivelou
Research Scholar, Department of Computer Science and Engineering
Bharathiyar University, Coimbatore, Tamil Nadu, India

E.Ilavarasan
Department of Computer Science and Engineering
Pondicherry Engineering College, Pondicherry, India

S. Prasanna
Department of Computer Science and Engineering
Pondicherry Engineering College, Pondicherry, India

ABSTRACT

The term Web Services describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. Merely providing services to the users in the heterogeneous distributed environments, service oriented systems are very important. Most of the time, the individual services do not have the sufficient conditions to provide any services to the users. In order to resolve the aforementioned problem, one may compose several individual services together. The proposed method currently applied to ensure the robustness and dependability of Web Services compositions do not effectively map to more open dynamic environments. The proposed approach has a new multi-layer Web Services composition model based on Multi-Agent System. We propose a different self-adaptive mechanisms corresponding to different environment's evolutions to improve the reliability of Web Services composition efficiently. Also we introduce an algorithm for dynamic approach to select the best composition. This composition is selected based on the quality and the compose-ability of participated services. Advantage of the proposed approach is to recognize the feasibility of the composition process at any point of execution and produce better throughput and a less consumption of memory to select composition of services dynamically.

General Terms: Information systems, Algorithm.

Keywords: Agents, Web Service, Web Service Composition

1. INTRODUCTION

Web Services are considered as self-contained, self describing, modular applications that can be published, located, and invoked across the Web. Amount of products and services available now on the Web increases dramatically and goes beyond user's ability to analyze them efficiently. At the same time the number of potential customers available via the Internet also increases significantly and starts to be beyond service providers' ability to perform efficient targeted marketing. Another important issue related to the development of Web services is their integration and composition. Recent progress in the field of Web Services has made it practically possible to publish, locate, and invoke applications across the Web. This is a reason why more and more companies and organizations now implement their core business and outsource other application services over the Internet. In particular, if no single Web service can satisfy the functionality required by a user, there should be a possibility to combine existing services together in order to fulfill the request. The challenge is that Web services can be created and updated on the fly and it is often beyond human capabilities to analyze the required services and compose them manually. The complexity of selecting and composing Web services descends from the following two sources [1]: 1) it is not always easy to define selection criteria for a Web Service; 2) Web services can be developed by different organizations, which provide different offers, so, the ability of efficient integration of possibly heterogeneous services on the Web becomes a complex problem (especially for dynamic composition during runtime).

The remainder of this paper is organized as follows. In the next section, we will introduce the basic concepts such as Web Service and Agents [2, 3]. Section III describes our related work. Section IV discusses about proposed frame work. Section V discusses about the implementation details and finally, the paper concludes with the future work in Section VI.

2. BACKGROUND

2.1 Web service

A Web Service is an accessible application that other applications and humans as well, can automatically discover and invoke. An application is a Web Service if it is [1]: (i) independent as much as possible from specific platforms and computing paradigms; (ii) developed mainly for inter organizational situations rather than for intra-organizational situations; and (iii) easily compos able (i.e., its composition with other Web services does not require the development of complex adapters)

Web Services are, in practice, transient and stateless processes that exist only during service execution, which is triggered by a request coming from a consumer, or client. Services are instantiated to perform specific tasks, thus facilitating scalable, concurrent service provision.[2, 3] The design of a Web Service is usually defined as a clearly articulated workflow, for the sake of reliability and quality of service.

2.2 Agents

An Agent is a piece of software that acts autonomously to undertake tasks on behalf of users [4, 5]. The design of many Agents is based on the approach that the user only needs to specify a high-level goal instead of issuing explicit instructions, leaving the how and when decisions to the agent. An SA exhibits a number of features that make it different from other traditional components [6] including autonomy, goal orientation, collaboration, edibility, self-starting, temporal continuity, character, communication, adaptation, and mobility.

