# Web Service Composition to Facilitate Grid and Distributed Computing: Current Approaches and Future Framework

**Muhammad Ahtisham Aslam, Sören Auer**
Betriebliche Informationsysteme
Universität Leipzig, Germany
aslam@informatik.uni-leipzig.de, auer@informatik.uni-leipzig.de

**Jun Shen**
Faculty of Informatics
University of Wollongong, Australia
jshen@uow.edu.au

**Michael Herrmann**
DaimlerChrysler AG, Sindelfingen Germany
michael.hm.herrmann@daimlerchrysler.com

## Abstract
As long as Web Services are getting into the heart of the growing e-business world, numbers of services available on the Web are increasing rapidly. With such a growth of available services, it is also becoming more and more difficult to discover these services manually. After manual discovery, manually composing these services to perform complex tasks is also a non-flexible and inefficient approach. Sometimes unavailability of a single service within Web services composition can cause a crash of the whole composite process in a distributed computing environment, even though some other services exist in UDDI registries that can be used to perform the same task. This is due to the lack of systems and approaches that can be used to dynamically discover and compose these alternate services on the fly. Therefore dynamically discovering and composing required Web services at run time is preferably needed in the service oriented e-business world. In this paper we provide a survey of existing dynamic and automated Web services composition approaches. We highlight limitations of these existing approaches and propose a new framework at an abstract level for dynamic and automated composition of Web services, especially in grid and distributed computing environment.

## 1. Introduction
Dynamic composition of Web services is highly needed by the growing e-business world to automat the process of Web services interaction. To perform complex business tasks, composition of Web services will remain inefficient and unreliable, as long as Web services are discovered and composed manually.

Composing Web services on the fly can efficiently affect the e-business world both at B2C and B2B levels. For example consider the simple scenario of a B2C interaction in which a client wants to order a pizza for delivery. In such a scenario the user has some specifications (e.g. pizza ingredients, specific geographical location to deliver pizza, pizza rates etc.). To perform such a task a client has to manually discover and execute required services one-by-one, which is not an effective approach. Similarly, B2B interactions in a distributive business environment involve prior agreements and pre-defined standards between interacting partners. Such prior agreements at different levels of integration cannot motivate efforts for dynamic interaction between Web services.

Current Web services standards (i.e. WSDL [19], SOAP [17] and UDDI [18]) provide syntax-based interaction and composition of Web services in a loosely coupled way. However, dynamic composition needs more than syntactical information about Web services functionality. SWSs are capable of providing such kind of information, which make Web services capabilities understandable for computers. Several current efforts (e.g. OWL-S [13] and WSMO [20] and WSDL-S [15]) aim at providing Web service semantics.

After all these efforts to add semantics to Web services technology, dynamic and automated composition is still an open question. Different solutions, like, enhancing BPEL4WS (shortly BPEL) [1] to create such dynamic composition or using AI planning to automate the composition process of required services have been proposed. Till now interaction between Web services either in the form of the BPEL process model or as a composite service generated by an AI planner, is not dynamic or does not consider both functional and non-functional semantics of a service in the composition process.

Several approaches have been proposed to address Web services composition problem. Most of them fall into one of the following two categories: methods based on pre-defined workflow model and methods based on AI planning [11]. The first method uses

workflow techniques. The second approach is based on AI planning techniques. Both of these methods have their own composition approaches. The workflow method is more meaningful and useful in situations where problem model (e.g. BPEL process model) is already defined. In such a method dynamic composition involves discovery and binding of required services within Web services composition. On the other hand, AI planning method is more suitable in situations where requester has no process model but has a set of constraints and preferences. On the basis of this set of constraints and preference final composition can be generated automatically by the program [5].

In section 3 of this paper we describe a motivational scenario for our work and highlight some composition issues (challenges) that can arise in case of syntax based Web services composition. Keeping in mind these issues, we first analyse major existing approaches for dynamic and automated composition of Web services both from workflow and AI communities and untangle their limitations (with respect to composition issues highlighted in our motivational scenario). Then we point out major challenges for dynamic and automated composition of Web services and propose our framework encompassing essentially required modules to bridge the gap within this domain.

