A User-Centric WS-Mediator Framework for on-the-fly Web Service Composition

Tuo Zhang and Ken Chen

Abstract — **Nowadays, the effective and adaptive dynamic Web service composition is a major challenge for a real success of Web services among ordinary users. For the latter, and from the viewpoint of a user centric paradigm, existing work has limitation on their agility to create a composed service on the fly according to the desire/need of an end-user at a given time/place. This article presents our approach which consists in providing a comprehensive framework for a user centric WS-mediator which is capable of dynamic service composition. It is based on a composition engine which follows user's specific needs and which yields a composed service through a WS knowledge base. Users can mash up the services at run time with their own logic and have a fully dynamic composition with context adaptation through a WS-mediator that is also capable of supporting the semantic web.**

Keywords — **Context adaptation, mash up, mediator, semantic Web, user centric, Web service composition.**

I. Introduction

THE trend of NGN (Next Generation Network) has pushed the cyber world towards a revolution. The deployment of such NGN raises the issue of development of the so-called Next Generation Service (NGS). Among the main characteristics of and challenging issues raised by the NGS, we can mention heterogeneity, mobility and user centric paradigms [1]. Our work in this paper focuses on one of them, namely the user centric paradigm. As the end-users want to be able to experience the advanced sophisticated interactions, to manage the lifecycle of the services, or to define the service strategy on their own logic through the Internet, the user-centric requirement becomes increasingly important through the technological development. In order to create the Web applications that are flexible and adapted to the NGN context, with an immediate deployment, Web Services paradigm has been widely applied in the services composition, in order to create the Web applications that are flexible enough to be adapted to the NGN context, with an immediate deployment, Service discovery and selection, service control, semantic-based composition, and service execution entity deployment are key components for providing flexible and effective user-centric services. Our approach, which aims at enabling user-centric in a dynamic and adaptive context, works as a mediator. Thus, it can accommodate easily semantic web methods.

Tuo Zhang is with L2TI, Institute Galilee, University of Paris13, France (tuo.zhang@univ-paris13.fr).

Ken Chen is with L2TI, Institute Galilee, University of Paris13, France (ken.chen@univ-paris13.fr).

The web service composition is a key element for the development of SOA (Service-Oriented Architecture), which means to create a composition model (the "Business Process" or BP) based on the WS's syntax (inputs and outputs), constraints, as well as contextual information. The web service composition can be classified into static and dynamic composition. The static composition allows the service providers to make service composition at design time ("offline"), usually under the form of a profile to be used, later, upon actual service composition requests. On the other hand, the dynamic service composition is becoming the center of intense research efforts as well as industrial-oriented experiences. Since WSDL/SOAP WSs and RESTful WSs adopt differing styles (imperative against declarative) and view the services from different perspectives (operation-centric against resource-centric), the composition problem of these two kinds of WSs is different and attracts many researches [2]. In this paper, our motivation is to provide a user-centric WS-mediation framework, so that an end-user can compose, on-the-fly (that we understand as a kind of dynamicity), the desired service by composition of available web services provided through the mediator. The latter can be barely considered as a "market" (or "Store"), which facilitates the setup of relation between goods providers (here WS provider) and consumer (here the end-user).

Furthermore, data mediation between heterogeneous WSs has been addressed in our framework. Specifically, both the syntactical and semantic heterogeneity may exist in the input and output messages exchanged between WSs. In other words, the output of the previous WSs may not exactly match the required input of the successive WSs. Data mediation can provide a formalized model and mechanism for managing data heterogeneity that may exist in the WSs component. This challenging problem is beginning to capture increasing research attention in the semantic Web service community [3].Actually, with Web Ontology Language (OWL) such as OWL-S [4], rules can be established for a dynamic service composition according to abstract level user need description. This is an open question due to numerous obstacles. One proposed approach [3] models the involved domains using ontologies and relies on the pre-constructed data mappings to solve the heterogeneity issue. One of the main problems is the dynamic composition of services, that is to say, the legerity and agility of a BP to adapt to the context. Our framework takes this into account and is capable of supporting semantic web extension.

