Ontology-Driven Web Services Composition Platform

I. Budak Arpinar, Boanerges Aleman-Meza, Ruoyan Zhang, and Angela Maduko
LSDIS Lab,Computer Science Dept., University of Georgia
{budak, boanerg, ruoyan, maduko}@cs.uga.edu

Abstract

Discovering and assembling individual Web Services into more complex yet new and more useful Web Processes is an important challenge. In this paper, we present techniques for (semi) automatically composing Web Services into Web Processes by using their ontological descriptions and relationships to other services. In Interface-Matching Automatic composition technique, the possible compositions are obtained by checking semantic similarities between interfaces of individual services. Then these compositions are ranked and an optimum composition is selected. In Human-Assisted composition the user selects a service from a ranked list at certain stages. We also address automatic compositions in a Peer-to-Peer network.

1. Introduction

In recent years, a growing number of Web Services (WSs) have emerged as the Internet develops at a fast rate. The Web is now evolving into a distributed device of computation from a collection of information resources [5]. Furthermore, the need for composing existing WSs into more complex services is also increasing, mainly because new and more useful solutions can be achieved. In general, this is a result of complex and increasing user demands and inability of a single WS to achieve a user's goals by itself. Hence, a collection of interacting WSs is more likely to accomplish these goals. Currently, users either utilize some services that they know already or find those services by looking it up in a keyword-based search engine (e.g., expedia.com or google.com) or by looking it up in a WSs registry (e.g., a UDDI – Universal Description, Discovery and Integration registry). Also, composition of discovered services and enabling data-flow among them are usually done manually, which are highly inconvenient, especially for more complex compositions. The problem lies with the fundamental abstractions used to model WSs, and methods to compose these services using these abstractions. In more complex examples of service compositions, even hundreds of data collection services can be involved in a composition (e.g., a search involving gene banks). In those cases, (semi) automatic composition can help in reducing composition generation and also execution time substantially.

Our service composition research aims for reducing the complexity and time needed to generate, and execute a composition and improve its efficiency by selecting the best possible services available at current time. In general, there are four different dimensions for a service composition: (i) degree of user involvement in a composition specification, (ii) whether the composition is based on templates, (iii) dynamicity (i.e., adaptation) of the composition, and again (iv) degree of user involvement in the adaptation of the composition (see Figure 1 - right-most branch is a replica of the tree rooted at *User-defined* node). In the first dimension, a composition can be defined fully by a user including its control and data-flow besides the individual services making the composite service. In contrast, a user is not involved in an automatic composition where the system defines control and data-flow. This is very challenging due to difficulties of mapping user needs to a collection of correlated services where their interim outputs can satisfy each other's input requirements and final deliverable meets the user demands. Besides that, in both of user-defined or automatic composition techniques either actual service instances or service templates can be used. In the latter, the individual service instances are searched and integrated automatically at execution time for a given plan [4]. In a dynamic composition, the composition itself can be adapted mainly because of (Quality of Service) QoS requirements at run-time. Also, a composition may not be defined at design-time but can be assembled dynamically at execution time. Finally, some hybrid methods such as semi-automatic compositions and semi-automatic adaptations are also possible.

This paper primarily focuses on the automatic composition techniques. These techniques use a WSs ontology and a process ontology to discover semantic

matches among user demands and service outputs besides semantic composability of services. In Semantic Web, interface, capabilities and effects of a service can be encoded in unambiguous, machine understandable form [8] using a WS ontology.

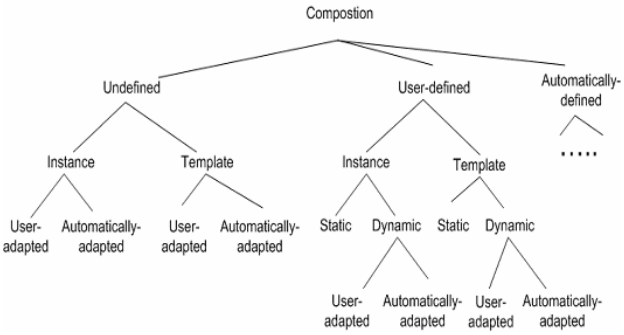

Figure 1. A classification of composition techniques

In Ontology-driven Web Services Composition Platform, the requirements of a composite service are specified by a user as a set of inputs (to the composite service), and a set of expected outputs (from the composite service) [16]. The WS composition aims to generate and execute a composite service that produces the expected outputs by combining existing individual services using their semantic descriptions. We present different approaches including a Human-Assisted (HA) composition, where a human is consulted at certain stages of a composition, where more than one alternative is possible. Interface–Matching Automatic (IMA) composition uses semantic matching of interface definitions of complimentary services to generate a composition.

