On-demand Data Integration On the Cloud

Mahmoud Barhamgi[1], Parisa Ghodous[2], Djamal Benslimane[3]
Claude Bernard University (Lyon1)
69622 Villeurbanne, France
[1]Mahmoud.barhamgi@liris.cnrs.fr
[2]Parisa.Ghodous@liris.cnrs.fr
[3]Djamal.benslimane@liris.cnrs.fr

Abstract— **On-demand data integration is among the key challenges in Cloud Computing. In this paper, we present an ontology-based framework for describing and integrating data on the fly to answer transient business needs. We provide a semantic modeling for cloud's data services. The proposed modeling makes it possible to automatically resolve the different types of data heterogeneity that would arise when data from heterogeneous and autonomous providers need to be combined together to answer the business's data needs. We validate our approach with a prototype. The main contribution of this paper is an efficient on-demand integration system for the clouds.**

Keywords— On-demand data integration; Ontologies; Services.

I. INTRODUCTION

Cloud computing has recently emerged as a new paradigm for hosting and delivering services over the Internet. Cloud computing is attractive to business owners as it eliminates the requirement for users to plan ahead for provisioning, and allows enterprises to start from the small and increase resources only when there is a rise in service demand. However, despite the significant benefits offered by cloud computing, the current technologies are not mature enough to realize its full potential. Many key challenges in this domain need to be addressed and solved. Data management and integration is among the key challenges that will keep receiving a particular attention from the research community over the coming years [6] [8] [14]. The Data-as-a-Service concept has been introduced in recent year as first step to virtualize access to data sources in clouds and SOA architectures [2][3][5][12]. A DaaS (Data-as-a-Service) service provides a simplified, integrated view of real-time, high-quality information about a specific business entity, such as a *Customer* or *Product*. The information that it provides may come from a diverse set of information resources, including operational systems, operational data stores, data warehouses, content repositories, collaboration stores, and even streaming sources in advanced cases.

Even though the introduction of DaaS services has allowed to shield the applications developers from having to directly interact with the various data sources that give access to business objects (i.e., *customers*, *orders*, *invoices*, etc.) and enabled them to focus on the business logic only, most of the time the business needs require the combination of multiple DaaS services from different service providers [13]. For instance, let us consider the following query: "*what are the driving directions for a facility of a given type (e.g., Restaurant, Theater, etc.) in a given city?*" -this is a typical application of Google maps *maps.google.com*. Let us assume that we have the following two DaaS services: S_1 returns the addresses of facilities of a given type in a given city; S_2 returns the driving directions between two given addresses. The execution of the above mentioned query involves the composition of S_1 and S_2 services. However, DaaS services composition is a hard task that may involve many data integration challenges. First, the semantics of DaaS services needs to be formally defined to automate their selection. The standardized service description languages (e.g., WSDL [17]) do not provide means for defining the services' semantics. Second, services may define different data structures for their manipulated data entities. For instance, the same piece of data such as "Address" may be represented differently by different DaaS services; i.e., the same data item has different XML structures. Structural data heterogeneities need to be addressed to allow for the automatic composition of DaaS services.

In this paper, we present an approach to compose cloud's DaaS services on the fly for the purpose of answering on-demand data integration needs. In the proposed approach, the semantics of DaaS services are defined using domain ontologies. This allows for automating their selection and composition and makes it possible to resolve the schematic data heterogeneities (a.k.a. structural data heterogeneities) of data items exchanged among heterogeneous DaaS services. We present also a system that exploits the proposed semantic modeling to compose DaaS services.

