Web Service Mediation Through Multi-level Views

Manivasakan Sabesan and Tore Risch

Department of Information Technology, Uppsala University, Sweden
{msabesan, Tore.Risch}@it.uu.se

Abstract. The web Service MEDiator system (WSMED) provides general query capabilities over data accessible through web services by reading WSDL meta-data descriptions. Based on imported meta-data, the user can define views that extract data from the results of calls to web service operations. The views can be queried using SQL. The views are specified in terms of declarative queries that access different web service operations in different ways depending on what view attributes are known in a query. To enable efficient query execution over the views by automatic query transformations the user can provide semantic enrichments of the meta-data with key constraints. We evaluated the effectiveness of our approach over multi-level views of existing web services and show that the key constraint enrichments substantially improve query performance.

Keywords: web service views, query optimization, semantic enrichment

1. Introduction

Web services [4] provide an infrastructure for web applications by defining sets of operations that can be invoked over the web. Web service operations are described by meta-data descriptions of operation signatures, using the *Web Services Description Language* (WSDL) [5]. An important class of operations is to access data through web services, e.g. Google's web page search service [12] and the United States Department of Agriculture nutrition database of foods [27]. However, web services don't support general query or view capabilities; they define only operation signatures.

We have developed a system, WSMED – Web Service MEDiator, to facilitate efficient queries over web services. The view definitions called *WSMED views* are defined in terms of imported WSDL descriptions of web service operations. Furthermore, *multi-level* WSMED views can be defined in terms of other WSMED views. Web services return nested XML structures (i.e. records and collections), which have to be flattened into relational views before they can be queried with SQL. The knowledge how to extract and flatten relevant data from a web service call is defined by the user as queries called *capability definitions* using and object-oriented query language, *WSMED query language* (WQL), which has support for web service data types.

An important semantic enrichment is to allow for the user to associate with a given WSMED view different capability definitions depending on what view attributes are known in a query, the *binding pattern* of the capability definition. The WSMED query

optimizer automatically selects the optimal capability definition for a given query by analyzing its used binding patterns. These view definitions enrich the basic web service operations to support SQL data access queries.

A WSDL operation signature description does not provide any information about which parts of the signature is a key to the data accessed through the operation. As we show, this information is critical for efficient query execution of multi-level WSMED views. Therefore, we allow the user to declare to the system all (compound) keys of a given WSMED view, called *key constraints*.

This paper is organized as follows: Section two describes the architecture of WSMED. Section three gives examples of WSMED view definitions using an existing web service and explains the capability definitions. Section four analyzes the performance of a sample query to verify the effectiveness of query transformations based on the semantic enrichments compared to conventional relational algebra transformations. Section five describes the strategies of the query processor. Section six discusses related work. Finally section seven summarizes the results and indicates future work.

2. The WSMED System

Figure 1a, illustrates WSMED's system components. Imported WSDL meta-data is stored in the *web service meta-database* using a generic *web service schema* that can represent any WSDL definition. The *WSDL Importer* populates the web service meta-database, given the URL of a WSDL document. It reads the WSDL document using the WSDL parser toolkits *WSDL4J* [24] and *Castor* [23]. The retrieved WSDL document is parsed and automatically converted into the format used by the web service meta-database. In addition to the general web service meta-database, WSMED also keeps additional user-provided *WSMED enrichments* in its local store.

The *query processor* exploits the web service descriptions and WSMED enrichments to process queries. The query processor calls the *web service manager* which invokes web service calls using *Simple Object Access Protocol* (SOAP) [13] through the toolkit *SAAJ* [19] to retrieve the result for the user query.

Figure 1b illustrates architectural details of the query processor. The *calculus generator* produces from an SQL query an internal calculus expression in a Datalog dialect [18]. This expression is passed to the *query rewriter* for further processing to produce an equivalent but simpler and more efficient calculus expression.

The query rewriter calls the *view processor* to translate SQL query fragments over the WSMED view into relevant capability definitions that call web service operations. An important task for the query rewriter is to identify overlaps between different subqueries and views calling the same web service operation. This requires knowledge about the key constraints. We will show that such rewrites significantly improve the performance of queries to multi-level views of web services.