We adopt the "mash up" approach as the starting point of our mediator framework. Our primary goal consists in providing a stand-alone process that can run on a user's device. Actually, today's devices are powerful enough to offer the advantage, and freedom, of getting a stand-alone execution entity on the fly, just like one download Android or Apple application, which will be based on available WSs. With this approach, we don't need a server for "orchestration" in a classic type of Business Process Execution Language (BPEL). In our framework, there is a composition engine that generates an executable autonomous entity.

In a dual manner, if the user prefers to keep the execution elsewhere, the execution entity can be generated to be hosted by a kind of hypervisor (within a kind of cloud for instance) assigned by the user. In this way, the hypervisor acts as a kind of services broker and the user can use his/her composed service from any device he/she may have.

Basically speaking, the mediator keeps a knowledge base of WSs. Each of them is given a descriptor, a kind of meta-model that contains various information, including the input/output, the semantics (for semantic web extension) as well as the locations of available entities providing the service (which allows context adaptation).

A natural extension is the creation of meta-WS, which can be integrated into the above knowledge base exactly in the same manner as a real basic WS. These meta-WS are a kind of "proxy". Thus, our WS-mediator also allows the creation of true "intermediaries" which we refer to as a semantic extension capacity. In particular, a WS-mediator can be used to choose, at run time for instance, the best WS according to the using context/reference. Exploring the knowledge on existing WS for the purpose of semantic web can also create it.

In this article, we present a prototype that we are developing for validating the above system and its applications. This work is conducted partially within the French ANR/VERSO/UBIS project. This article is organized as follows: Section II presents the related work about service composition. Section III proposes our new approach for dynamic service composition. Section IV shows our prototype implementation based on the .NET 4.0 frameworks. Section V provides a sample application. Finally, we conclude this paper and discuss the future work in Section VI.

II. Related Work

There are different views and focus among researchers working on the Web Service composition (WSC). There exist several methods for the service composition, of which two different execution models are usually applied syntactically: Orchestration and Choreography [5]. There are already lots of works around it. Reference [7] proposes an on-the-fly approach to web-based service composition as well as a component model for separating the service business and user interface so that they can be changed dynamically and independently in the adaptation of service selection and composition. But it does not distinguish the design-time and runtime when the Web services are composed. Chafle et al. [8] introduce adaptation on different levels, the instance level, logic level and physical level. Multiple back up workflows are prepared to substitute the failed components or to adapt to environmental changes but they lack considering user requirement. Two approaches — top-down and bottom-up are proposed in [12], top-down composition can create new services with management using a main composer, while the bottom-up creates a new service under human-oriented direction. Reference [13] proposes an end-to-end service composition for information transport based on principles from SOA such as dynamically composing a transport service with characteristics matching the requirement of a given application. Besides the popular BPEL4WS and WSFL (Web Service Flow Language), the other interaction composition models are: Petri Nets, Labeled Behavior Diagrams, State Chart Diagram, Mathematical model and UML Activity. Finite automata, Markov process model, Temporal Logic of Actions, Web Service Flow Graph and Hierarchical Task Network [14]. The IEEE Next-Generation Service Overlay Networks (NGSON) working group is focusing on the integration, architecture was proposed in [9] according to the NGSON concept with its extension for services composition. Reference [10] proposes some criteria to identify the levels of dynamism and automation in service compositions. Moreover, they propose a strategy where different techniques can be used to make compositions more automatic and dynamic with a model driven approach, but the problem is that although the method can generate the process model automatically, there is a lack of interactions of designers, which means it cannot accept the designers decision during the composition process as auxiliary information to generate a next flow path at runtime because only the syntactic binding exists. Li and Kumara proposed a forward and backward (bidirectional) search based approach [15] for WSC. They compared their algorithm with other approaches in two simplistic WS composition benchmarks as part of the WS Challenges, exhibiting good results in terms of the speed of composition. Only the input and output as the functional description of WS are considered. But this may not be realistic because multiple WSs with the same input and output often exist and the approach provides no way to choose between them. So it is necessary to consider not only the syntax issues of service composition but also the semantic heterogeneity within it. Compared to data, a service can present a broader form of heterogeneity. Correspondingly, the WS research community has identified a broader form of semantics-data (I/O), functional (behavioral), nonfunctional (QoS, policy), and execution (runtime, infrastructure, exceptions) [16]. Several research works and projects have been conducted in semantics for traditional (WSDL/SOAP) WSs to help address heterogeneity and mediation challenges, such as in [3], they present an approach for resolving the service heterogeneities and focus on data mediation in a Web services based environment using pre-defined mappings