The remaining paper is organized as follows: Section 2 provides a review of related technologies. Section 3 discusses the motivational scenario. Different composition approaches have been briefly discussed and compared in section 4. In section 5 we exploit and integrate these approaches and propose a new framework for dynamic and automated composition of Web services. Section 6 concludes our work and discusses future directions.

## 2. Background

This section gives a review of different technologies that are being used by academic and industrial researchers for dynamic and automated Web services composition (e.g. BPEL [1], OWL-S [13]).

### 2.1 BPEL4WS

Business Process Execution Language for Web Services (BPEL4WS) is a well-known process modelling and execution language that can be used to model business processes as composition of Web services. BPEL composes Web services by defining a workflow and binding required services at design time. Discovering required services manually and composing them syntactically is not enough for dynamic and automated composition. A BPEL process model can itself be exposed and used with other services to perform some complex jobs that a single service alone cannot do. For example, different process modelling tools (e.g. MS BizTalk Server, IBM WebSphere, Vitria's BusinessWare and BEA WebLogic) support such kind of tasks. Exporting a BPEL process as a Web service has same syntactical limitations as traditional syntax based WSDL services. However, we have presented an approach and relevant tool [9] that can be used to map syntax based composition to SWSs composition (OWL-S composite service). The mapped OWL-S service exposes semantic information to facilitate dynamic discovery, composition and invocation.

BPEL uses *primitive* and *structured activities* to define a process as a composition of Web services. *Primitive activities* (e.g. *Receive*, *Send* and *Invoke)* can be used to communicate with the outer world by sending and receiving appropriate messages. The sample code given below shows a very simple example of BPEL *primitive activity* (*Invoke*), which performs a Web service operation "*DeliverPizza*" by sending and receiving appropriate messages from a Web service. The BPEL's *Structured activities* (e.g. s*equence*, *flow*, *while*, *switch* etc.) can be used to define control flow between process components. For example, *structured activity* (s*equence*) defines that child activities will be performed in a sequence.

```
<invoke partnerLink="DP_Ser_Port"portType="DPSerPort-
  Type" operation="DeliverPizza" inputVariable="Mess_
    _To_DP_Ser" outputVariable "Mess_From_DP_Ser"/>
```

### 2.2 OWL-S

OWL-S is suite of OWL ontologies. It provides machine understandable description of a Web service (annotated with domain ontologies). Such semantic based descriptions of Web services facilitate dynamic discovery, invocation and composition tasks. OWL-S suite consists of *Profile*, *Process Model* and *Grounding* ontologies.

*Profile* ontology describes capabilities of a Web service. Semantics about service capabilities can be categorized as functional and non-functional semantics. Functional semantics include information about input, output, pre-condition and post-condition of a service. Sample code given below provides a very simplified example of *Profile* ontology (i.e. DeliverPizzaProfile) for an *atomic process* "DeliverPizzaProcess". The sample code shows that input and output parameters of an *atomic process* have data type of ontological concepts "Pizza" and "Delivery" which are defined in appropriate domain ontologies.
Non-functional semantics provide information about service provider, geographical location, QoS semantics etc. Section 4 describes different composition approaches, which use this semantic information to dynamically compose Web services. Specially, we will see how non-functional semantics can be used to filter and select a single service when multiple services have been discovered for composition on the basis of matching functional semantics.

```
<profile:Profile rdf:ID="DeliverPizzaProfile">
  <service:isPresentedBy
    rdf:resource="#DelilverPizzaService"/>
```

```
  <process:hasInput rdf:resource="#PizzaIngredients"/>
  <process:hasOutput
    rdf:resource="#PizzaDeliveryNotification"/>
</profile:Profile>

<process:Input rdf:ID="PizzaIngredients">
  <process:parameterType rdf:datatype="&xsd;#anyURI">
    &bibtex;#Pizza</process:parameterType>
    <rdfs:label>Pizza Ingredients</rdfs:label>
</process:Input>

<process:Output rdf:ID="DeliveryNotification">
    <process:parameterType rdf:datatype="&xsd;#anyURI">
      &concepts;#Delivery</process:parameterType>
    <rdfs:label>Pizza Name</rdfs:label>
</process:Output>
```