Agents are one of the important contributions of Artificial

Intelligence about the nature of computing [4, 7, 8] . Agents are software entities which interact with an environment, and are subject to modify themselves and evolve according to both external and internal stimuli, the latter due to the proactive and deliberative capabilities of agents themselves. Agents are problem-solvers and may have reasoning abilities[9, 10, 11]: therefore, they react how and when they deem it appropriate. Reaction may imply performing actions to affect the environment. Agents are proactive in the sense that, according to past experience and internal reasoning, they are able take initiatives that may imply performing actions, but also setting an objective (or “goal”) and constructing and executing a plan to achieve that goal. Agents work is usually based upon a background knowledge base composed of “beliefs” [12, 13, 14]. Agents can be to some extent “intelligent”, which from an observer’s point of view means that agents are able to exhibit a flexible and adaptive behavior. As a pretty natural consequence of previously-mentioned features agents are autonomous, i.e., they are able to inhabit an environment and control their own behavior independently of external influence. Autonomous agents are typically state full and persistent.

2.3 Web Service and Composition

Normally, service providers advertise their service in a common market. Some of them claim similar functionality, which are called semantically equivalent. When a composition process is executed, how to select the best suitable one from several candidates is important. It is envisaged that some of them are not suitable judging by QoS. The service composition process must make balance from different perspective. It is useful to make the composition process transparent to requesters. Typically, there are various different composition candidates. A good composition model should be efficient to solve the requirement by automatically composing service advertised by different providers.

Thus, in this paper, Web Service Composition can be seen as a process to find a new service S, which consists of a set of component web services {S_1, S_2, S_3…S_n}. Each component web service is mapped to a set of real web services {S_{i1}, S_{i2}, S_{i3}…S_{im}}, which we can say are semantically equivalent. Figure 1 is an example of these four relationships, where S_1, S_2 … S_6 are component web services and S_{21}, $S_{22}$are real web services that can be invoked at run-time.

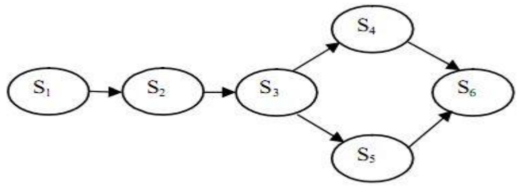


Figure 1: Web service composition

3. RELATED WORK

Service composition has been the subject of many research projects, such as the Ninja project [3] and SAHARA [8].which includes specifications for WSDL, SOAP and other protocols that may be used to describe, access, execute, and discover services on the Web. There are several works on incorporating agents into Web Service systems. In particular, Gibbins et al [2] demonstrated usage of DAML-S for Web Services descriptions within agents. Another step towards incorporating Web Services into agents is proposed by Ardissono et al [1]. Since their focus has been set to non-symbolic negotiation, their work could be seen as a complementary part to our work, where we focus on logic-based Web Services Composition. Sirin et al [9] presents a semi-automatic method for Web Services Composition. The main difference between our approach and the above-mentioned methods is that we propose a unified solution to Web Services Composition problem.

Zakaria Maamar [14] develops a service composition framework, in which multiple-agent-system that composes of composition agent, service agent and service instance agent is the engine of service composition. During the composition process, software agents engage in conversations with their peers to agree on the Web Services that participate in this process. Conversations between agents take into account the execution context of the Web Services. But this paper doesn’t consider context aware service.

In [6, pg 575], Marinescu discusses the use of the Bond agent architecture to enact a workflow description captured in XPDL. Most closely related to our vision of using contemporary BPM tools and Web services for multiple agent system design is the work described in [5]. In this paper, Korhonen, et al. describes the creation of a workflow ontology that is used to describe both agents and Web services. They hope to build a workflow enactment mechanism that can utilize the ontology to bridge the communications gap between agents and Web Services.