which is a part of the METEOR-S project. Radiant [17] which is an eclipse plug-in graphical tool that enables you to annotate existing Web service descriptions with Ontologies to create a SAWSDL files [18].Meanwhile, the WSMO project [20] which coined the term data mediation in the WSs context, it defines Preconditions and Effects and can be used for semantic annotation of WSDL with WSDL-S but there is no mechanism to describe Choreography or Orchestration in it, which seems an incomplete mix of semantic and syntactic. As to the WSMX that is the reference implementation of WSMO, the aim is to increase business process automation in a very flexible manner while providing a scalable integration solution [21]. A language for dynamic service composition that is called MetaBPEL is defined in [13]. It extends the WS-BPEL 2.0 language with semantic information, and acts as an abstract workflow definition mechanism but not avoid the complexity if the BPEL. Besides, there is also the "mash up" [7] type composition approach, which offers a better legerity. Such as APIhut [19] builds a nice ecosystem in which people can reuse Web APIs, but we must also develop advanced capabilities leading to dynamic configuration and composition for complex services.

While a variety of WS composition approaches and algorithms have been proposed, few solutions or implemented tools are available to support a loose-couple for both the syntax and semantics. This is because of the complex nature of the WS composition problem and the inherent scalability issues exist. So we propose a WS-mediator in a mash up way for on-the-fly WS composition in which we address the data mediation while composing the services syntactically. In addition, our architecture can also provide Semantic Web extension to make automatically WS identification and chaining according to the user's semantic logic. Coupling our mediator with WS search function, we can also provide context-aware functionalities.

III. Proposition: A User-Centric WS-Mediator Approach

We present hereafter the main functional entities of our approach.

The user begins by choosing the web services that he wants according to his logic from a WS Catalog (termed by us as "BDC"). This catalog is provided by a WS Knowledge base with all the information of each WS element, and in particular, a model for each WS. Furthermore, the functional semantics, domain ontology and parameter ontology based Web service description methods can be used for WS composition that allow the semantic web. The composition is done, in our current prototype, through a GUI (Graphic User Interface), script-based extension will be added in the future. In this way, we get a composed execution entity that is totally autonomous. As the composition is based on the model of the WS, The WS-Mediator system monitors Web Services at different locations in the Internet and dynamically assesses their dependability (Fig. 1).

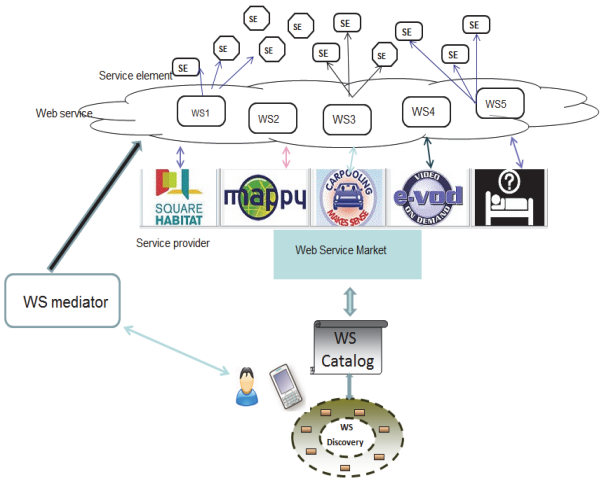

Fig.1. A user-centric WS-mediator approach.

A. *Knowledge Base (BDC)*

This is our WS Catalog, which is a web service Knowledge database whose rule consists in gathering all the information (such as URI, operation, input/output from WSDL) about Web services. The "BDC" is made up manually (offline) in the current stage of our work from the WSDL description of each service. This "BDC" is used to display the available WS Catalog and to know the exact model of chosen WS.