*Process Model* ontology composes Web services in flow like a workflow language. The OWL-S defines a subclass (*Process*) of *ServiceModel*, which draws upon well-established work in a variety of fields, including work in AI on standardizations of planning languages [13]. *Process Model* ontology has three kinds of processes i.e. *atomic processes*, *composite processes* and *simple processes*. An *atomic process* is a process that can be performed in a single step. A *composite process* may have one or more sub *atomic* or *composite processes* and can define control flow between processes by using OWL-S control constructs (e.g. *Sequence, Split, Split-Join etc.*). Third kind of processes (i.e. *simple processes*) can be used to define a level of abstraction between process components.

The *Grounding* describes how to interact with an OWL-S service. The *Grounding* of a process establishes its correspondence to a particular WSDL operation, and the correspondence of each I/O element to a particular WSDL message part element. Also, XSL Transformation script can be defined in *Grounding* ontology to transform input/output of complex data types to WSDL supported syntax form.

## 3. Motivational Scenario

To understand the motivational task behind efforts for dynamic and automated Web services composition, let us consider a simple scenario in a Pizza delivery use case. Figure 1 shows a pizza delivery process as syntax based composition of different Web services (e.g. *Check Location* service, *Check CreditCard* service etc.). If any of the services involved in such syntax based composition of Web services failed or is not accessible over network, the process will not be able to perform the pizza delivery task. Such syntax based and manual composition of Web services will result in following challenges for pizza delivery process to complete:

- Syntactically composed service (e.g. *Sell Pizza* service) can fail to deliver pizza to every pizza request with required ingredients.
- It may be possible that *Deliver Pizza* service will not be geographically suitable for every pizza request.
- *Check CreditCard* Web services can possibly change its behaviour with the passage of time.

With such potentially unfriendly behaviour of Web services, syntax based composition cannot be considered as flexible, reliable and efficient approach in rapidly growing dynamic and automated e-business world. With these limitations of syntax based Web services composition, the only possibility to complete the pizza delivery process is either to change process flow with available services that provide same results or to manually find and compose other services that perform same task. But relying on syntax based composition and changing the process flow manually is not a flexible and an efficient approach.

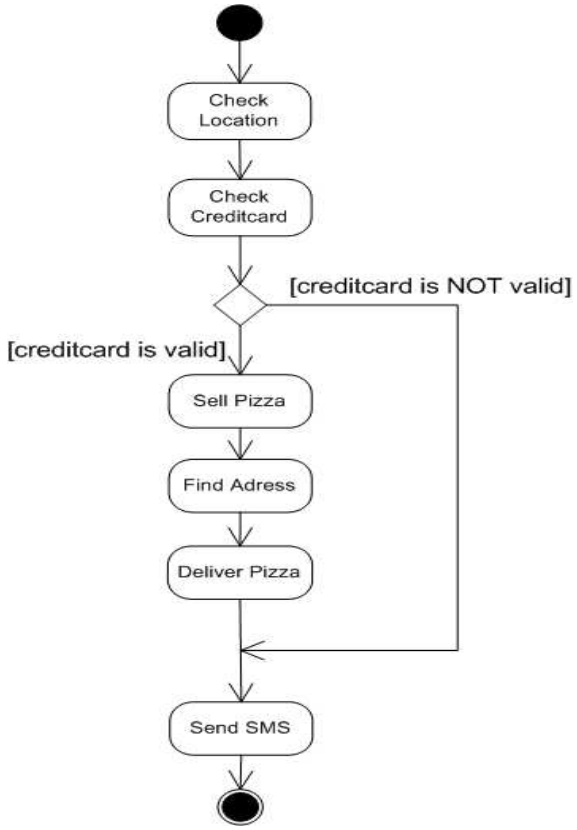


Fig.1. Pizza Delivery Process.