4. PROPOSED FRAMEWORK

We proposed a framework to support negotiation during QoS aware Web Service Composition. We add two layers between service requirement and web service candidates, as shown in figure 6. In the first layer, each Web Service candidate is linked to a home norm base, which can be used by home agent to negotiate among other home agents. Norms are a set of rules and regulations, an under-lying protocol governing the agent communications network and agents complex behavior. Norms revolves around agents, which influences the agents to execute a series of concerted actions to achieve a particular goal. And the second layer is composition algorithm which links with the home agent for choosing the best services for the composition.

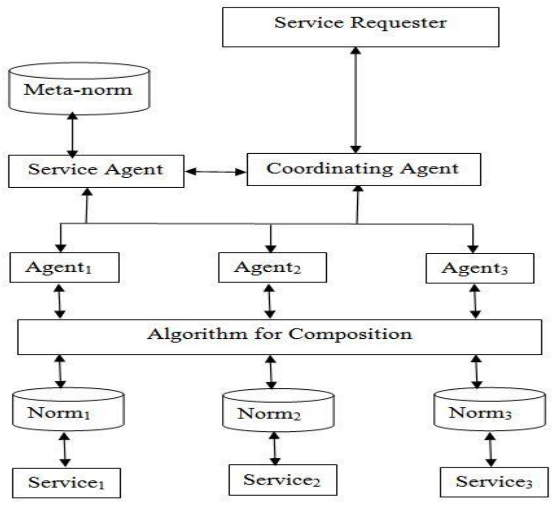


Figure 2: Proposed frame work

4.1 Home Agent

The main function of discovery agent can be divided into two modules:

Service equalizer model: Service matching is based on the similarity between service descriptions that mainly contain service name and other related information.

Service selection module: By the use of selection algorithm, discovery Agent selects one optimum service or a group of services.

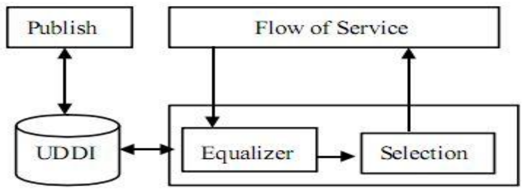


Figure 3: Home agent models

Algorithm for Composition

```
Past = NULL; Now = NULL;
Create_node(nb, B); Create_node(nf, E);
Foreach n in Tasks do {
        Past = Now; Now = get_cm($TR_n$);
        Foreach CM in Now do {
                If n = 1 then CMP = {B}
                else CMP= match (CM, Past);
                if CMP is not NULL then {
                foreach s in CM do {
                create_node(ns, s);
                foreach sp in CMP do
                if n= k – 1 then
                add_edge(ns, nf, 0);
                }
                remove(CMP, Past);
        }       else
        remove(CM, Now);
   }
if Now is NULL then
        throw(“It is not feasible!”);
}
```

4.2 Service Agent

Service Agent has four function modules:

Agent conversation: Agent conversation involves two parts of conversation. One is the conversation between service agent and composition agent. The other is the one between service agents.

Access control: Controlling other agent’s access to the service.

Service invoker: Invoking the method of Web Services.

Service adaptation: In the process of service invoking, service agent regulates the methods of Web Service according service context.

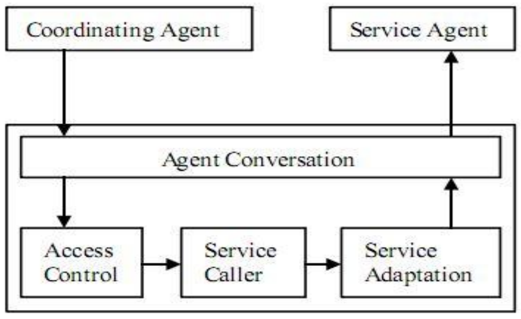


Figure 4: Service agent model

4.3 Coordinating Agent

The main function of composition agent is divided into four sub-modules:

Divergence in Flow: The service composition flow is divided into some sub-flows that are performed by service agent.

Agent conversation: Service Agents that used to perform sub-flow are determined by the conversation of composition agent.

Execution: The module is responsible for controlling and regulating service composition.