B. *Composition Entity*

- Selection module: as we said before, the user interface is graphical in the current step. The end-user can choose the available web services from the "Catalog".
- Link module: it serves to establish a link between a pair of services (*WSi,WSj*) which include that the user does the matching between the output of *WSi* and input of *WSj*, in case of incompatibility types, it displays a warning message.
- Configuration module: the user can control whether each service is perfectly configured. If all the inputs are configurable, it re-launches the connection module according to a need.
- Display module: The user can view at any moment in graphic form or textual form of web service composition done.

Considering each WS is autonomous, when we want to run the service composition, the end-user has no control over the services, even though the services themselves are not reliable, they can be changed, removed or updated by the service providers. Consequently, the possibility of faults is high when the service composition is executed. We handle this problem at run time during the composite process. If we want to compose two web services, we will begin with choosing the services that we want to use, testing whether the services are available, and check the inputs and outputs of them. Once the services are ready to be used, the binding will be implied in the invocation of the composed services. And then we have the service selection and the process model generation done.

In particular, there will be an interaction with "BDC" to know the inputs of each chosen WS. We insure that every input of each invoked WS be set. Each input is in one of 3 cases: 1) "constant": in this case, the composer invites the user to fix the value; 2) "run-time": in this case, the composer will insert in the final autonomous execution entity a specific code allowing the user to enter the value at runtime; 3) "from another WS": in this case, a link is established between this input and an output of a chosen WS. Of course, the composer has to check the compatibility of the links established by the user (for instance, a "character chain" type output cannot be mapped to an "integer" type input).

C. Execution Entity

The execution entity receives the composition pattern provided by the composition entity. The end-user is involved in this phase by introducing the values of input type caught from the various WSs and the execution entity is totally autonomous.

D. Capacity for the Semantic Web Extension of the WS-Mediator

Web service description is the foundation of SOA methodology. For our framework, we choose SOAP and WSDL as the traditional technologies for Web services. The basic elements are defined in WSDL 2.0 including <description>, <types>, <interface>, <binding> and <service> as well as the operations which represent the simple interaction in practice. And the WSDL document may declare multiple interfaces, each of which describes a service presenting multiple operations that are called for by the real Web services. So based on WSDL, Semantic Annotations for WSDL and XML Schema (SAWSDL) [22] help us to add semantics to a pure WSDL syntax description. This allows the annotation of service discovery, composition, selection, negotiation, mediation and invocation [23] by using appropriate tools that can be integrated into our framework. Therefore we can define a semantic extension for the WS-mediator that can combine the information and services from multiple sources, and it has the capability of context adaptation, exchanges and mashes up the services from the existing ones with OWL-S or SAWSDL.

Alternatively, the web services can also be divided and then reformed by providing the restricted descriptions of the Input, Output, Preconditions and effects of the web services. It also provides the mapping between elements of SAWSDL. To illustrate various mapping representation options, we can use SPARQL for representing mappings through the ontology knowledge base that will be integrated in our BDC. The reasoning ability of ontology can help to resolve the substitution operation for the mismatching problem of web services so that we have a context adaptation with the semantic extension. The semantic extension will be integrated in accordance with certain rules for effective bonding to form a new, user centric web service composition. If there is no single service that could meet the user requirements, we can proceed by deductions and the dynamical combinations of semantic, based on the self-descriptions and the marks on the OWL-S of either the functional or the non-functional requirement among the known web services. Therefore, our approach enables us a loose-coupled mash up way for the WS-mediator both on syntax and semantic. The implementation of the semantic extension (which will be based on existing tools) is out of the scope of this paper.

IV. Implementation

Although many approaches have been proposed in the literature, few implemented tools exist. Based on SOA, Web services, and WSC technologies, we have implemented our approaches and provided a comprehensive tool. The tool suite accepts WSs described using a standard language such as WSDL as well as SAWSDL that can provide us a semantic extension. It provides users or process designers with an intuitive interface to specify the requirements, goals and a hierarchal composition, and generates an execution entity, while hiding the complexity of the planning and BPEL from users.

We choose to do our implementation using web service developed in WCF (Windows Communication Foundation) of the Framework .NET 4.0. because WCF supports not only SOAP message, but also it can be configured to support standard XML data that is not wrapped in SOAP, or can even be used to support other formats. This yields opportunities for evolutions such as the integration of the RESTful service.

We build the WCF based on three elements:

- Address: the address that the user must connect to use the service.
- Binding: the protocol to be used by the user to communicate with the service.
- Contract: the information exchanged between the mediator and user so that he knows how to use the service.

