Composing Data-Providing Web Services

Mahmoud Barhamgi
Supervised by
Djamal Benslimane
LIRIS Laboratory
Claude Bernard University Lyon 1
mahmoud.barhamgi@liris.cnrs.fr
djamal.benslimane@liris.cnrs.fr

Abstract

Today, more than ever, modern enterprises are using Web services for data sharing within and across the enterprise's boundaries. We call this kind of Web service as Data Providing Web services. In this paper, we present our approach to automatically compose primitive data providing Web services for the purpose of creating data integration applications. Our approach exploits existing mature works done in data integration systems. Specifically, data providing services are modeled as RDF Parameterized Views over mediated ontologies. Then, an RDF oriented query rewriting algorithm is used to compose services for answering received queries. The composition is then optimized and deployed as a new Web service accessible on top of the Web.

1. Introduction

Recent years have witnessed a growing interest in using Web services as a reliable medium for data publishing and sharing among organizations and individuals [1]. Modern enterprises are moving towards a service-oriented architecture for data sharing on the Web by putting their databases behind web services, thereby providing a well-documented, interoperable method of interacting with their data. Furthermore, data not stored in traditional databases also is being made available via Web services. We call this type of Web services as *Data-Providing Web services*, where services correspond to calls (i.e. parameterized queries) over the data sources' schemas.

As in traditional Web services, which we refer to as *Effect-Providing Web services* (**EP** Services), composing Data-Providing Web services (**DP** services in the hereafter) opens up the door for building interesting data-centric applications on top of the World Wide Web in many scientific and industrial domains. For example in bioinformatics, it allows biomedical scientists to conduct the so-called *in silico* experiments by assembling DP biomedical web services, it helps reconstituting the patient medical record in the healthcare application domain, etc.

In this paper we are interested in the problem of transparently answering user queries on the fly by composing DP services that are available on top of the Web. Composing web services is not a new problem; indeed there have been many approaches to compose traditional web services (EP services) (the reader is referred to the review [2]). These approaches generally select and compose web services based on their functional and/or non-functional properties. However, the determinative factor in selecting and composing DP services is the semantic relationship between their input and output parameters sets. Consequently, these approaches cannot be applied to DP service since they fail to take such semantic relationships into account. The automation of DP service composition requires the specification of such semantic relationships in a declarative and informative way, which can be only done by describing DP services as views over a mediated ontology.

In this paper, we lay out the foundations of an approach to query and automatically compose DP services like the approaches followed in traditional Database Management Systems (DBMS) with replacing (local) traditional data sources by DP services that are available on the Web. In the proposed approach, DP services are modeled as *RDF Views* over mediated ontologies to automate their selection and composition. Users are no longer concerned with finding and manually orchestrating the relevant services. They need only to specify their queries (formulated over domain ontologies) through a DBMS-like capability and then the system will find and orchestrate the DP services needed in answering the query in a transparent and integrated fashion.

The rest of the paper is organized as follows. In section2 we give a motivating example and highlight the challenges. In section 3 we describe our approach to compose DP services; this includes modeling and composing DP services and optimizing compositions. Finally, we conclude the paper with a concise summary and an overview of other achievements (in the context of DP service composition) we made throughout the thesis.

2. Motivating Scenario

Let us consider a portion of an RDFS mediated ontology in the healthcare domain as depicted in figure1 (part A). This ontology specifies that *patients* take *drugs*; *drugs* have different characteristics like *name*, a *universal code* and an *URL reference* to detailed information about it. Drugs may interact with each other and they have specializations like for instance the concept *Medications.* Domain concepts (e.g. *patient*, *drug*, etc) are drawn in figure1 as ovals, and datatypes as rectangles. Domain concepts are interlinked by *object properties* and linked to datatypes through *datatypes properties.*

Now consider the case of an *e-prescription* scenario in which a physician wants to verify whether the medication she wants to prescribe does not interact with the medications currently administered to the treated patient. The query the physician wants

VLDB '09, August 24-28, 2009, Lyon, France.