Result Collection: Composing agent is responsible for collecting the result of service composition and returns the result to user application.

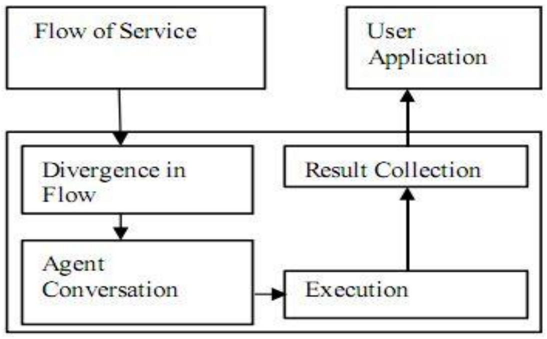


Figure 5: Coordinating agent model

4) Meta norm repository: Meta-norms are a special type of norms, which are same as normal norms but service agent use them to guide home agent’s internal norm update.

4.4 Composing of Services

Service oriented systems are functioning based on services.

Composing of the services can make a value added service. These services provide service to the clients (A person or a software module that gets benefits from services of servers) [7]. A client may find its appropriate services from the metadata information of the services. Of course, by increasing the number of services and by expanding the space of system environment, it is better to register the services through a service broker. Moreover, having a more speed and a better property of services are the reasons that a client may use a service broker. Each service has three properties as follows: the quality, the interface and the functionality. The quality of each service depends on functional and non-functional requirements. In a service, finding a proper output is called the functional requirements. While the speed of the computation of finding this output is called a non-functional requirement. Based on some criteria, a client gives score to each satisfied requirements. As the result, this client may accept the immediate first output or wait for a longer time to find a more proper output. In order to compare the quality of service, we need to transfer the quality of the service into the quantity values. We measure the total quality of service (QoS) by using the total points that a client assigns to each satisfied requirement [7]. We calculate this quantity (QoS) as follows:

$$QoS(s) = \sum w_i * Score_i(s)$$

In the above formula, client assigns score w_i to the i^{th} satisfied requirement and service s satisfies this requirement with quality $Score_i$. Interface is another property of the services. Each service has two types of interfaces: Input and output. Input interfaces indicate the input parameters and output interfaces indicate the output parameters. Two services are called match if output parameters of one of them is the same as the input parameters of the others. A community is a set of services, which are matched together. In order to increase the usability of services most of the time we design services in such a way that the functionality property of service could solve the problems, which are simple and basic. Software developers select the proper composition of services to solve problems that are more complex. By executing each of these compositions through a specific process, they can reach to the solution. This process is determined during execution plan associated to the problem [7].

4.5 Web Service Composition using Graph

We introduce a method that considers each service as a graph node. With attention to order of tasks in execution plan, we connect nodes as the edge of the graph. After we create the graph, by searching in the existed paths we can find the best path. The set of services of that path declares as the best composition. All the proper services for each task in the execution plan are requested from the service broker. The service broker presents these services in the form of different communities. Figure 6 shows an instance of the execution plan. Tasks are presented as ellipses on the top of this figure and the related candidate services are shown in column under each task. Services in a community are besieged in the same rectangles. As mentioned earlier, we create the nodes of the graph from some of the services. To connect these nodes we use the weighted directed edges of the graph. In Figure 6, the label near each edge is the weight of that edge respectively.

Algorithm for optimal composition introduces the graph creation operations completely. In this algorithm and for each task, first we receive all candidate services. Then we insert these services into a list in form of the sets of the same community services. We name this list as Now list (For example, the broker introduces services s31, s32, s33 for task t3 in Figure 6).