Dynamic and automated composition of Web services can perform the pizza delivery process in a more flexible and efficient way. For example, if one service in composition is not able to deliver pizza with required pizza specifications, dynamic composition approach can discover, invoke and compose another Web service on the basis of matching functional and non-functional semantics (a Web service which can deliver pizza with required pizza ingredients). Similarly, a user requesting to deliver a pizza in New

York, needs a pizza service delivering pizzas in New York, not in Chicago. Such a task can be performed by dynamically composing a required service that meets pizza specifications and is also geographically suitable for a pizza request. If a Web service (e.g. *Check CreditCard* Web service) within Web services composition changes on the fly, dynamic and automated composition can handle this problem by finding, invoking and composing another service on the basis of matching semantics to perform same task. In next section we describe some existing approaches to perform Web services composition task in dynamic and automated fashion.

## 4. Current Approaches for Dynamic and Automated Web Services Composition

Dynamic and automated composition by means of Web services semantics is most important and promising task for SWSs community. Different approaches both by workflow and AI communities have been presented for this purpose. In this section we have a look on some of most promising composition approaches and then we discuss how these approaches are limited to perform dynamic composition.

### 4.1 Web Services Composition and Execution Framework

The framework discussed in [10] provides mechanism and tools for visual orchestration of semantically well-defined building blocks and semantic invocation of services that match to the user specifications. The dynamic composition approach presented in this work uses pre-defined flow of complex service extended with abstract functional building blocks. These abstract building blocks define requirements for a service to perform a specific task. The best matching service is discovered and invoked at execution time. A part of this work has been discussed in [12] which describes how to handle BPEL limitations of static Web service binding with late binding by using the idea of "generic Web service proxies". This work presents idea of service ontology based semi-automatically generated *activity components*, which can be used and manipulated by tools (e.g. for visual modelling of complex services, in deployment phase, in execution phase during their invocation by workflow engine by using set of interfaces exposed by *activity components etc.)*. Framework proposed in [10] does not fully support dynamic composition on the basis of both functional and non-functional service semantics, which reduces efficiency of proposed framework.

### 4.2 Dynamic Composition by Using WSDL-S

WSDL-S [15] is another effort to provide Web service semantics. WSDL-S development team has presented a tool for dynamic composition of Web services. The dynamic composition tool uses abstract processes for defining and discovering required services dynamically. Among multiple numbers of discovered services one service is selected for composition. Process designer uses BPEL for modelling processes. Partner services in a BPEL process are specified by using service templates. Service template allows a process designer to specify semantic description of required Web service or binding to a known Web service. A required service described semantically results in automatic semantic base discovery of Web service. Once a service is selected from bundle of discovered Web services, required information is extracted from WSDL file and added in BPEL process and relevant WSDL file. Also at this stage user has to define required data flow between two activities.

Following the template based approach, the project team has presented their relevant work in [16 & 6] for semi-automatically integrating partner services either at design time or deployment time. The METEOR-S work supports design time or deployment time binding of Web services because location of all WSDL files is required in a BPEL process before deploying process. Once a WSDL file is selected and bound in the process, the final BPEL file is displayed in the designer so that user can complete workflow. The resulting file is executed with BPWS4J. But such a design time or deployment time binding of services is not enough for real dynamic Web services composition.

Another important point of this tool is the use of WSDL-S (a METEOR-S project approach to describe Web service semantics) rather than using OWL-S ontology. The METEOR-S's work presented in [7 & 14] discusses the project approach for adding semantics to Web services standards. The proposed work extends WSDL tags to add semantic information and add new tags (e.g. pre-condition and post-condition tags). These semantically enhanced Web services can be published to semantically enriched registries [8].

In above-discussed work of METEOR-S project, process designer has facility to define semantic templates for required services. On the basis of these templates, matching services can be discovered from service repository dynamically and most suitable service can be added to the process from this bundle of discovered services. However, such a manual selection of a required service in composition process keeps the project work away from ground realities of dynamic Web services composition. Also this work has not considered the issue that if a single service does not meet requirements of a Web service directly, then METEOR-S framework should create a service chain by discovering and combining multiple services so that they closely match to required service. The resulting service chain should take as input the input of the required service and should return required output. METEOR-S claims to improve dynamic composition of Web services from design time or de-

ployment time to run-time dynamic composition in upcoming versions of their tool.