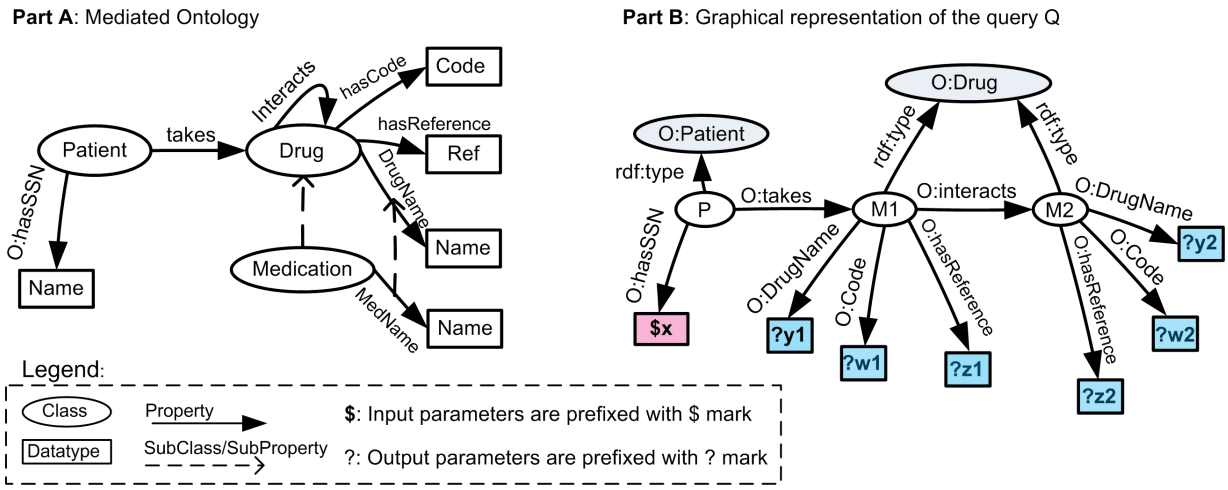

Figure 1: Part A shows a portion of a meditated ontology. Part B shows a graphical representation of the query Q

to answer is the following: ***Q***: ``*return the medications (their names* ***y****, codes* ***w*** *and a reference* ***z*** *to technical information) which interact with the medications administered recently to a given patient (denoted by her identifier* ***x****)".*

The physician did not bind her query to a specific patient (e.g. John Smith); rather she left the variable ***x*** unspecified. In other words, the query is parameterized over patient identifier ***x*** so that it can be exploited by the physician with different patients' identifiers. Given the mediated ontology, part B of figure1 shows the graphical representation of the query ***Q***, which is formulated concretely in SPARQL query language. Input parameters are prefixed with "$" mark and the outputs sought are prefixed with "?" mark.

Now assume that we have the **DP** services in table1 available to us, where the services S_1 and S_2 return the currently administered medications (denoted by their code *b*) of a given patient (denoted by an identifier *a*). They accept different ranges of values *a* as specified in the constraints column of table1. The service S_3 provides the interacting medications of a given medication. The services S_4 and S_5 provide the different information about a medication including its name, code, and a URL reference to more details on the Internet. They cover different value ranges of medications. Service S_6 provides equivalent medications of a given medication.

Table1. Descriptions of the DP services used in the example

Service	The semantics of the service	Constraints
S_1 ($a, ?b$)	Returns medications (denoted by their codes ***b***) taken by a given patient (denoted by an identifier ***a***)	a>=x888
S_2 ($a, ?b$)	Returns medications (the codes ***b***) taken by a given patient (denoted by her id ***a***)	a<=x888
S_3 ($a, ?b$)	Returns interacting medications (***b***) of a given medication (***a***)	
S_4 ($a, ?b, ?c$)	Returns various information of a given medication	a>=p660
S_5 ($a, ?b, ?c$)	Returns various information of a given medication	a<=x8999
S_6 ($a, ?b$)	Returns equivalent medications (***a***) of a given medication (***b***)	

Obviously the physician can use these DP services to obtain the detailed information of the interacting medications. Specifically, she can invoke S_1 or S_2 with the identifier of the patient, and retrieve his list of medications. Then she can invoke S_3 to retrieve the identifiers of interacting medications. Finally she can use these identifiers to invoke S_4 or $S_5$to obtain detailed information about the interacting medications.