Currently, descriptions of many WSs are contained in a single central registry, such as UDDI. Using emergent Peer-to-Peer (P2P) computing techniques, this registry can be moved from its centralized nature to a distributed one. With decentralization, problems of availability, reliability and scalability can be addressed. This, for example, can enable service providers to choose any particular registry in which their services would be listed. This might be beneficial in situations where competitors don't want to be listed in the same registry. Thus, we also briefly present a P2P Automatic (PPA) composition technique. The rest of the paper is organized as follows: In Section 2, we review the related research. Section 3 briefly describes the system architecture. IMA, HA, and PPA composition techniques are described in Sections 4, 5, and 6 respectively. Section 7 concludes the paper, and outlines the future work.

2. Related Work

2.1. Web Services Specification

The service specification methods help software systems to capture capabilities of WSs. In general, these specification methods are based on either industry-oriented standardization efforts, or academia-oriented WS ontologies.

UDDI and Web Services Description Language (WSDL) are current industry standards developed for e-commerce. The services are described according to an XML schema, defined by the UDDI specification and registered by the service providers along with keywords for their categorizations. Therefore, a UDDI does not provide a semantic search rather it depends on a predefined categorization of WSs through keywords. In complimentary roles, WSDL and Simple Object Access Protocol (SOAP) describe WSs as a set of endpoints (or ports) operating on messages, and a protocol for exchange of these messages between the services respectively. Also, some other industry standards have been emerging to represent data and control-flow, and transactional properties among a collection of services. Business Process Modeling Language, XLANG, Web Services Flow Language (WSFL), and Business Process Execution Language for Web Services (BPEL4WS) can be mentioned in this category. However, many of the existing approaches for process modeling lack an adequate specification of the semantics of the process terminology, which leads to inconsistence of interpretations of the information. Process Specification Language is an attempt to resolve this issue by providing a process terminology [11].

The semantic approach for WS specifications includes DAML+OIL based DAML-S [1]. DAML-S specifies three main components for each service. A service-profile indicates the functions of the service, the process-model provides the information of how the service works, and the service grounding describes how an agent can access the service. Other techniques try to add semantics to existing services by providing mappings between WSDL, UDDI definitions and domain ontologies (e.g., METEOR-S [13]).

2.2. Web Services Discovery and Composition

WS discovery related work includes [3], which describes how to evaluate a degree of similarity between a service template and an actual service by measuring the syntactic, operational, and semantic similarity. Unlike other discovery approaches on the basis of WS interfaces, [7] explored ways to search services according to the

functionality requirements, and proposed Process Query Language (PQL) to search process models from a process ontology.

The main concept behind service composition is not new in computer science. Earlier, software composition techniques aimed to find a good combination of components that responds to the client specific requirements by matching requested properties with provided properties. One approach for finding a suitable composition is to delegate the responsibility for solving certain requirements posed on a component to other components after fulfilling it partially [14]. Similarly, our system propagates requirements (that are set of a user's expected outputs) to corresponding WSs in an incremental way.

Template-based composition techniques are used in [9], and ICARIS project [15]. SWORD uses a rule-based expert system to determine if a plan of composite service can be built out of existing services [10]. In [2], WSs are declaratively composed and then executed on a P2P environment. However, only the executions of the static composite services are carried out on a P2P environment, not the composition itself. [12] and [6] present different P2P infrastructures for WS discovery.

3. An Overview of System Architecture

A Semantic WS is a unit of composition that can be deployed independently, and may be subject to composition by a third party on the Web [17]. At the same time, it is defined, and advertised in a machine-processable form through an ontology so it can be automatically discovered, composed, and invoked in new and complex Web processes. A WSs ontology describes the interfaces of the services and the relationships among them. Like domain ontologies, a service inherits properties and functionality of its parent service in WSs ontology. In our platform we use the DAML-S WS ontology, and complement it with a process ontology describing process-oriented features of services. The latter includes relationships among services such as semantic matches among the input and output parameters of services.

A composite service query is represented in a very similar way as a service description in DAML-S. The query includes the interface of the expected composite service, in which a user can define output parameters, output constraints, input parameters, and their constraints.