### 4.3 Semi-automatic Composition Using OWL-S

The work discussed in [4] presents a prototype semantic Web services composition tool. The tool discovers semantically matching services from available services repository, filter these services and present them at each step of Web services composition.

The service composition tool consists of two components (i.e. inference engine and a composer) and discovers and filters these discovered services on the basis of their matching functional and non-functional semantics. The inference engine stores information about all available services in its knowledge base and is capable of finding matching services. The composer is the user interface that handles communication between human operator and the inference engine. The inference engine discovers matching services on the basis of matching semantics and filters most suitable services on the basis of functional and non-functional attributes. But the tool interface does not allow the user to define the control flow between atomic processes during the composition process. Also users are not able to define condition statements to have some conditional results of a composite process.

*Filtering on Functional Attributes:* At each step of composition, the composer presents those services for composition whose output can be taken as input for a selected service. On the basis of the *Profile* description, services are matched as exact match or generic match. Exact match is the result of belonging of a service parameter of two services to the same class. Such matches have priority in composition and are displayed on top in the list of matching services. A service is marked as generic match, if its output is sub-class of input parameter of current service. These services have less priority in composition and are presented at bottom in the list of matching Web services.

*Filtering on Non-Functional Attributes:* It is difficult to choose a service from a list of available services, if the number of services becomes high. It also becomes difficult to select a service from list only on the basis of the service name or small description about a service. At this stage non-functional attributes (e.g. geographical location, response time etc.) are used to filter and select a suitable service.

After selection and composition of services, final composition is generated as a DAML-S (OWL-S) *composite process* with its *Profile* so that it can be advertised and composed with other services. The composition framework can execute the resulting composite service by invoking individual services in Web services composition and passing data between services according to the defined data flow.

Such a composition involves human interaction at each step of composition that not fully automates the process of Web services composition. Composition process becomes more and more complex as number of matching services increases. One of the major drawbacks of this approach is that composition of services doesn't allow defining control flow for execution order of different services within composition.

## 5. A Framework for Dynamic and Automated Web Services Composition

In this section we describe a general framework at an abstract level for dynamic and automated Web services composition. On the basis of above discussed challenges and limitations of recent approaches we propose a composition framework, which consists of four modules (fig. 2). Each of these modules is responsible to perform a specific task that, in combination with other modules results in a SWSs composition framework. We describe each of these modules in detail and discuss which specific composition problem is addressed by each module.

**Semantic Service Requester:** The first step to perform the dynamic Web services composition is to discover and select required services on the basis of matching semantics. This dynamic discovery and selection is a run time process. Because semantic base discovery and selection of required services at design time also involves human interaction, which no more automates the process of Web services composition. The discovery and selection process of required services is based both on matching functional and non-functional semantics of a Web service. For example, in case of a pizza delivery process, a user sitting in New York requests a vegetable and mutton pizza. In case of such a request, there would be multiple services that offer vegetable and mutton pizza delivery. But in this case, a service with non-functional matching semantic (e.g. suitable geographical location for a pizza request) is selected for composition. At this stage it is assumed that suitable work has already been done to publish SWSs on semantically enriched registries that have capabilities to reply for SWSs queries. In our proposed framework, module 1 (Semantic Service Requester) is responsible to perform such a semantic base service request and to select a service for composition, which has closer semantic match to service request.

**Service Binder:** This module is responsible to bind a dynamically discovered and selected service within composition. Runtime binding of required services can help to meet challenges produced by services which change on the fly or which become inaccessible. For example, in case of composing services into a workflow each partner service is bound in workflow at run time so that only those services become part of the composition which are currently accessible and meet the functional and non-functional requirements. Similarly, in case of the AI planning approach a single service performing some action in a

single step (*atomic process*) becomes part of the final composition (complex service) generated by a composition plan. Module 2 of the proposed framework is responsible for run-time binding and referencing of a service within Web service composition.