The user (i.e. the physician in the example) in this task is confronted with the following challenges:

***Manual and long service selection and invocation processes*:** The user has to go manually through an overwhelming number of DP services to select the ones that may fulfill her task. Then, after services are selected, she has to invoke them manually one by one with the list of intermediate results and to compute the potential joins, if any, between the intermediate results of independent services and to filter out irrelevant row data.

***Inadequate data service description*:** DP Web services are simply parameterized queries exported on top of some data sources. The semantics of a DP service resides not only in its input and output types but also in how these input and output are interconnected in relation with underlying schema, which we call as the in/out semantic relationship. For example, both of the services $\boldsymbol{S_3}$ and $\boldsymbol{S_6}$ have the same input and output types (i.e. Medication), yet they have completely different semantics, i.e. one is returning the interacting medication of a given medication whereas the other is returning the equivalent ones. Such semantics can be represented in a *declarative way* via the views representing these services. Unfortunately, with the Web Service Description Language (WSDL[1]) it is impossible to specify such views that allow for in/out relationships to be exported in a compact and informative way. Moreover, most of Semantic Web service description languages (e.g. OWL-S and SAWSDL) allow only to specify the types of inputs and outputs, but not the relation linking the two; i.e. based on these languages the physician will not be able to distinguish between services like $\boldsymbol{S_3}$ and $\boldsymbol{S_6}$. As a result, much of interesting services will be missed. Even worse, the user may wrongly choose services that are irrelevant to her query.

[1] http://www.w3.org/TR/wsdl

The aforementioned challenges motivate the need for an approach to query and automatically compose DP services like the approaches followed in conventional DBMS. Similarly to a DBMS system, the user (i.e. the physician) in the targeted approach should be shielded from having to find and manually compose then invoke relevant services- and potentially apply joins and selections on intermediate raw data. All what she needs is just to specify her query (formulated over domain ontologies) through a DBMS-like capability and then the system, which we call as the Web Service Management System (WSMS), will find and orchestrate the DP services needed in answering the query in a transparent and integrated fashion. Second, the service description languages should be extended to allow for specifying the in/out relationship of a DP service in a declarative and informative way. This is essential for the automation of DP service selection and composition.

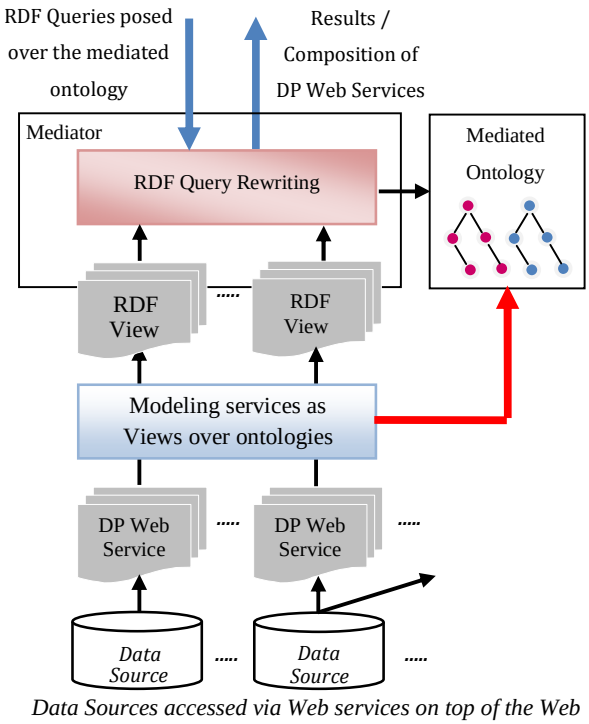

Figure2: An overview of the proposed approach for service composition

3. Query rewriting based approach to compose web services

3.1 An overview

We adopt a query rewriting based approach to compose data providing Web services [3,4]. Specifically, as Fig.2 shows, DP Web services are modeled as *RDF Views* over mediated ontology. RDF views capture in a faithful and declarative way the semantic relationships between input and output parameters using ontological concepts and relations whose semantics are well defined in the mediated ontology. RDF views are incorporated within services description files as annotations. Users pose their queries on a mediated ontology using SPARQL query language. Then, the WSMS exploits the defined RDF views within WSDL files to select the services that can be combined to answer the posed query using an RDF query rewriting algorithm. Then, it generates an execution plan for the composition and executes it to provide the user with requested data. Another possibility is to deploy the generated plan as a new Web service invocable by users and accessible on top of the Web. In this approach, users are no longer concerned with finding and manually orchestrating the relevant DP services. They need only to specify their queries (formulated over domain ontologies) through a DBMS-like capability and then the system will find and orchestrate the DP services needed in answering the query in a transparent and integrated fashion.