The system architecture is composed of two categories of components: querying and composition components, and ontology and service storage components as illustrated in Figure 2. When a service with DAML-S description sends a registration request, a *service extractor* retrieves the information from the service profile and stores it in a services database. If the services are described in UDDI schema, their profile would be sent directly into a UDDI registry. The remaining components are responsible from analyzing the query, searching and composing the WSs. Once the *process composer* generates data and control-flow for a process, it sends them to the *process execution* component.

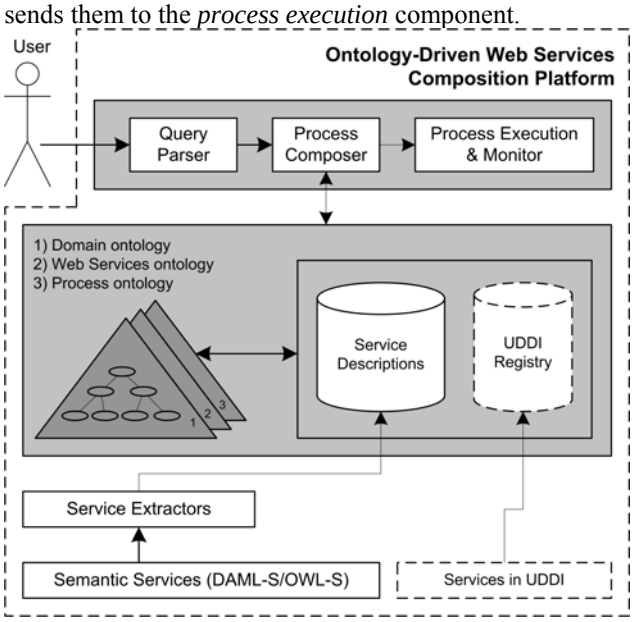

Figure 2. An overview of system architecture

4. Interface-Matching Automatic (IMA) Composition

IMA composition technique aims for generation of complex WS compositions automatically. In IMA, individual services placed earlier in the composition should supply appropriate outputs to the following services in an orchestrated way similar to an assembly line (i.e., pipe-and-filter) in a factory so they can accomplish the user's goals.

In IMA, WSs and process ontologies are navigated to find the sequences starting from the user's input parameters and go forward by chaining services until they deliver the user's expected outputs. The composition starts from the service that needs one or more of the input parameters given by the user. If this WS does not produce all of the expected outputs, more WSs need to be found to provide the expected outputs. This algorithm aims to find a composition that has the best QoS (e.g., shortest execution time) and the best matching of input and output parameters.

For example, in a simple scenario a user inputs a *seafood* type, and expects matching *wine prices* (Figure 3). Both *Wine* (*S1*) and *World Wine* (*S2*) services are

food-wine matching services which output the name of matching wines with the food type given by the user. *Wine Price Information* (*S3*) and *Beverage Price Information* (*S4*) services provide the prices of corresponding wines or beverages.

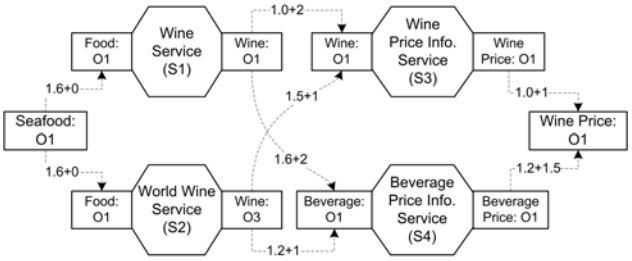

Figure 3. Seafood-based wine price composition

In the process ontology, interface relationships of services can be represented as a directed graph by matching input and output parameters (see Figure 3). Nodes represent services and edges connect services if the output of a service can be "feed into" the input of another service. Dashed-line edges represent parameters that are semantically equivalent (but not an exact match). We consider four cases to check equivalence (i.e., similarity matching) of an output and an input parameter: (1) if they are same, their similarity is maximal. For example, the output parameter of *S1* matches exactly with the input parameter of *S3* (smallest similarity value is 1.0). (2) If an output parameter of the former service subsumes the input parameter of the succeeding service, this is the second best matching level. For example, the output of *S4* (*beverage price*) subsumes the expected output parameter (*wine price*). The similarity value depends on a normalized distance between these terms in the ontology [16]. (3) If the output parameter of the former service is subsumed by the input parameter of the succeeding service, the requirements of a parameter could be partially satisfied. That applies to the relationship between *S1* and *S4*. (4) When two parameters have no subsumption relation or they are from different ontologies (e.g., output and input of *S2* and *S3* respectively, which are defined in ontologies *O3* and *O1*), the similarity value can be obtained by using Tversky's feature-based similarity model [3], which is based on the idea that common features increase the similarity of two concepts, while feature differences decrease the similarity.