 ISBN: 978-1-61208-153-3

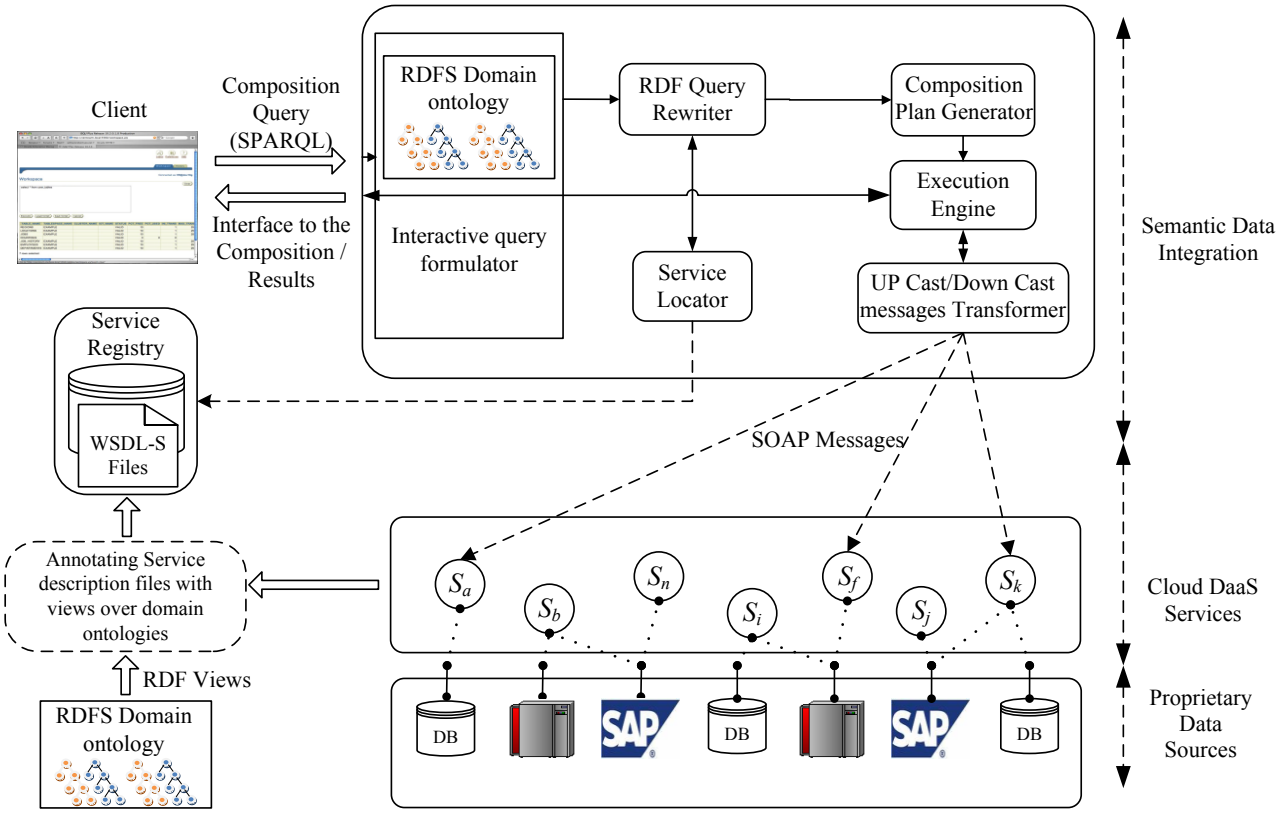


Figure 1: An overview of the proposed declarative approach to cloud services composition

The rest of this paper is organized as follows. In Section 2, we describe our framework for on demand data integration. In Section 3, we present our modeling to cloud DaaS services and users' queries. In Section 4, we showcase through an example how data integration queries are resolved by query rewriting and DaaS service composition. In Section 5, we overview related work. We provide concluding remarks in Section 6.

II. A Declarative Approach to Compose Cloud DaaS Services

In this section, we present a declarative framework for composing cloud DaaS services that addresses the challenges discussed earlier in the introduction. We show the different phases involved in DaaS services composition, starting from the service modeling to the generation of the final composition that will be returned to users.

Figure 1 presents our DaaS service composition framework. The first step towards the automation of DaaS services composition is to semantically represent their capabilities. In our approach, we model DaaS services as *RDF views* over domain ontologies. An RDF view uses concepts and relations whose meanings are formally defined in domain ontologies to define the semantics of a DaaS service. The RDF views are then used to annotate the service description files (e.g., WSDL files, SA-Rest, etc.).