**Composition Generator:** This module (module 3) is responsible for generating the final composition of semantic services, discovered and bound within composition at run time. In case of a workflow language as a composition of these dynamically discovered and bound services, this module is responsible for generating the final composition process in some workflow language (e.g. BPEL). In case of an AI planning for automatic Web services composition this module generates the final composition as a complex service (composite service). Composition Generator composes these services with well-defined control flow and data flow within composition. Different approaches have been discussed [2, 3] to automatically compose SWSs defined by using OWL-S descriptions. The automatic composition of OWL-S services can result in an OWL-S composite service. Since, WSDL-S does not support to define composite services, no approach has been discussed which allows to automatically compose Web services using the WSDL-S.

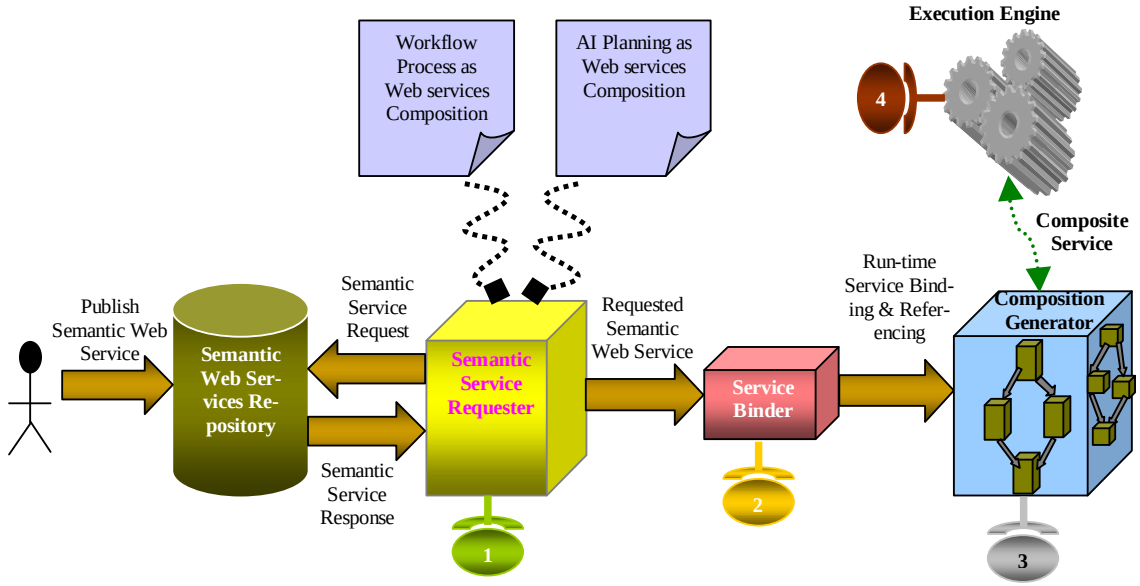


Fig.2. Architecture of proposed framework for dynamic Web services composition.

**Execution Engine:** Finally, the composition of dynamically and automatically composed services is executed at this stage (module 4). Each service involved with in the composition is executed according to the defined control flow. The data flow definition helps to pass data between services with in composition. For example, the approach discussed in section 4.3 uses its execution engine to execute the resulting OWL-S *composite process*.

## 6. Conclusions and Future Work

Web services and SWSs are being adopted rapidly in grid computing, distributed environments and P2P systems. A successful use of SWSs in distributed and grid environments is only possible if its related issues (i.e. dynamic discovery, invocation and composition) are resolved efficiently. Dynamic and automated Web services composition will maximize the process of Web services interaction. Complex business tasks like arranging a trip to a conference or requesting a pizza delivery, which involve the interaction and execution of multiple services, can be performed in more flexible and reliable way by dynamically composing required services. In this paper we provided a review of research work and highlighted major challenges and issues that need to be addressed to meet ground realities of dynamic Web services composition. On the basis of identified problems and limitations of the existing work, we have presented a novel framework that provides a solution at an abstract level for incentive composition of Web services.

## Acknowledgments

This work is partially supported by the **Higher Education Commission (HEC) of Pakistan** under the scheme *"Partial Support Scholarship for PhD Studies Abroad"*.