The overall edge weights are calculated using a function of QoS (e.g., execution time of the source service of an edge and semantic similarity value between input and output parameters. As a matter of fact, other factors can be considered in computing weight of edges, such as reliability, security, and other properties of services. Relative weights of these factors (λ) can be defined by a user as follows:

$$W = (\lambda) * \text{execution time} + (1-\lambda) * \text{similarity value}.$$

For example, the weight of edge <*S2*,*S4*> is 1.2 + 1 in Figure 3, which means that similarity value of *Wine*: *O3* and *Beverage*: *O1* is 1.2 and execution time of *S2* is 1 unit.

We use a shortest-path dynamic programming algorithm based on Bellman-Ford's algorithm to find the shortest sequence between two special nodes representing initial and final services. In a traditional directed graph, only one incoming edge and one outgoing edge can be selected for every node in the shortest path. However, services may need more than one incoming edge as input parameters in a composition. Furthermore, a service can be executed only when all the required input parameters are available. Therefore, the distance of every node is determined by the maximum value of weights of all the incoming edges into a service node. In a different case, when there is more than one alternatives for one input parameter of a service then we choose the minimum weight as a distance associated with this input parameter. The algorithm running time is $O(n^3)$, and if λ is set to different values by the user, different shortest paths can be obtained in IMA composition technique [16].

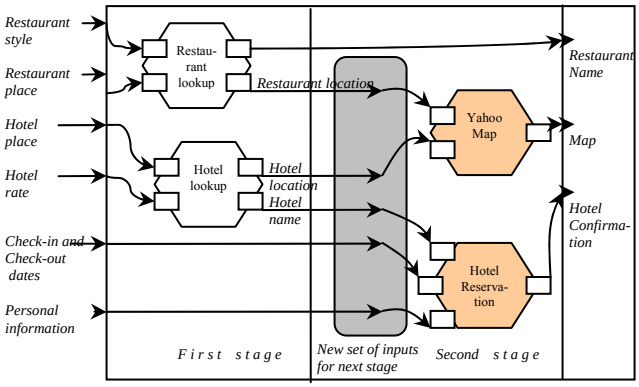

Figure 4. Stages of an interactive composition

5. Human-Assisted (HA) Composition

The goal of HAA composition is to help users in selecting appropriate WSs, and build a composition incrementally. The first step is to consider all inputs by semantically matching them with all WSs that take one or more of them as input. A list of these WSs is provided to the user with the WSs ranked based on their similarity matching score. The user can select WSs s/he considers best for the desired composition. Furthermore, to facilitate a better selection process, each listed WS includes a description of its functionality as well as the output(s) it produces. Then the system determines whether all output parameters of the desired composition

are produced by the services selected so far. If that is the case, the composition is completed. Otherwise the interactive composition process continues with more stages. Figure 4 shows the first stage of a composition for a trip arrangement, where the user has selected two WSs.

For the second stage, a new set of input parameters is generated by the system. This set includes the input parameters considered and also includes all outputs of the WSs selected by the user. The user is given the option to discard elements in this new set of input parameters that may no longer be needed. The user may also mark some of the new input as "optional". This helps in the ranking of the list of WSs that will be shown to the user in subsequent stages. The list of ranked results is grouped by input parameters to facilitate selection when the list is large. In the example of Figure 4, *Restaurant style*, *Restaurant place*, *Hotel place*, and *Hotel rate* are no longer considered as inputs in next stages. The inputs *Check-in* and *Check-out dates* and *Personal information* are still considered; outputs *Restaurant location*, *Hotel location*, and *Hotel name* from the selected services are now considered as inputs in next stages; and output *Restaurant name* from one of the selected services is no longer considered as input (unless stated otherwise by the user), because it satisfies an output parameter *Restaurant name*. The shaded box highlights the new set of inputs for the next stage, and in this way the composition problem has been reduced. Figure 4 also shows the second and final stage of the composition (only two stages were needed to complete this composition example).