Users (i.e., cloud application developers) in our approach formulate their composition queries over domain ontology using the do facto ontology query language SPARQL [18]. Non-savvy users can be assisted in formulating their queries by the `Interactive Query Formulator` component. Based on our proposed modeling to DaaS services (i.e., RDF views), the well-known query rewriting techniques can be used to compose them; i.e., our composition system rewrites the received queries in terms of available DaaS services using a query rewriting algorithm. For that purpose, we have devised an efficient RDF-oriented query rewriting algorithm [1]. The algorithm is implemented by the `RDF Query Rewriter` component and exploits the semantic annotations that we added in the service description files to select and compose the DaaS services that are relevant to the query. The composition system will then arrange the selected services in the composition execution plan (this is carried out by the `Composition Plan Generator` component). The composition plan will be displayed to the users, who can then invoke the compositions with their inputs. Note that when service providers define the semantics of their DaaS services using the RDF views over domain ontologies, they also provide the mappings between the defined views and the XML schemas of input and output messages of their

 ISBN: 978-1-61208-153-3

services. The mappings are also attached to the service description files as annotations and are used by the *Up-Cast/Down-Cast Messages Transformer* component when invoking component services. This is necessary since the same data item may have different structures between the ontology and the XML schemas of Input and Output messages (for instance, a (datatype) property in the ontology like "NAME" may be represented by two elements "FirstName" and "LastName" in an Input or Output XML schema). We detail all of the previous steps in the subsequent subsections.

III. A Semantic Description for DaaS Services and Composition Queries

In our approach, we model DaaS services as *RDF views* over domain ontologies. An RDF view describes the semantics of a DaaS service in a *declarative way* using concepts and relations whose meanings are formally defined in domain ontologies. Consider, for example, the services: $S_1(\$t,\$c,?n,?s,?b)$ and $S_2(\$c_1,\$s_1,\$b_1,\$c_2,\$s_2,\$b_2,?r)$ that we will use throughout the paper. Inputs are prefixed with "$" and outputs with "?". S_1 returns the facilities of a given type "*t*" (e.g., hospitals, hotels, etc) in a given city "*c*". The service S_2 returns the driving directions "*r*" between two addresses represented by the cities ("c_1" and "c_2"), the streets ("s_1" and "s_2") and the buildings ("b_1" and "b_2"). These two services can be composed together to look for facilities of a given type and obtain the driving directions to them. Figure 2 (Part-A) shows a graphical representation of the RDF views defined for S_1 and S_2. The RDF views in Figure 2 describe the semantics of services from the ontology point of view, where the blue ovals are concepts in ontology (e.g., *Facility*, *Address* and *Route*) whereas the arcs are properties. The defined RDF views are then used to annotate the service description files (e.g., WSDL files, SA-Rest, etc). These views define the semantics of services in a formal way and will be used during the selection and composition of DaaS services.

In the proposed approach, users (i.e., application developers) need only to focus on the needed data by formulating their composition queries over domain ontologies. They are not required to manually select services and build the composition plan by mapping the inputs and outputs of component services to each other and drop code to resolve data incompatibilities. Figure 2 (Part-B) shows a graphical representation of the query in the running example.