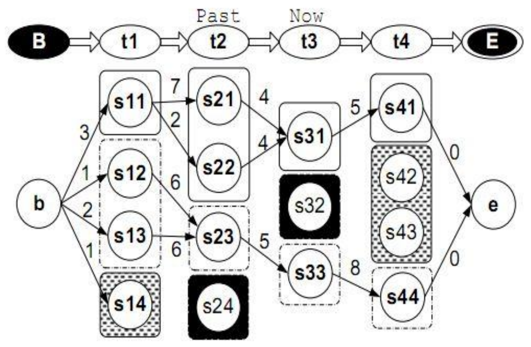


Figure 6: Execution plan and candidate services

In the next step, we study each community in the created list to figure out whether it has any matched services in the list of previous task (Which is called Past) or not. If such any services exist (e.g. community of s31), we connect all services in the two communities together two-by-two (e.g. edges (s21, s31) and (s22, s31)). Then we assign the quality of destination service (that is in the Now list) to these new edges. In order to avoid the superfluous searches, we delete that community from the Past list. If we cannot find any proper community for a service in the list of previous task (e.g. the community that has services s42 and s43 in Figure 6), we delete that community from the list of studying task. The reason for this deletion is that we do not need to search this community again, when we start studying the next task. These deletions lead decreasing of additional searches so that we can create and search graph with a better speed.

In summary, at the beginning of the execution for task n, the Now list has all candidate services related to this list. Moreover, the Past list has services from task n-1 that has at least one matched service in the services of task n-2.1 For example, if we are at the beginning of the execution for task t3, the Now list has services s31, s32 and s33 and the Past list has services s21, s22 and s23. (at this point of execution, we do not know any services related to the next tasks). Hence, we do not create any graph node from services that have not any matched services in the previous task. This operation leads to the less memory consumption as well. After executing the algorithm, the final graph has all drawn edges and services in Figure 6 (as nodes) except services s24, s32, s42 and s43. In some situation, however, we may not be able to create any composed service from the services in the system environment. In this case, the Now list would be empty at the end of the execution for that task. Therefore, we may inform the user that the composition is not feasible and then we terminate the execution. In the Algorithm, each function is described with these operations:

- create_node (nodeName, service): Creates a graph node with assigned service and called it as nodeName.
- get_cm (TRn): It requests all services from service broker that can do the task n. Service broker places all that services as a set of communities of services and deliver to the requester.
- Match (CM, Past): It searches in the Past list and return the community that is matched with CM. if there are no

such community, returns NULL as the output.

- add_edge(node1,node2,Q): Connects a directed edge from node1 to node2 and determines its weight as Q.
- remove (community, list): Removes the community from the list.
- throw(message): Terminates execution of the algorithm with showing proper message to the user.

5. IMPLEMENTATION AND RESULTS

The proposed algorithm has been implemented in java using Intel Pentium IV processor with 256 MB RAM and operating system Windows XP. In addition to the all improvements that we mentioned in the earlier sections, the proposed approach does huge improvements on the time and the memory consumption compare to the other methods in the literature (e.g. [15] and [16]). As an experimental evaluation, we implement both the proposed approach in this paper and the latest work in this area ([14]). To do this comparison, for both of approaches, we calculate the time and the memory consumption with the same input at the execution time. In order to compare time and memory consumption of these approaches, we may consider the varieties in the number of tasks, the number of services and the number of types of service interfaces during the time. The time of algorithm execution shows the time consuming and the number of nodes in the graph which determines the memory consumption. We calculate these two values for different executions with considering the variation of one of those varieties. These results are illustrated in Figure 8 to Figure 10. Chart "a" in each figure shows the execution times of two algorithms. Chart "b" shows the required memory for their execution respectively; it means that in each chart, the top curve diagram is related to the latest method in the literature and the bottom curve is related to the proposed approach in this paper (Figure 7 shows the legend of these charts).