3.2 Modeling Data Providing Web Services as RDF Views

Semantic Web services are usually modeled with the de facto standard for service description OWL-S[2] **[5]**. In particular, OWL-S's Service Profile permits the modeling of the service's input, output, its mandated preconditions and the produced effects of invoking the service. It also allows the categorization of services according to their functionality in a domain using the Service Category class in conjunction with some categorization schemes available within an application domain. Categorization of EP Services plays a very important role in service matching and selection. DP services on the other hand are simply concerned with retrieving the appropriate output data given a specific input. They do not provide any functionality, beyond retrieval, and have no external effects. The concern is properly capturing the semantic relationship holding between their inputs and outputs. Therefore OWL-S is not suitable to describe DP services. The same applies for other languages like SAWSDL **[6]** and WSMO **[7]**.

In our approach **[3]**, we capture the semantic relationship between input and output sets using *RDF Parameterized Views (PVs)* over mediated ontologies. A parameterized RDF view uses concepts and relations whose meanings are formally defined in domain ontologies to define the semantic relationships between input and output sets of a data providing Web service.

Definition 3: *RDF Parameterized View*

An RDF Parameterized view over an ontology O is a predicate $S_i(\bar{X}_i, \bar{Y}_i) :- < \Phi(\bar{X}_i, \bar{Y}_i, \bar{Z}_i), Ct_i >$, *where:*

- $\bar{X}_i$ *is the set of variables denoting the input parameters necessary to invoke* S_i, *they are called as the* ***input variables****.*
- $\bar{Y}_i$ *is the set of variables denoting the returned literals from invoking* S_i, *they are called as the* ***output variables****.* Input and output variables are also called as the ***distinguished variables***.
- $\Phi(\bar{X}_i, \bar{Y}_i, \bar{Z}_i)$ *is the semantic relationship holding between input and output variables.* $\bar{Z}_i$ *is the existential variables set relating* $\bar{X}_i$ *and* $\bar{Y}_i$. Φ *has the form of RDF triples where each triple is of the form <<<subject.property.object>>>.*
- Ct_i *is the constraints set that is imposed on the variables* $\bar{X}_i$, $\bar{Y}_i$ *or* $\bar{Z}_i$. *A constraint has the form:* $x\ \theta\ CONSTANT$, *where* θ *is:* $>, \geq, <, \leq$.