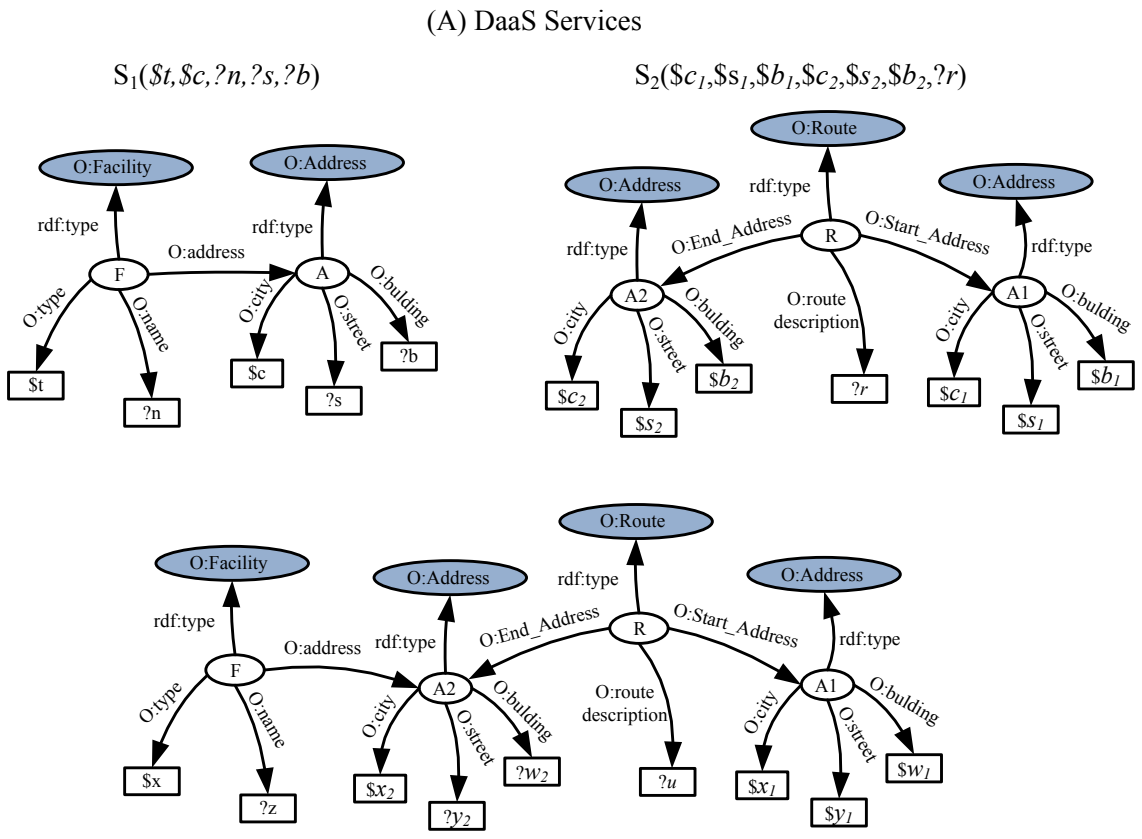


Figure 2: (A) the RDF views of services in the running example; (B) the user mashup query formulated on domain ontologies.

We will see in subsequent sections that they are still able to select the services participating in the resulting composition.

IV. COMPOSING DAAS SERVICES BY QUERY REWRITING

Our proposed composition approach relies on an RDF query rewriting algorithm (presented in [1]) to resolve the users' composition queries. Specifically, users' queries are matched against the RDF views of available services. These RDF views can be retrieved from the services description files (e.g., WSDL files). In the matching process, our matching algorithms identify the RDF sub-graphs of the query that can be covered by individual DaaS services. For example, as we can see in Table 1, the service S_1 covers the following nodes of the query: *F($x,?z)*, *A2($x$_2$,?y$_2$,?w$_2$)* and the object property linking the two *address(F,A2)*. The service S_2 covers the following nodes of the query: A2(x_2$,?y$_2$,?w$_2$),R(?d), A1($x$_1$,?y$_1$,?w$_1$) and the object properties : *end_address(R, A2)*, *start_address(R,A1)*.

Service	Covered sub-graphs
S_1($x,?z,$x$_2$,?y$_2$,?w$_2$)	F($x,?z), A2($x$_2$,?y$_2$,?w$_2$), address(F,A2)
S_2(x_1$,$y$_1$,w_1$,$x$_2$,y_2$,$w$_2$, ?u)	A2(x_2$,?y$_2$,?w$_2$), R(?d), A1($x$_1$,?y$_1$,?w$_1$),end_address(R, A2), start_address(R,A1)

Table 1: the query's sub-graphs that are covered by services in the running example

If these two services are combined together, the whole nodes and object properties sets of the query will be covered. Therefore, our composition algorithm will combine both of these services and consider the combination as a rewriting of the query as follows:

```
Q(?z,?y2,?w2,?u):- S1($x,?z,$x2,?y2,?w2)×
                   S2($x1,$y1,$w1,$x2,$y2,$w2,?u)
```

The composition algorithm will then orchestrate the used DaaS services in the rewriting to produce the composition execution plan that will be displayed to the user for further customization (if desired).

Figure 3 (A) shows the interface to the composition system. Users formulate their composition queries in the query panel using SPARQL language and submit the query to the system. The composition system will compose the DaaS services and present the user with composition plan in Figure 3 (B), where users can refine the composition by selecting the desired services among the possible ones and validate the composition. The composition system will then present the user with an interface where the users can specify specific values for the mashup parameters and invoke it. Figure 3 (A) shows the composition inputs values and the obtained outputs for the running example.