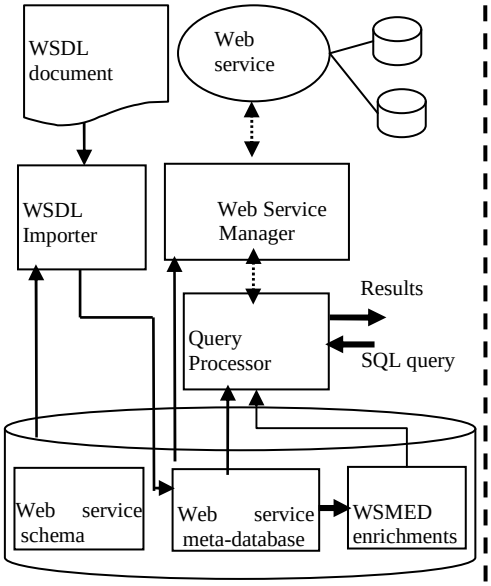

Figure 1a: WSMED components

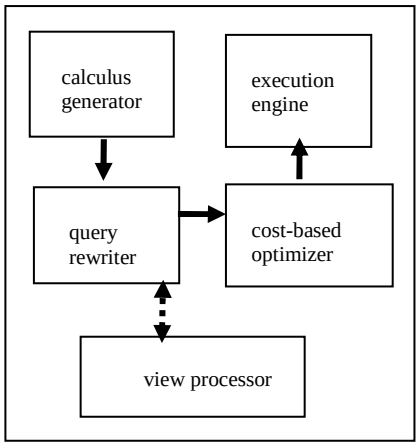

Figure 1b: Query Processor

The rewritten query is finally translated into an algebra expression by a *cost-based optimizer* that uses a generic web service cost model as default. The algebra has operators to invoke web services and to apply external functions implemented in WSDL (e.g. for extraction of data from web service results). The algebra expression is finally interpreted by the *execution engine*. It uses the web service meta-database to generate a SOAP message when a web service operation is called.

3. WSMED Views

To illustrate and evaluate our approach we use a publicly available web service to access and search the National Nutrient Database for US Department of Agriculture [28]. The database contains information about the nutrient content of over 6000 food items. It contains five different operations: *SearchFoodByDescriptions*, *CalculateNutrientValues*, *GetAllFoodGroupCodes*, *GetWeightMethods* and *GetRemainingHits*. We illustrate WSMED by the operation *SeachFoodByDescriptions* to search foods given a *FoodKeywords* or a *FoodGroupCode*. The operation returns *NDBNumber*, *LongDescription*, and *FoodGroupCode* as the results. The WSMED view named *food* in Table 1 allows SQL queries over this web service operation.

Table 1. **WSMED view *food***

ndb	keyword	descr	gpcode
19080	Sweet	Candies	1900
……	……	……	……

For example, the following SQL query to the view *food* retrieves the description of foods that have food group code equal to 1900 and keyword 'Sweet':

```
select descr
from   food
where  gpcode='1900' and keyword ='Sweet';
```

The view *food* is defined as follows:

```
create SQLview food (Charstring ndb,
   Charstring keyword,Charstring descr, Charstring gpcode)
as multidirectional
  ("ffff" select ndb, "",descr, gpcode
          where foodDescr("","")= <ndb,descr,gpcode>)
  ("fffb" select ndb, "",descr
          where foodDescr("",gpcode)= <ndb,descr,gpcode>)
  ("fbff" select ndb,descr,gpcode
          where foodDescr(keyword, "")= <ndb,descr,gpcode>)
  ("fbfb" select ndb, descr
           where foodDescr(keyword,gpcode)
                 = <ndb,descr,gpcode>)
```

Figure 2: WSMED view definition

A given WSMED view can access many different web service operations in different ways. When the user defines a WSMED view he can specify the view by several different declarative queries, called *capability definitions*, using an object oriented query language called WQL having special web service oriented data types. Each capability definition implements a different way of retrieving data through web service operations using WQL. Different capability definitions can be defined based on what view attributes are known or unknown in a query, called the *capability binding patterns*. The query optimizer automatically chooses the most promising capability definitions for a given query to a WSMED view. Each capability definition provides a different way of using the web service operations to retrieve food items. The capability binding patterns of the view *food* are:

1. *ffff*- all the attributes of the view are free in the query. That is, the query does not specify any attribute selection value. In this case the capability definition specifies that all food items should be returned.
2. *fffb*- a value is specified only for fourth attribute *gpcode*. This means that the capability definition returns all food items for a given food group code.
3. *fbff*- a value is specified in the query only for the second attribute *keyword*, i.e. all food items associated with the given keyword are retrieved.
4. *fbfb*- both the values *keyword* and *gpcode* are specified in the query, finding the relevant food items.