[2] http://www.w3.org/Submission/OWL-S/

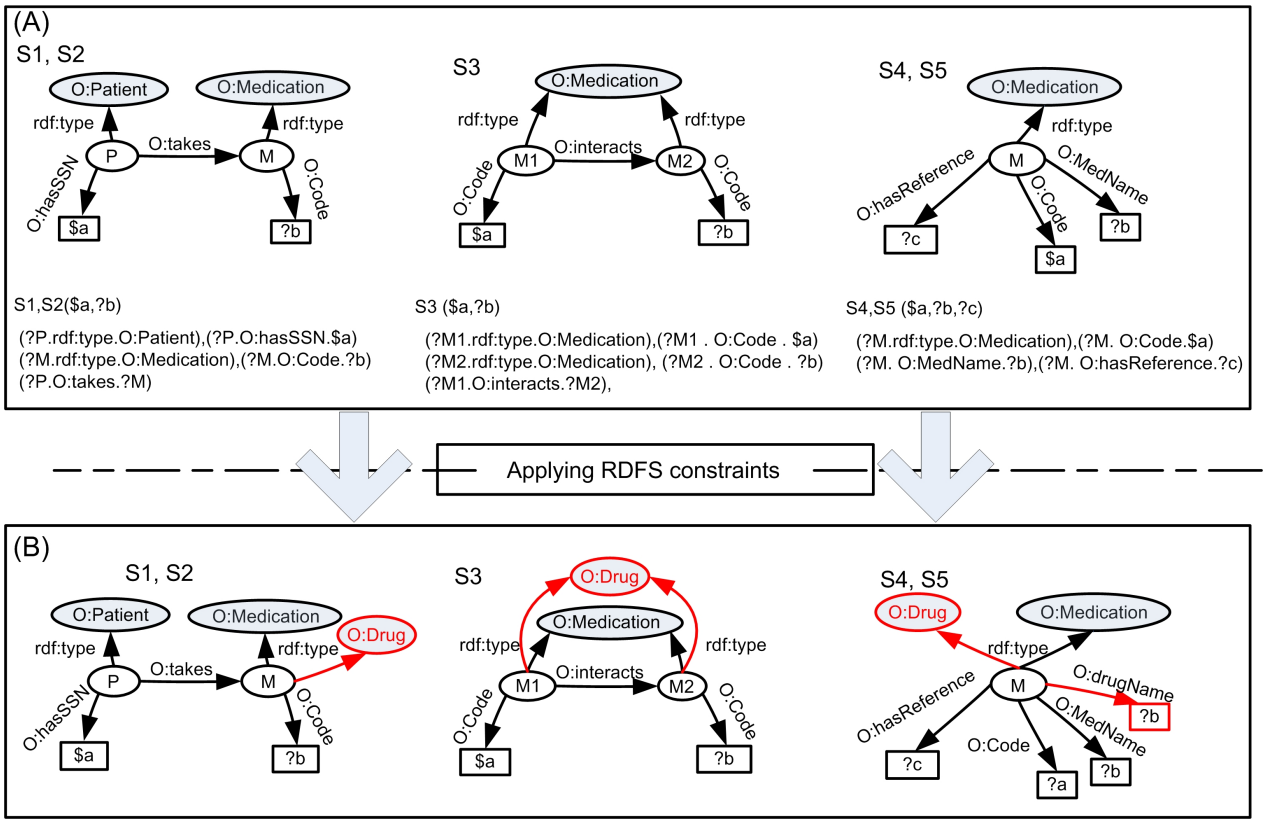

Figure 3: (A) Graphical representations of services; (B) The RDF views after applying the RDFS semantic constraints

Fig.3 (part A) shows the defined parameterized views for the services in the running example.

3.3 Extending the RDF Views with RDFS Constraints

In this step RDF views are extended to take into account the RDFS semantic constraints of the mediated ontology. RDFS semantic constraints include: *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:domain*, and *rdfs:range*. This extra inference process is carried out offline, and it is valuable because it enables the RDF query rewriting algorithm to return more results given the available services. For example, a query making reference to the concept ``Drug" would not be answered with a service if the latter makes reference to a narrower concept (e.g. the concept ``Medication", a specialization of ``Drug") although this service does indeed return relevant information for the query. Fig.3 (part B) shows that new triples were added to state that a variable of type Medication is also of the type Drug.

3.4 Composing DP Services by Query Rewriting

Given a Query Q and a set of services represented by their corresponding views $V = v_1, v_2, \dots, v_i$, a rewriting of Q using the services is constructed by a composition of services whose RDF graphs union covers the RDF graph of the query. More precisely the term ``covers" means: (a). all object properties that hold between *class-nodes* in Q are covered by the graph of the composition, (b). there is a mapping $\boldsymbol{\beta}$ between class nodes in Q with those in the graph of the composition, i.e. they have the same types (class), $\boldsymbol{\beta}$ maps also the literal nodes in Q to some literal nodes in the graph of the composition, and (c). Distinguished variables in Q (i.e. the requested literals) are provided by the composition. Individual services may cover only partitions of the query RDF graph; therefore answering the query will require investigating the different combinations of these partitions that would fulfill the above conditions. Valid combinations will be interchangeably called *rewritings* or *compositions*.