V. RELATED WORKS

Since the DaaS services composition research problem is relatively new, there has been only a small amount of research work addressing it. In the following, we review the most prominent ones of these works.

A considerable body of recent work addresses the problem of composition (or orchestration) of multiple web services to carry out a particular task, e.g., [15][16]. In general, that work is targeted more toward workflow-oriented applications (e.g., the processing steps involved in fulfilling a purchase order), rather than applications coordinating data obtained from multiple DaaS services, as addressed in this paper. Although these approaches have recognized the importance of automating the composition process, they have not, as far as we are aware, addressed the DaaS services.

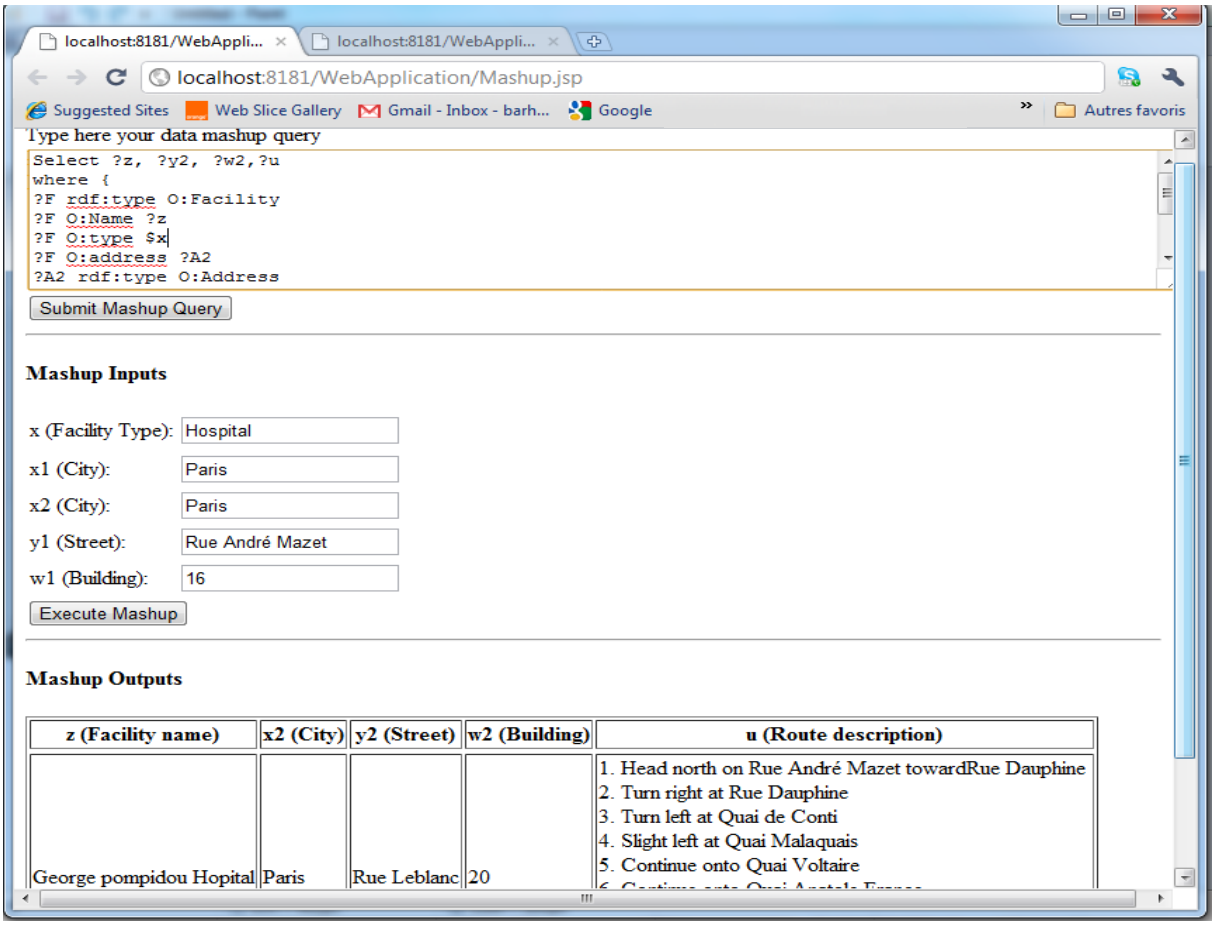
The Web Service Mediator System WSMED [9] allows users to mashup data services by defining relational views on top of them. Users can then query data by formulating their mashup queries over defined views. Users can also enhance defined views with primary-key constraints which can be exploited to optimize the mashups. The main drawback of the WSMED system is its high reliance on users; i.e. users are supposed to import the services relevant to their needs; define views on top of them and enhance the views with primary key constraints. The latter task requires from users to have a good understanding of the services' semantics. In our system, DaaS Web services are modeled as RDF views over domain ontologies where *primary key* constraints are defined explicitly by the concepts' skolem functions, thus the discussed *Primary key* based optimizations are included by default in our query processing model.