The tool suite is an integrated development environment for the process designers to:

- Import candidate WSs and their description files,
- Specify process hierarchies, initial state and goals,
- Generate the plan and convert the plan into the corresponding execution entity.

In order to realize the three points mentioned above, we define and create the service contract and its parser module to provide the output of the previous WS to exactly match the required input of the successive WS. Service contract is defined:

- To be exchanged between the mediator and user,
- To allow the user to know what are the methods proposed by the service and how to use them.

The development of the Service Contract is performed through the 3 following metadata:

- <ServiceContract>: This metadata is used to define a class. It serves to indicate the class or an interface is a Service Contract.
- <OperationContract>: This metadata is attached to the methods that we want to expose through WCF service. Thus, it is technically possible to expose certain methods of a class to the user.

- <DataMember>: This attribute is placed before the properties of classes that define objects that are going to exchange the parameters to the service.

For example, we want to compose two accessible WSs when we just know their WSDL. One is GeoIp by which we can get the location of the country via the IP address, the other is Weather by which we can get the weather via a given list of the cities.

The service contract outlines the information that describes the service delivery. It defines a mechanism for the service orchestration between the service elements. It defines in particular the interface specification as well as describes the service logic and its purpose as to implement the process information about the service elements to supply a more efficient treatment. It focuses on the organization and parsing of input/output data treatment as well as the QoS (if needed). Fig. 2 provides the details of the service contract with schema XSD (XML Schema Document).

```
<?xml version="1.0" encoding="utf-8" ?>
<Contrat xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="Structure_Contrat.xsd">
  <Services>
    <Service>
      <ServiceNom>geoip</ServiceNom>
      <Inputs>
        <Input>
          <InputNom>IP</InputNom>
          <InputType>string</InputType>
        </Input>
      </Inputs>
      <Outputs>
        <Output>
          <OutputNom>Pays</OutputNom>
          <OutputType>string</OutputType>
        </Output>
      </Outputs>
      <Indice>1</Indice>
      <Visibilite>true</Visibilite>
    </Service>
    <Service>
      <ServiceNom>GetVille</ServiceNom>
      <Inputs>
        <Input>
          <InputNom>Pays</InputNom>
          <InputType>string</InputType>
        </Input>
      </Inputs>
      <Outputs>
        <Output>
          <OutputNom>Ville</OutputNom>
          <OutputType>string</OutputType>
        </Output>
      </Outputs>
      <Indice>2</Indice>
      <Visibilite>true</Visibilite>
    </Service>
  </Services>
  <Relation>
    <ServiceConnecte>
      <Nom>GetVille</Nom>
      <IndiceIn>2</IndiceIn>
      <Nom_Input>Pays</Nom_Input>
      <Service_Entrant>geoip</Service_Entrant>
      <IndiceOut>1</IndiceOut>
      <Nom_Output>Pays</Nom_Output>
    </ServiceConnecte>
  </Relation>
</Contrat>
```

Fig. 2. Schema XSD of the service contract.

Then we parse this XSD file and generate the code (see Fig. 3) that allows creating execution entity for the end-user. Finally in the GUI, we will get the result in Fig. 4.

V. Application

In order to apply our system, we assume that the BDC contains WS1: “Mappy” Map service, WS2: “Square habitat” a real estate service, WS3: VOD, WS4: Car-pooling Taxi as well as WS5: a Real-Estate Agent localization service.

The user can choose the available services from the “Catalog”, after his launch interface for service dynamic composition (Fig. 5) and composes an automatized service chain, according to his/her own logic: for instance: find a real-estate property by showing automatically some photo/video, for a selected property, locate an agent and find a car pooling taxi to get there.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace PFE27012012
{
 public class Structure
{