The construction of the compositions comprises two phases. These are:

Phase 1- Finding the RDF sub-graphs of the query covered by each service:

In this phase, the rewriting algorithm compares Q with every view v_i in V and determines the covered RDF sub-graph (of Q) in each v_i and documents this information as a partial containment mapping in the *Mapping table*. This table contains the different possibilities of using a view to cover a sub-graph of the query. Table2 presents the mapping table for the running example. The first row indicates that the service S_1 covers the node P_Q and the object property $\boldsymbol{takes}(P_Q, M_{1Q})$. This last property is joined in the query Q with the property $\boldsymbol{interacts}(M_{1Q}, M_{2Q})$ over the variable M_{1Q}, however S_1 provides the medication code on the output which can be used to enforce the join over M_{1Q} by using other services - the skolem function associated with the classes *Drug* and *Medication* states that these concepts can be uniquely identified by their code. Therefore, S_1 is considered as covering the RDF sub-graph $\boldsymbol{takes}(M_{1Q}, P_Q)P_Q(x)M_{1Q}(w_1)$. The same applies in the case of S_2, hence its insertion in the second row. The service S_3 covers

the object property $\boldsymbol{interacts}(M_{1Q}, M_{2Q})$since the join with other object properties is possible as S_3 takes the codes of medication as input and returns the code of a medication as output where medications codes can be used to enforce the joins with other properties. The services S_4 and S_5 covers the nodes M_{1Q} and M_{2Q} as can be seen in the last four rows.

Table2: the mapping table

Service	Variables Mapping	Covered nodes and Object properties
S_1 ($\$x, ?w_1$)	$P_Q \rightarrow P_{S1}$, $M_{1Q} \rightarrow M_{S1}$ $x \rightarrow a$, $w_1 \rightarrow b$	$takes(M_{1Q}, P_Q)P_Q(x)$ $M_{1Q}(w_1)$
S_2 ($\$x, ?w_1$)	$P_Q \rightarrow P_{S2}$, $M_{1Q} \rightarrow M_{S2}$ $x \rightarrow a, w_1 \rightarrow b$	$takes(M_{1Q}, P_Q)P_Q(x)$ $M_{1Q}(w_1)$
S_3 ($\$w_1, ?w_2$)	$M_{1Q} \rightarrow M_{1S3}, M_{2Q} \rightarrow M_{2S3}$ $w_1 \rightarrow a, w_2 \rightarrow b,$	$interacts(M_{1Q}, M_{2Q})M_{1Q}(w_1)$ $M_{2Q}(w_2)$
S_4 ($\$w_1, ?y_1, ?z_1$)	$M_{1Q} \rightarrow M_{S4}$ $w_1 \rightarrow a, y_1 \rightarrow b, z_1 \rightarrow c$	$M_{1Q}(w_1, y_1, z_1)$
S_5 ($\$w_1, ?y_1, ?z_1$)	$M_{1Q} \rightarrow M_{S5}$ $w_1 \rightarrow a, y_1 \rightarrow b, z_1 \rightarrow c$	$M_{1Q}(w_1, y_1, z_1)$
S_4 ($\$w_2, ?y_2, ?z_2$)	$M_{2Q} \rightarrow M_{S4}$ $w_2 \rightarrow a, y_2 \rightarrow b, z_2 \rightarrow c$	$M_{2Q}(w_2, y_2, z_2)$
S_5 ($\$w_2, ?y_2, ?z_2$)	$M_{2Q} \rightarrow M_{S5}$ $w_2 \rightarrow a, y_2 \rightarrow b, z_2 \rightarrow c$	$M_{2Q}(w_2, y_2, z_2)$

Phase 2- Forming the compositions:

In the second phase, rows from the mapping table are combined to cover the complete list of objects properties and class-nodes in the query. For instance, the first, third, fourth and the sixth rows can be combined together to cover the whole the query Q. This corresponds to the composition of S_1, S_3 and S_4. However, as some primitive services cover partially the required ranges of input/output parameters values, multiple similar primitive services are combined to cover completely the required ranges of values, for example, the services S_1 and S_2 were combined together to cover the complete range of patient's identifier. The same applies to the services S_4 and S_5 in covering the concepts M_{1Q} and M_{2Q}. The final composition is shown in figure 5. The composition is tested to verify whether it is executable or not. It is executable if the input parameters necessary for the invocation of its primitive services are available.