In other academic mashup systems [4][7][10][11], data mashup users are required to select the data services manually (which assumes they are able to understand their semantics), figure out the execution plan of selected services (i.e. the services *orchestration* in the mashup) and connect them to each other and drop code (in JavaScript) to mediate between incompatible inputs/outputs of involved services. This prevents average users from mashing up DaaS services at large. Our composition system addresses this limitation by proposing a declarative composition approach, where users need only to focus on the required data and the system will find and compose the services for them.

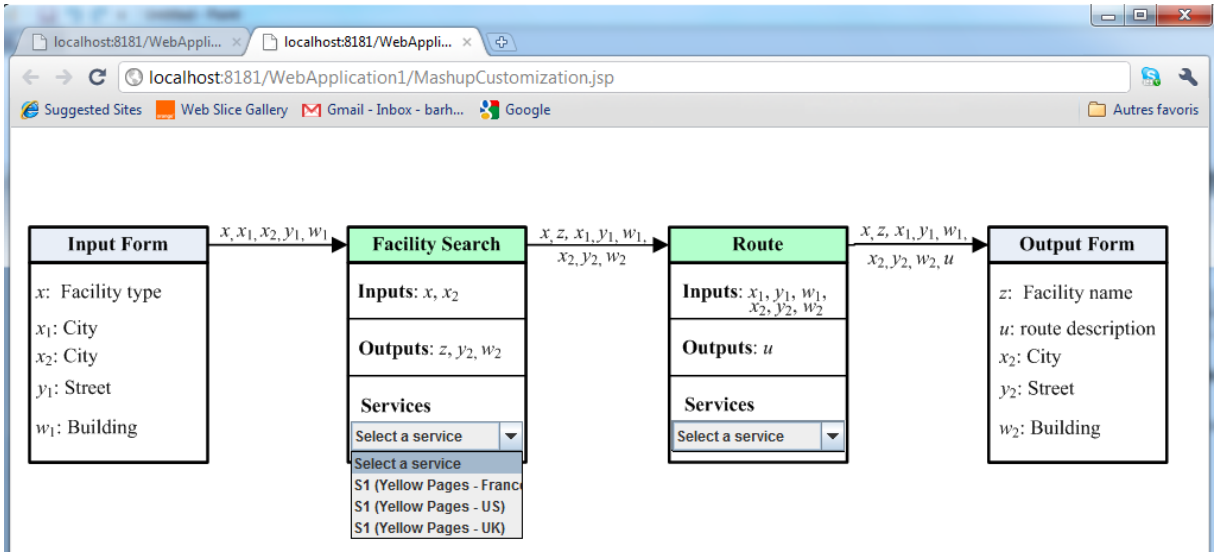
VI. CONCLUSION

In this paper, we presented an approach that caters for on-demand data integration for cloud business's data needs. We presented an ontology-based semantic modeling for cloud DaaS services. The proposed modeling makes it possible to automatically combine heterogeneous DaaS services and resolve the different types of data heterogeneity that would arise when data needs to be exchanged between composed services. We also validated our approach with a prototype. As a future work, we intend to contextual data heterogeneities between composed services (i.e., when composed services have different interpretation contexts for the data they exchange).