6. Peer-To-Peer Automatic Composition

In a Peer-to-Peer (P2P) network, each peer can provide some WSs, and they can be grouped into communities based on the domain(s) for which they provide services. When a peer provides more than one WS for different domains, it becomes a member of different communities. The structure of the communities adheres to that of the ontology, so that where the ontology is hierarchical, the communities then have a hierarchical structure. Each community has both a master and backup peers. The master peer in each community maintains a list of the masters and back up peers for every other community, and backup peers have a replica of this list.

Figure 5 depicts such an ontology-driven P2P network. This network evolves in two different dimensions: (1) the first dimension is based on the domain for which the services are provided, and (2) the second is based on input-output matching relationships amongst the peers' services (the second dimension is not depicted in Figure 5). The master peers (for (1)) maintain the list of all peers within their communities with the services they provide as well as the input and output parameters they accept and generate respectively. In this way, the network is structured in such a way that peers with similar functionality and data dependencies are grouped together in different dimensions. The following explains how the network evolves in dimensions 1 and 2 (using the example in Figure 5).

(1) Initially there's only one peer (P1) in the network; when P2 contacts P1 to join the network, they determine based on their common ontology, what domains they provide services for, in this case "Flight" and "Hotel" respectively. Since they are the first peers for these domains, they automatically become the master peers and record this in their lists. P3 contacts P2 to join the network, as before, based on the common ontology, P2 determines that P3 is in the "Flight" domain. Since P2 is itself a master of its domain it determines which peer to redirect P3 to, in this case P1. P3 then contacts P1 and then records that P1 is its master. Since P3 is the only other peer in the community besides P1, it is automatically made the back up peer for this domain. This self-organizing network can also handle situations where a peer about to join the network contacts a non-master/back up peers as occurs when P5 joins the network or where peers form different sub-communities within a community (e.g., P6 is a master for "Motel" sub-community of "Hotel" services).

(2) As each peer joins the network, its master queries all the other masters of different communities to determine if there is any peer in their community for which the new peer's output matches any of their input or for which their output matches any of the new peer's input. In the first case, the new peer becomes a "predecessor" of any such peer and in the second case it becomes a "successor" of any such peer. If such a peer exists, the master of the new peer notifies it about this association.

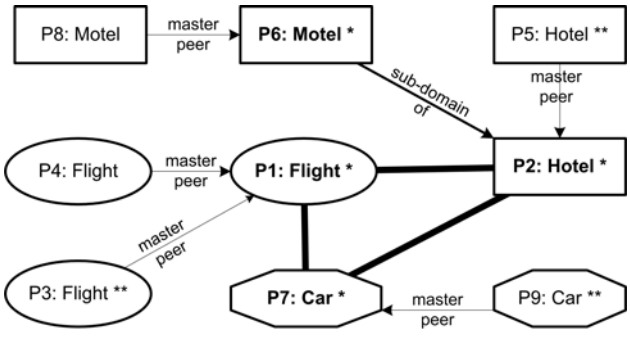

Figure 5. Ontology-driven P2P network. Different shapes indicate different domains; thick lines indicate communication lines between super peers (denoted by *, backup peers are denoted by **).

When any of the peers receives a request by the user for a particular service or service composition, this peer passes this request with the input and expected output provided by the user to its master. The master peer determines based on the common ontology, which communities need to be contacted to process the user's request and forwards it to the master peers of the participating communities. When all the masters of the participating communities have received the inputs and outputs provided by the user, they determine which services in their community provide all or some of the expected outputs for the user and what inputs these services require.

The peers that would participate in the composition are then discovered by traversing a series of predecessor and successor relationships amongst the peers, starting from the peer(s) producing the user's outputs to the peers accepting the inputs (provided by the user) required for the composition. For example, in Figure 6 the user sends the request with input a,b,c,d,e,f and output x,y,z to P13. P13 forwards this request to its master P7, which then determines based on the ontology to forward this request to Peers 1 and 2. These master peers then determine which peers within their community provide some or all of the expected outputs. P2 determines that P10 and P5 would participate in the composition. Since the input of P10 (g) does not match any of the inputs provided by the user, P10 contacts P1, the master of the community to which its predecessor P4 belongs. P1 matches the input of P4 with those provided by the user. In this case, there is a match so this master/slave traversal ends. P7 determines that P15 would participate in the composition. Actual composition then starts with P4, P15, P5 and then P10.