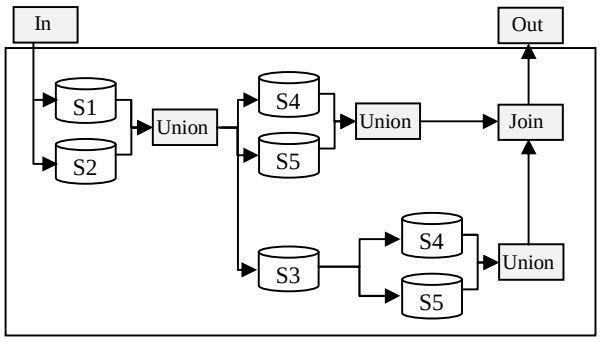

Figure5: The resulting composite Web service

3.5 Optimizing the Composition:

The execution of the composition obtained in the previous steps is inefficient. The reason is that some Web services are called with values of input parameters violating their specified constraints on accepted input values. Indeed each web service call has some fixed overhead, typically parsing SOAP/XML headers and going through the network stack. Therefore, eliminating superfluous calls (i.e. calls with values violating the service's input constraints) will have a significant impact on the execution time of the whole composition.

We exploit the constraints placed on the accepted values of input parameters to filter out superfluous calls to the composed primitive services. For example, filters were inserted before calling S_1 and S_2 to verify whether the used patient identifier is greater than "x888" in the case of S_1, and lower than "x888" in the case of S_2. In addition, filters were placed on S_4 and S_5 as can be seen in figure6.

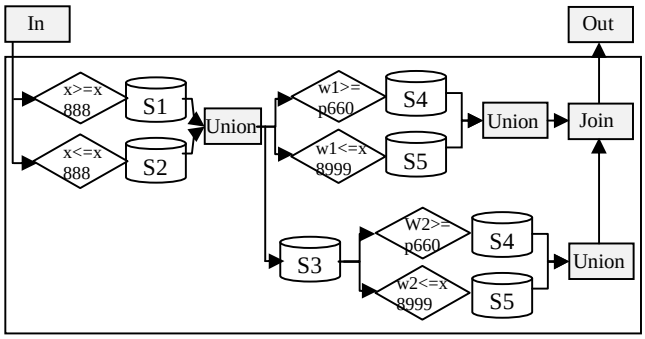

Figure6: the optimized composite Web service

3.6 Generating a plan for the composite service

We translate the composition to an execution plan that can be executed by a dataflow streaming engine like for example the system THESEUS [8]. The streaming system THESEUS proposes a certain number of operations (like for example *SELECT*, *RETRIEVE*, *JOIN*, *UNION*) that can be executed in parallel if they are independent. Data flow can be streamed between these operations. Filters are translated to *Select* operations that can be used to test whether the values used in invoking services satisfy the specified value constraints. *Retrieve* operations are used to carry out the invocations of primitive web services and the retrieval of the results. *Union* operations are used to unify the results returned from the invocations of primitive similar services.

4. Related works

There has been some recent research work addressing the problems of querying, and optimizing queries over DP services [9,10]. The project in [9] addresses query optimization over DP services. Queries are directly expressed in terms of services' predicates; i.e. users are implicitly assumed to have an understanding of the semantics of each DP service that is available to them. In contrast, in our work users are not required to have such understanding. They express their queries over domain ontologies and then queries are rewritten in terms of services based on their RDFS views. Also, in addition to specific queries, we are able to answer parameterized queries. Further, our data model is richer, i.e. our queries are formulated on mediated ontologies rather than on services' relational predicates and not

restricted to “pipeline queries”. Furthermore, as optimization we employ the constraints defined on each service to filter superfluous calls, but we plan to include data chunk optimizations in our future work. The CLIDE System [10] guides the user in her efforts towards formulating queries over DP Web services. However, contrary to our system, CLIDE system does not handle parameterized queries; optimization and deployment issues of the composite service were not addressed either. In addition, we adopt the RDF/S data models rather than the relational data model.

Another area of related research is that of data integration systems (see the review [11]). Our work differs from works in that area in many ways. First, the key focus in these systems is shifted towards resolving specific queries given a set of incomplete data sources, whereas in our work the focus is on constructing a composition of services that is independent of a particular input value (as is the case in EP services composition). That is, we resolve parameterized queries. This necessitates the introduction of an optimization mechanism to filter out “at the execution time” irrelevant services when the composition is invoked with a specific input value. Second, compared to previous query rewriting algorithms [12,11] that are used in the context of these data integration systems, our query rewriting algorithm is compliant with the RDF/RDFS data models. Our proposed algorithm takes into account the RDFS semantic constraints of “subClassOf”, “subPropertyOf” and “domain” and “range” that are defined in domain ontologies.