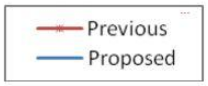
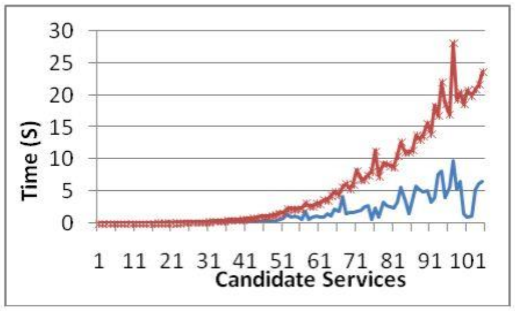
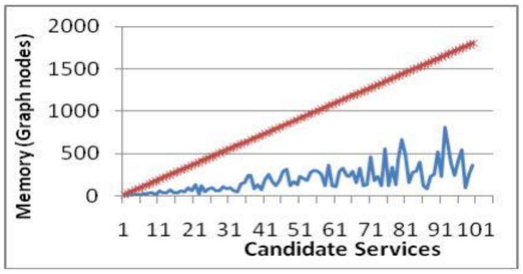


Figure 7: Legend of the charts

In Figure 8, we consider an execution plan with twenty tasks in which the number of candidate services is variable. We randomly select the type of interface for each service from ten available types.



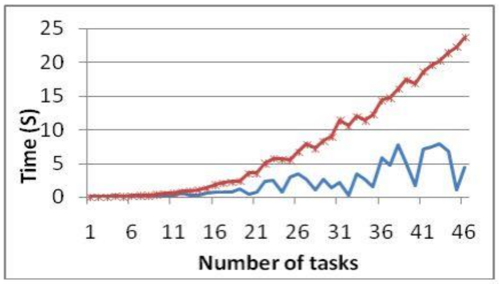
(a)



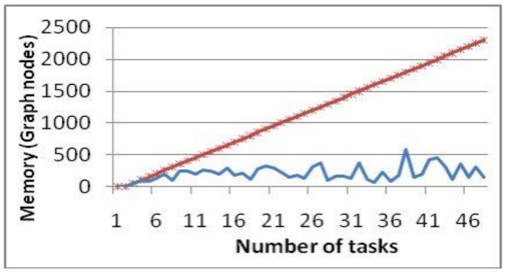
(b)

Figure 8: variety of number of candidate services

In Figure 9, the number of tasks is variable. For each task, we consider fifty candidate services. These candidates choose their interfaces from ten available types.



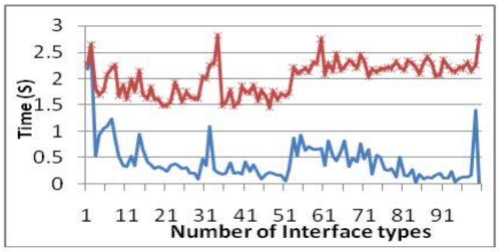
(a)



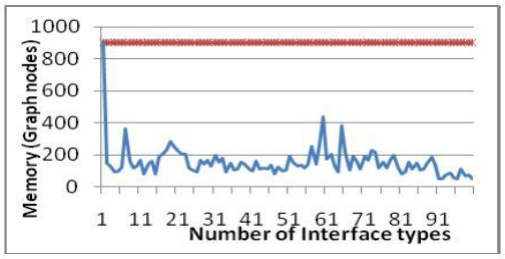
(b)

Figure 9: variety of number of tasks

In Figure 10, we consider an execution plan with twenty tasks and fifty candidate services. Interface of each service is selected from a variety number of types.



(a)



(b)

Figure 10: variety of type of interfaces

6. CONCLUSION

We propose a self-adaptive Web Services Composition model based on Multi-Agent Systems and an algorithm for selecting an optimal composition of services dynamically. Our model aims to support the design, deployment and maintenance of distributed systems by allowing the combination, reorganization and adaptation of services. We encapsulate the Web Services with Agents to make them be more autonomy, reliability and robustness in response to the dynamic environments. The algorithm, we propose first create a graph from candidate services based on the execution plan (with attention to their interfaces). Then we label the best path in the graph as the optimum composed service. Considering variety of service interfaces makes outputs of this algorithm more close to the real life situation. This algorithm shows the feasibility of the composition process at any point of the execution. According to the implementation result, the experimental evaluations show this algorithm has a better throughput and use the less memory consumption compare with other similar approaches. The speedup in the creation and the searching of graph leads to the overall speedup of the execution. The aforementioned improvements decrease the waste of the resources.