 ISBN: 978-1-61208-153-3



(Figure 3-A): The Mashup Interface: users type their mashup queries in the query panel, they will be presented then with the interface "*Mashup Inputs*" that is used to specify the values of input parameters to execute the mashup



(Figure-3-B): The Mashup Customization Interface MCI: the MCI allows users to select the desired services among the possible ones.

REFERENCES

[1] Mahmoud Barhamgi, Djamal Benslimane, and Brahim Medjahed, "A Query Rewriting Approach for Web Service Composition," *EEE Transactions on Services Computing (TSC)*, pp. 206-222, 2010. http://www.computer.org/portal/web/csdl/doi/10.1109/TSC.2010.4

[2] Michael J. Carey, "Data delivery in a service-oriented world: the BEA aquaLogic data services platform.," in *SIGMOD Conference*, 2006, pp. 695-705.

[3] Asit Dan, Robert Johnson, and Ali Arsanjani, "Information as a Service: Modeling and Realization," in *International Conference on Software Engineering (Workshop on Systems Development in SOA Environments)*, 2007, pp. 2-10.

[4] Hazem Elmeleegy, Anca Ivan, Rama Akkiraju, and Richard Goodwin, "Mashup Advisor: A Recommendation Tool for Mashup Development," in *2008 IEEE International Conference on Web Services (ICWS 2008)*, Beijing, China, pp. 337-344.

[5] Mike Gilpin et al., "Information-As-A-Service: Waht's Behind This Hot New Trend?," Forrester Research, Research Report 2007. http://www.forrester.com/rb/Research/information-as-a-service_whats_behind_this_hot_new_trend/q/id/41913/t/2, accessed on 29 June, 2011.

[6] Hector Gonzalez et al., "Google fusion tables: data management, integration and collaboration in the cloud," in *SoCC*, 2010, pp. 175-180.

[7] Anne H. H. Ngu, Michael Pierre Carlson, Quan Z. Sheng, and Hye-young Paik, "Semantic-Based Mashup of Composite Applications," *IEEE Transactions on Services Computing*, vol. 3, no. 1, pp. 2-15, 2010.

[8] Raghu Ramakrishnan, "Data Management in the Cloud," in *ICDE 2009*, pp. 5, 2009.

[9] Manivasakan Sabesan and Tore Risch, "Adaptive Parallelization of Queries over Dependent Web Service Calls," in *1st IEEE Workshop on Information & Software as Services, WISS 2009*, Shanghai, China, 2009.

[10] Junichi Tatemura, "UQBE: uncertain query by example for web service mashup," in *SIGMOD Conference*, Vancouver, Canada, 2008, pp. 1275-1280.

[11] Junichi Tatemura, "Mashup Feeds: : continuous queries over web services," in *SIGMOD Conference*, 2007, pp. 1128-1130.

[12] Hong-Linh Truong and Schahram Dustdar, "On Analyzing and Specifying Concerns for Data as a Service," in *The 2009 Asia-Pacific Services Computing Conference (IEEE APSCC 2009)*, Singapore, 2009, pp. 7-11.

[13] Qi Yu, Xumin Liu, Athman Bouguettaya, and Brahim Medjahed, "Deploying and managing Web services: issues, solutions, and directions," *VLDB Journal*, vol. 17, no. 3, pp. 537-572, 2008.

[14] Qi Zhang, Lu Cheng, and Raouf Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7-18, 2010.

[15] Mazen Shiaa, Jan Ove Fladmark, and Benoit Thiell, "An Incremental Graph-based Approach to Automatic Service Composition" Proc. of the Int. Conf. on Services Computing (SCC'08), Honolulu, pp. 212-220, 2008.

[16] Patrick Hennig and Wolf-tilo Balke, "Highly Scalable Web Service Composition Using Binary Tree-Based Parallelization," Proc. of the Int. Conf. on Web Services (ICWS'10), Los Alamitos, pp.123-130, USA, 2010.

[17] http://www.w3.org/TR/wsdl, accssed on 29 June, 2011

[18] http://www.w3.org/TR/rdf-sparql-query/, accssed on 29 June, 2011

 ISBN: 978-1-61208-153-3