Our work is related also to web service composition. Service matching and composition algorithms (the reader is referred to the review [2]) in these approaches cannot be applied to DP services, as these algorithms suppose that services can be categorized according to their functionalities in a standard functional ontology. Indeed, the semantics of a DP service cannot be simply captured by a standardized concept in domain ontology (as opposed to EP services); rather this requires the definition of a complete view over a mediated ontology (to capture the semantic relationship between input and output sets of a DP service). For this reason, we have opted to use query rewriting techniques to compose services.

5. Summary and other works achieved in our WSMS framework

In this paper we have presented an approach to compose DP Web services, which exploits existing mature work from data integration. In the proposed approach DP services were described as RDF views over a mediated ontology, the obtained views were then enriched with RDFS semantic constraints and finally an RDF query rewriting algorithm is used to compose DP services based on their associated views.

In addition to the work presented in the paper, we have extended the used RDF rewriting algorithm with a new mechanism to address data privacy when DP services are used to access privacy-sensitive data. Now the rewriting algorithm takes as input the posed query, a set of privacy conditions expressed within a privacy policy and a set of DP services, then it modifies the posed query to accommodate the relevant privacy conditions (that have the form of SPARQL queries) and rewrites it in terms of DP services. The result is a composition of services that respects the privacy conditions defined on the accessed data. This new algorithm is used currently in the health care application domain to compose DP services that give access to patients’ medical records which are the subject to many international privacy regulations (e.g. HIPPA).

6. REFERENCES

[1] Michael J. Carey, "Declarative Data Services: This Is Your Data on SOA," in *IEEE International Conference on Service-Oriented Computing and Applications*, Newport Beach,California, USA, 2007, p. 4.

[2] Schahram Dustdar and Wolfgang Schreiner, "A survey on web services composition," *IJWGS*, vol. 1, no. 1, pp. 1-30, 2005.

[3] Mahmoud Barhamgi, Djamal Benslimane, and Aris M Ouksel, "Composing and optimizing data providing web services," in *Proceedings of the 17th International Conference on World Wide Web, WWW 2008*, Beijing, China, April 21-25, 2008.

[4] Mahmoud Barhamgi, Pierre-Antoine Champin, Djamal Benslimane, and Aris M Ouksel, "Composing Data-Providing Web Services in P2P-Based Collaboration Environments," in *Advanced Information Systems Engineering, 19th International Conference, CAiSE2007*, Trondheim, Norway, June 11-15, 2007, pp. 531-545.

[5] David Martin et al., "Bringing Semantics to Web Services: The OWL-S Approach," in *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, USA, July 6-9, 2004, pp. 26-42.

[6] Amit P Sheth, Karthik Gomadam, and Ajith Ranabahu, "Semantics enhanced Services: METEOR-S, SAWSDL and SA-REST," *IEEE Data Eng. Bull.*, vol. 31, pp. 8-12, 2008.

[7] Dumitru Roman et al., "WWW: WSMO, WSML, and WSMX in a Nutshell," in *The Semantic Web - ASWC 2006, First Asian Semantic Web Conference*, Beijing, China, September 3-7, 2006, pp. 516-522.

[8] Greg Barish and Craig A Knoblock, "An Expressive Language and Efficient Execution System for Software Agents," *J. Artif. Intell. Res. (JAIR)*, vol. 23, pp. 625-666, 2005.

[9] Utkarsh Srivastava, Kamesh Munagala, Jennifer Widom, and Rajeev Motwani, "Query Optimization over Web Services," in *VLDB*, Seoul, Korea, 2006, pp. 355-366.

[10] Michalis Petropoulos, Alin Deutsch, Yannis Papakonstantinou, and Yannis Katsis, "Exporting and interactively querying Web service-accessed sources: The CLIDE System," *ACM Trans. Database Syst.*, vol. 4, no. 32, 2007.

[11] Alon Y. Halevy, "Answering queries using views: A survey," vol. 10, pp. 270-294, 2001.

[12] Rachel Pottinger and Alon Y Halevy, "MiniCon: A scalable algorithm for answering queries using views," *VLDB Journal*, vol. 10, no. 2-3, pp. 182-198, 2001.