## References


[1] Business Process Execution Language for Web Services Version 1.1. 5th May 2003. [online] Available ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

[2] D. Sell, F. Hakimpour, J. Domingue, E. Motta and R. C. S. Pacheco: Interactive Composition of WSMO-based Semantic Web Services in IRS-III. Proceedings of the First AKT Workshop on Semantic Web Services (AKT-SWS04) KMi, The Open University, Milton Keynes, UK, December 8, 2004.

[3] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau: HTN Planning for Web Service Composition using SHOP. Journal of Web Semantics, 1(4): 377-396,2004.

[4] E.Sirin, J. Hendler and B. Parsia: Semi-automatic Composition of Web Services Using Semantic Descriptions.Proceedings of Web Services: Modeling, Architecture and Infrastructure Workshop (WSMAI), Angers, France, April 2003, pp. 17-24.

[5] J. Rao and X. Su: A Survey of Automated Web Service Composition Methods. First International Workshop, SWSWPC 2004 San Diego, C, USA, July 2004, Revised Selected Papers.

[6] K. Sivashanmugam, J. Miller, A. Sheth and K. Verma: Framework for Semantic Web Process Composition. Special Issue of the International Journal of Electronic Commerce (IJEC), Eds: Christoph Bussler, Dieter Fensel, Norman Sadeh, Feb 2004.

[7] K. Sivashanmugam, K. Verma, A. Sheth and J. Miller: Adding Semantics to Web Services Standards. Proceedings of 1st International Conference on Web Services (ICWS'03), Las Vegas, Nevada (June 2003) pages 395-401.

[8] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar and J. Miller: METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. Journal of Information Technology and Management 2004.

[9] M. A. Aslam, S. Auer, J. Shen, M. Herrmann: Expressing Business Process Model as OWL-S Ontologies. In proceedings of the 2nd International Workshop on Grid and Peer-to-Peer based Workflows (GPWW 2006) in conjunction with the 4th International Conference on Business Process Management (BPM 2006), Vienna, Austria, LNCS 4103 , Sept. 4, 2006, pp.400-415.

[10] M. Flügge and D. Tourtchaninova: Ontology-derived Activity Components for Composing Travel Web Services. Presented at the International Workshop on Semantic Web Technologies in Electronic Business (SWEB2004), Berlin, Germany, October 2004.

[11] M. Matskin, P. Küngas, J. Rao, J. Sampson and S.A. Petersen: Enabling Web Services Composition With Software Agents. Proceedings of the Ninth IASTED International Conference on Internet and Multimedia Systems and Applications, IMSA 2005, Honolulu, Hawaii, USA, August 15-17, 2005, ACTA Press, pp. 93-98, 2005

[12] M. Paolucci, T. Kawarmura, T. R. Payne and K. Sycara: Importing the Semantic Web in UDDI. In Proceedings of Web Services, E-Business and Semantic Web Workshop, CAiSE 2002, Trontoi, Canada.

[13] OWL-S: Semantic Markup for Web Services. [online] Available http://www.daml.org/services/owl-s/1.2/overview/

[14] P. Rajasekaran, J. Miller, K. Verma and A. Sheth: Enhancing Web Services Description and Discovery to Facilitate Composition. International Workshop on Semantic Web Services and Web Process Composition, 2004 (Proceedings of SWSWPC 2004).

[15] R. Akkiraju, J. Farell, J.A. Miller, M. Nagarajan, A. Sheth and K. Verma : Web Service Semantics – WSDL-S [online] Available http://www.w3.org/2005/04/FSWS/Submissions/17/WSDL-S.htm.

[16] R. Mulye, J. Miller, K. Verma, K. Gomadam and A. Sheth: A Semantic Template Based Designer for Web Processes. Proceedings of the IEEE International Conference on Web Services (ICWS'05), pages 461-469.

[17] SOAP Version 1.2 Part 1: Messaging Framework [online] Available http://www.w3.org/TR/soap12-part1/.

[18] UDDI Spec Technical Committee Draft, Dated 20041019 [online] Available http://uddi.org/pubs/uddi_v3.htm.

[19] Web Services Description Language (WSDL) 1.1. [online] Available http://www.w3.org/TR/wsdl.

[20] Web Services Modeling Ontology [online] Available http://www.wsmo.org/