In our example query the binding pattern is *fbfb*. The capability definitions are defined as declarative WQL queries that all call a function *foodDescr* in different ways. The function *foodDescr* is defined as a WQL query that wraps the web service operation *SearchFoodByDescription* given two parameters *foodkeywords* and *foodgroupcode*. It selects relevant pieces of a call to the operation *SearchFoodByDescription* to extract the data from the data structure returned by the operation.

To simplify sub-queries and provide heuristics for estimating selectivities, it is important for the system to know what attributes in the view are (compound) keys.

Therefore, the user can specify *key constraints* for a given view and set of attributes by a system function *declare_key*, e.g.:

```
declare_key("food", {"ndb"});
```

Key constraints are not part of WSDL and require knowledge about the semantics of the web service. In our example web service the attribute *ndb* is the key. The attributes are specified as a set of attribute names for a given view *(e.g. {"ndb"})*. Several keys can be specified by several calls to *declare_key*.

The query optimizer may also need to estimate the cost to invoke a capability and the estimated size of its result, i.e. its *fanout*. Costs and fanouts can be specified explicitly by the user if such information is available. However, normally explicit cost information is not available and the cost is then estimated by a *default cost model* that uses available semantic information such as signatures, keys, and binding patterns to roughly estimate costs and fanouts. Key constraints will be shown to be the most important semantic enrichment in our example, and additional costing information is not needed.

3.1 Capability definition function

The function *foodDescr*, used in the capability definitions in Figure 2, has the following definition:

```
1.create function foodDescr (Charstring fkw,
2.                           Charstring fgc)
3.          ->Bag of <Charstring ndb,Charstring descr,
4.                    Charstring gpcode>
5. as select re["NDBNumber"],re["LongDescription"],
6.           re["FoodGroupCode"]
7.    from  Record out, Record re
8.    where out =
9.      cwo("http://ws.strikeiron.com/USDAData?WSDL",
10.         "USDAData",
11.         "SearchFoodByDescription",
12.         {fkw, fgc})
13.    and re in out["SearchFoodByDescriptionResult"];
```

Given a food keyword, *fkw*, and a group code, *fgc*, the function *foodDescr* returns a bag of result rows extracted from the result of calling the web service operation named *SearchFoodByDescription*. Any web service operation can be called by the built-in generic function *cwo* (line 9). Its arguments are the URI of WSDL document that describes the service (line 9), the name of the service (line 10), an operation name (line 11), and the input argument list for the operation (line 12). The result from *cwo* is bound to the query variable *out* (line 8). It holds the output from the web service operation temporarily stored in WSMED's local database. The system automatically converts the input and output messages from the operation into records and sequences where records are used to represent complex XML elements [7] and sequences represent ordered elements. In our example, the argument list holds the parameters *FoodKeywords* and *FoodGroupCode* (line 12). The result *out* is a record structure from which only the attribute *SearchFoodByDescriptionResult* is extracted (line 13). Extractions are specified using the notation *s[k]*, where *s* is a variable holding a record, and *k* is the name of an attribute.

The function *foodDescr* selects relevant parts of the result from the call to the operation. In our example, the relevant attributes are *NDBNumber*, *LongDescription*, and *FoodGroupCode*, which are all attributes of a record stored in the attribute *SearchFoodByDescriptionResult* of the result record. Our example web service operation *SearchFoodByDescription* returns descriptions of all available food items when both attributes *foodkeywords* and *foodgroupcode* are empty strings. On the other hand, if *foodkeywords* is empty but *foodgroupcode* is known, the web service operation will return all food with that group code. Similarly, if *foodgroupcode* is empty but *foodkeywords* is known, the web service operation will return all food with that keyword. If both *foodkeywords* and *foodgroupcode* are non-empty, the operation will return descriptions of all food items of the group code with matching keywords. This knowledge about the semantic of the web service operation *SearchFoodByDescription* is used to define the capability definition function in Figure 2.