private List<Service> _liste = new List<Service>();
public Structure()
{

/*-----------------------0--------------------*/
Service geoip1= new Service();
geoip1.nom="geoip";
geoip1.Indice=1;
geoip1.Visible=true;
AttributService i00= new AttributService();
i00.nom = "IP";
i00.type = "string";
geoip1.Inputs.Add(i00);
AttributService O00= new AttributService();
O00.nom = "Pays";
O00.type = "string";
geoip1.OutPuts.Add(O00);
geoip1.ServiceTete=true;
/*-----------------------1--------------------*/
Service GetVille2= new Service();
GetVille2.nom="GetVille";
GetVille2.Indice=2;
GetVille2.Visible=true;
AttributService i10= new AttributService();
i10.nom = "Pays";
i10.type = "string";
GetVille2.Inputs.Add(i10);
AttributService O10= new AttributService();
O10.nom = "Ville";
O10.type = "string";
GetVille2.OutPuts.Add(O10);
GetVille2.ServiceTete=false;
geoip1.Suivant = GetVille2;
GetVille2.getInput("Pays").connextion=geoip1.getOutput("Pays");
_liste.Add(geoip1);
_liste.Add(GetVille2);
}
public List<Service> liste
{
get { return _liste; }
}
}
}
```

Fig. 3. Code to generate the execution entity.

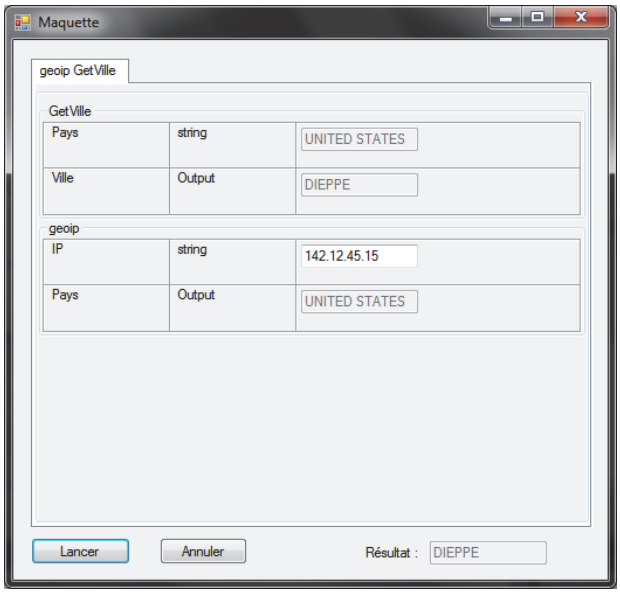

Fig. 4. Result in the GUI.

VI. Conclusion and Future Work

The dynamic web service composition is a challenging issue in the NGN/NGS context, especially for the user centric requirement. In this paper, we provide a user-centric WS-mediator that allows the end-user to mash up the service in his way on the fly. Furthermore, this framework is capable of enabling dynamic Web Service with context adaptation and semantic extension.

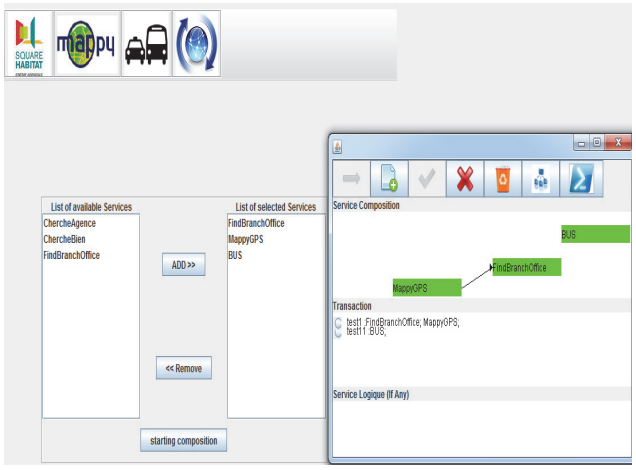

Fig. 5. Application.

We are going to implement the semantic extension in our tool with an appropriate web semantic language. Some of the nonfunctional aspects as to QoS parameters (such as Response time, reliability and invocation cost) are not considered in our framework. These QoS measurements might be crucial in certain application domains and heterogeneous networks. Meanwhile, the current development of the WS-Mediator system does not explicitly address security issues. Therefore, the WS-Mediator should be compatible with those applications that employ security models and mechanisms such as authentication and non-reputation. Furthermore, we should pay attention to the RESTful Web services because these services can integrate easily into various applications or services. This supposition needs, however, to be considered in our future work.

REFERENCES

[1] C. Yin, N. Simoni, and G. du Chéné, "A personalization and mobility aware service enabler for a service continuity in heterogeneous networks," *MSPE'06*, 2006.

[2] H. Zhao and P. Doshi, "Towards Ahtomated RESTful Web Service Composition," *ICWS*, 2009.

[3] M. Nagarajan, K. Verma, A. P. Sheth, and J. A. Miller, "Ontology driven data mediation in web services," *Internationnal Journal Web Service Research*, pp. 104-126, 2007.

[4] T. O. S. Coalition, "*Owl-s: Semantic markup for web services*," Available: http://www.daml.org/services(2004)

[5] C. Peltz, "Web services orchestration and choreography," *IEEE Computer*, pp. 46-52, 2003.