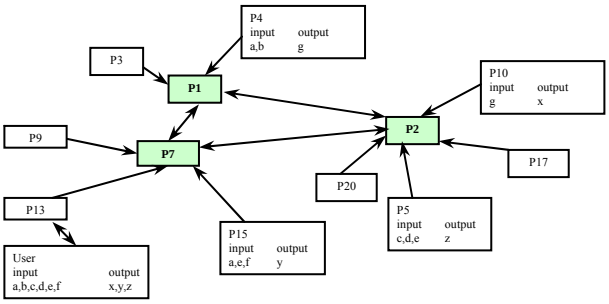

Figure 6. Example of P2P-based composition

7. Conclusions and Future Work

Today's search engines and knowledge discovery tools help users to locate relevant documents and assemble relevant knowledge for effective decision-making respectively, and improve their capabilities continuously using semantics. Similarly, users need new tools to help them discover and assemble services into processes for easier and better quality workflow executions given increasing number and complexity of WSs. In this paper we illustrate some techniques for (semi) automatic composition of semantic WSs. However, some interesting technical problems still lie ahead. For example, users may need to compose services based on their internal computations when their profiles may not convey adequate semantics to differentiate them. We plan to consider validation of the composition based on pre- and post-condition semantics. Besides, a layered (e.g., top-down) composition methodology can help users in an interactive composition.

8. References

[1] A. Ankolenkar, M. Burstein, et al., "DAML-S: Web Service Description for the Semantic Web", *The First International Semantic Web Conference*, Stanford, 2001.

[2] B. Benatallah, M. Dumas, Q. Sheng, and A. Ngu, "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services", *IEEE Intl. Conf. on Data Eng.*, San Jose, 2002.

[3] J. Cardoso, and A. Sheth. "Semantic e-Workflow Composition", *Journal of Intelligent Information Systems*, 2003.

[4] S. Chandrasekaran, J. Miller, G. Silver, B. Arpinar, and A. Sheth, "Performance Analysis and Simulation of Composite Web Services", *Electronic Markets: The Intl. Journal of Electronic Commerce and Business Media*, 13(2), 2003.

[5] D. Fensel, and C. Bussler, "Semantic Web Enabled Web Services", *2nd Annual Diffuse Conference*, Brussels, Belgium, January 2002.

[6] W. Hoschek, "Peer to Peer Grid Databases for Web Services Discovery", *Grid Computing: Making the Global Infrastructure a Reality* Ed(s): F. Berman, G. Fox, and T. Hey, Nov. 2002, Wiley.

[7] M. Klein, and A. Bernstein, "Searching for Services on the Semantic Web Using Process Ontologies", *International Semantic Web Working Symposium*, August 2001.

[8] S. McIlraith, T. C. Son, and H. Zeng, "Semantic Web Services", *IEEE Intelligent. Systems*, March/April 2001.

[9] S. Narayanan, and S. A. Mcllraith, "Simulation, Verification and Automated Composition of Web Services", *11th Intl. WWW Conference*, Honolulu, 2002.

[10] S. R. Ponnekanti, and A. Fox, "SWORD: A Developer Toolkit for Building Composite Web Services", *11th WWW Conference*, Honolulu, 2002.

[11] C. Schlenoff, M. Gruninger, F. Tissot, J. Valois, J. Lubell, and J. Lee, The Process Specification Language (PSL): Overview and Version 1.0 Specification, NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD, 2000.

[12] M. Schlosser, M. Sintek, S. Decker, and W. Neijdl, "A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services", *2nd IEEE Intl. Conf. on Peer-to-Peer Computing*, 2002.

[13] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller, "Adding Semantics to Web Services Standards", *Intl. Conf. on Web Services*, Las Vegas NV, June 2003.

[14] I. Sora, and F. Matthijs, "Automatic Composition of Software Systems from Components with Anonymous Dependencies", *Technical Report CW 314*, Leuven, Belgium, May 2001.

[15] V. Tosic, D. Mennie, and B. Pagurek, "On Dynamic Service Composition and its Applicability to E-business Software Systems", *Workshop on OO Business Sol. ECOOP*, Budapest, Hungary, 2001.

[16] R. Zhang, I. B. Arpinar, and B. Aleman-Meza, "Automatic Composition of Semantic Web Services", *Intl. Conf. on Web Services*, Las Vegas NV, June 2003.

[17] R. Zhang, Ontology-Driven Web Services Composition, MS Thesis, Department of Computer Science, University of Georgia, April 2004.