4. Impact of key constraints

To illustrate the impact of key constraints we define two views in terms of the WSMED view *food*. The view *foodclasses* is used to classify food items while *fooddescriptions* describes each food item:

```
create view foodclasses(ndb, keyword, gpcode)
     as select ndb,keyword,gpcode from food;
create view fooddescriptions(ndb, descr)
     as select ndb, descr from food;
```

This scenario is natural for our example web service that treats *foodclasses* different from *fooddescriptions*. The following SQL query accesses these views.

```
select fd.descr
from   foodclasses fc, fooddescriptions fd
where  fc.ndb=fd.ndb and fc.gpcode='1900';
```

First the example query is translated by the calculus generator (Figure 1b) into a Datalog expression:

```
Query(l) :- foodclasses(ndb,keyword,gpcode) AND  fooddescrip-
tions (ndb,descr) AND descr=l AND gpcode='1900'
```

The definitions of the views *foodclasses* and *fooddescriptions* are defined in Datalog as[1]:

```
foodclasses(ndb, keyword, gpcode) :- food(ndb, keyword, *,
                gpcode).
fooddescriptions(ndb,descry) :- food(ndb, *, descr, *).
```

Given these view definitions the Datalog expression is transformed by the view processor (Figure 1b) into:

```
Query(l) :- food(ndb,*,*,'1900') AND food(ndb,*,l,*).
```

Here the predicate *food* represents our WSMED view. At this point the added semantics that *ndb* is the key of the view play its vital part. Two predicates *p(k,a)* and *p(k,b)* are equal if *k* is a key and it is then inferred that the other attributes are also

[1] '*' means don't care.

equal, i.e. *b=a* [9]. If a key constraint that *ndb* is the key is specified, this is used by a query rewriter to combine the two calls to *food*:

```
Query(l) :- food(*,*,l,'1900').
```

Without knowing that *ndb* is the key the transformation would not apply and the system would have to join the two references to the view *food* in the expanded query. The simplification is very important to attain a scalable query execution performance as shown in Section 5.

The next step is to select the best capability definition for the query. The heuristics is that if more than one capability definition is applicable, the system chooses the one with the most variables bound. Since *l* is the query output and *gpcode* is given, the binding patterns *ffff* and *fffb* both apply, and the system chooses *fffb* because it is considered cheaper. The call to *food* then becomes:

```
Query(l) :- l=foodDescr("","1900").
```

Similar to relational database optimizers, given the definition of *foodDescr*, a cost based optimizer generates the algebra expression in Figure 3a, which is interpreted by the execution engine. The apply operator (γ) calls a function producing one or several result tuples for a given input tuple and bound arguments [14]. By contrast, Figure 3b shows an execution plan for the non-transformed expression where the system does not know that *ndb* is key. It is using a nested loop join (NLJ) to join the capability definitions. An alternative possible better plan based on hash join (HJ) that materializes the inner web service call is shown in Section 5. In case no costing data is available about the capability definitions (which is the case here), the system uses built in heuristics, i.e. a default cost model. In our case the cost based optimizer produces the plan in Figure 3a, which is optimal for our query.

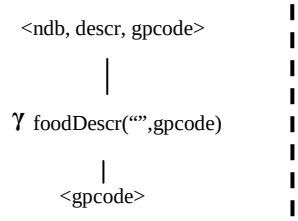

Figure 3a: Full semantic enrichment

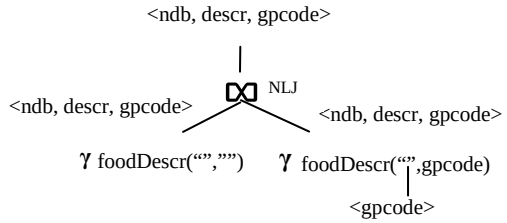

Figure 3b: Naïve execution

5. Query Performance

To determine the impact of semantic enrichments on query processing strategies, we have experimented with four different kinds of query execution strategies. They are:

1. The *naïve implementation* does not use any semantic enrichment at all and no binding pattern heuristics. That is, no *key* is specified for the *food* view definition and no default cost model was used. This makes the capability definition be regarded as a black box called iteratively in a nested loop join since the system does not know

that *foodDescr* returns a large result set when both arguments are empty. The execution plan in Figure 3b shows the naïve plan.