[6] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards Service Composition Based on Mashup services," *IEEE Congress on Services*, pp. 332-339, 2007.

[7] Q. Zhao, G. Huang, J. Huang, X. Liu, and H. Mei, "A Web-based Mashup Environment for the On-the-fly Service Composition," *IEEE International Symposium on Service-Oriented System Engineering*, pp. 32-37,2008.

[8] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava, "Adaptation in Web Service Composition and Execution," in *IEEE International Conference on Web Services*, *IEEE Computer Society*, pp. 549-557, Chicago, 2006.

[9] C. Macaya, B. Falchuk, D. Chee, F. J. Lin, S. Das, M. Ito, S. Komorita, T. Chiba, and H. Yokota, "Services Composition based on Next-Geeration Service Overlay Networks Architecture," in *New Technologies, Mobility and Security (NTMS)*, *4th IFIP International Conference*, *IEEE Computer Society*, pp. 1-6, Paris, 2011.

[10] S. P. Sivasubramanian, E. Ilavarasan, and G. Vadivelou, "Dynamic Web Service Composition: Challenges and Techniques," in *Intelligent Agent & Multi-Agent Systems (IAMA)*, *International Conference*, pp. 1-8, Philharmonic Luxembourg, 2009.

[11] M. Popovici, M. Muraru, A. Agache, L. Negreanu, C. Giumale, and C. Dobre, "An ontology-based dynamic service composition framework for in Telligent houses," in *Autonomous Decentralized Systems (ISADS)*, *10th International Symposium*, pp. 177-184, Kobe, Japan, 2011.

[12] H. Mizugai, I. Paik, and W. Chen, "Scalable Orchestration Strategy for Automatic Service Composition," in *Computer and Information Technology (CIT)*, *IEEE 10th International Conference*, pp. 1474-1479, Bradford, UK, 2010.

[13] C. Fortuna and M. Mohorcic, "Dynamic Composition of Services for End-to-End Information Transport," in *Wireless Communications*, *IEEE Communications Society*, pp. 56-62, 2009.

[14] Challenge (Reasearch Issues) in Web Services, *International Conference on Advances in Computer Engineering*, 2010.

[15] S. C. Oh, H. Kil, D. Lee, and S. R. T. Kumara, "Algorithms for web services based on syntactic and semantic service descriptions," *The third IEEE International Coference on Enterprise Computing*, *E-Commerce and E-Services*, 2006.

[16] K. Sivashanmugam et al., "Adding Semantics to Web Services Standards," *Proc. Int'l Conf. Web Services (ICWS)*, *CSREA Press*, pp. 395-401, 2003.

[17] Radiant SAWSDL/WSDL-S Annotation Tool, Available: http://lsdis.cs.uga.edu/projects/meteors/downloads/index.php?page=1

[18] K. Gomadam, et al., "Radiant: A tool for semantic annotation of Web Services," in *The Proceedings of the 4th International Semantic Web Conference (ISWC)*, 2005.

[19] K. Gomadam et al., "A faceted Classification-Based Approach to Search and Rank Web APIs," to appear in *Proc. IEEE Int'l Conf. Web Services*, *IEEE Press*, 2008.

[20] http://www.wsmo.org/TR/

[21] http://www.wsmx.org/

[22] www.w3.org/TR/sawsdl

[23] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell, "SAWSDL: Semantic Annotations for WSDL and XML Schema" *IEEE Internet Computing*," 11(6), 60-67. doi: 10.1109/MIC.2007.134., 2007.