REFERENCES

[1] L.Ardissono, A.Goy and G.Petrone. “Enabling conversations with web services”. Proc. of the 2nd Int.Conf on Autonomous Agents and Multiple agent Systems, 2003, Melbourne, pp.819-826.

[2] N.Gibbins, S. Haris and N.Shadbolt. “Agent- Based Semantic Web Services”. In Proceedings of the 12th Int. WWW Conf., WWW2003, Budapest, Hungary, 2003, ACM Press, 2003, pp.710-717.

[3] Gribble, S.D.Welsh, M.von Behren, R.Brewer,E.A. Culler, D.Borisov, N.Czerwinski, S.Gummadi, R.Hill, J.,Joseph, A.D.,Katz, R.H.,Mao, Z., Ross, S., and Zhao, B.“The Ninja Architecture for Robust Internet- Scale Systems and Services”, Special Issue of Computer Networks on Pervasive Computing, March 2001.

[4] N. Jennings, K. Sycara, and M. Wooldridge. “A Roadmap of Agent Research and Development. Autonomous Agents and Multiple-Agent Systems”, 1(1):738, 1998.

[5] J. Korhonen, L. Pajunen and J. Puustijarvi, "Using Web Services and Workflow Ontology in Multiple-Agent Systems", presented at Workshop on Ontology’s for Multiple-Agent Systems, Saguenay, Spain, 2002.

[6] D.C. Marinescu, Internet-based workflow management: toward semantic web. New York; Wiley- Interscience, 2002

[7] McIlraith, S.,Son T., Zeng, H.“Semantic Web Services”. IEEE Intelligent Systems, Special Issue on the Semantic Web, Volume 16, No. 2, pp. 46-53, March/April, 2001.

[8] Raman, B., Agarwal, S., Chen,Y., Caesar, M., Cui, W., Johansson, P., Lai, K. Lavian, T.,Machiraju, S., Mao, Z.M., Porter, G.,Roscoe, T., Seshadr. “The SAHARA Model for Service Composition Across Multiple Providers”, Pervasive Computing, August 2002 Lecture Notes in Computer Science LNCS 2414, Springer, 2002.

[9] E. Sirin and J. Hendler and B. Persia, “*Semi automatic Composition of Web Services using Semantic Descriptions*”, Workshop on Web Services: Modeling, Architecture and Infrastructure in conjunction with ICEIS2003.

[10] Stefania Costantini1, “*Agents and Web Services*”, Dip. Di Informatica, Universit`a di L’Aquila, Coppito 67100, L’Aquila, Italy 10.

[11] S. Thakkar et al. “*Dynamically composing web services from on-line sources*”. In Proceeding of 2002 AAAI Workshop on Intelligent Service Integration, Edmonton, Alberta, Canada, 2002.

[12] Wang, Jiying, “Information Discovery, Extraction and Integration for the Hidden Web”, Proc. VLDB PhD Workshop, 2003.

[13] Zakaria Maamar “Interleaving Web Services Composition and Execution Using Software Agents and Delegation”. College of Information Systems Zayed University Po Box 19282, Dubai, U.A.E.

[14] Zakaria Maamar, Soraya Kouadri Mostéfaoui, and Hamdi Yahyaoui, “Toward an agent-based and context-oriented approach for Web services composition”, IEEE Trans on Knowledge and Data Engineering, Vol 17, No. 5, PP. 686-697, 2005.

[15] Gao, Yan, et al., "Optimal Web Services Selection Using Dynamic Programming." s.l. : Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06) , 2006.

[16] Gao, Yan, et al., "Optimal Selection of Web Services for Composition Based on Interface-Matching and Weighted Multistage Graph." s.l. : Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT’05) IEEE, 2005.