2. With the *default cost model* the system assumes that the view *food* is substantially more expensive to use when attribute *gpcode* is not known than when it is known, i.e. it is cheaper to execute a capability definition where more variables are bound. Still there is no key specified. Figure 5b illustrates the plan using nested loop join.
3. Figure 5a shows the execution plan with the default cost model and a *hash join strategy* where the results from web service operation calls are materialized by using hash join to avoid unnecessary web service calls. This can be done only when the smaller join operand can be materialized in main memory.
4. With *full semantic enrichment* the key of the view is specified. Figure 3a, shows the execution plan. It is clearly optimal.

As shown in Figure 4a, the naïve strategy was the slowest one, somewhat faster than using the default cost model with nested loop join. The default cost model with a hash join strategy scaled significantly better, but requires enough main memory to hold the inner call to *foodDescr*. Figure 4b compares the default cost model with hash join with the performance of full semantic enrichments. The hash join strategy was around five times slower. This clearly shows that semantic enrichment is critical for high performing queries over multi-level views of web services.

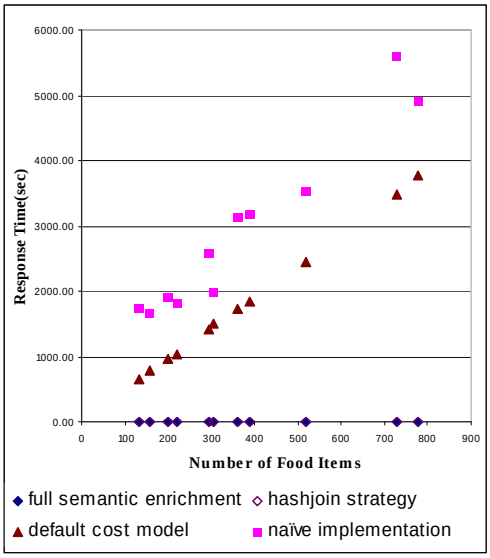

Figure 4a: Performance comparison of four query execution strategies

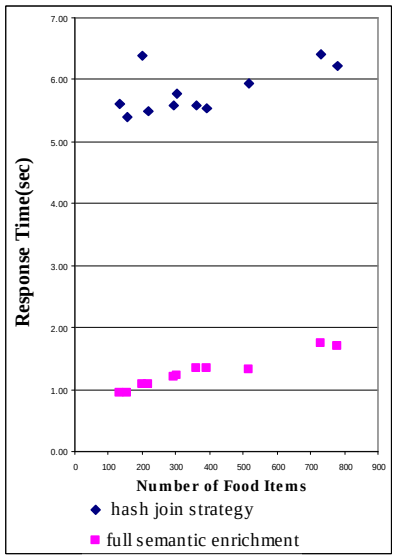

Figure 4b: Performance comparison of hash join and full semantic enrichment execution strategies

The diagrams are based on the experimental results in Table **2** and the experiment was made by using the real values to actually retrieve the results through web service operations. VG, NF, S1, S2, S3, and S4 denote the value used for parameter *gpcode*,

the number of food items (actual fanout), and the execution time in seconds for the four different strategies.

With the naive strategy the system does not use any binding pattern heuristics and will call *foodDescr* with empty strings ($\gamma_{\text{foodDescr}(\text{“”},\text{“”})}$) which produces a large costly result containing all food items in the outer loop. This is clearly very slow.

Table 2. Experimental results

VG	NF	S1	S2	S3	S4
0900	303	1985.14	1512.74	5.77	1.22
0600	390	3177.28	1848.28	5.55	1.33
1400	219	1831.05	1041.74	5.50	1.08
1100	779	4891.13	3785.30	6.22	1.69
2000	157	1655.48	777.31	5.41	0.94
0800	359	3114.28	1723.28	5.59	1.35
0400	201	1914.23	955.38	6.38	1.08
1800	517	3524.34	2452.22	5.93	1.33
2200	132	1741.51	645.03	5.62	0.93

With the default cost model strategy the system assumes that queries over the view *food* produce larger results when the attribute *gpcode* is unknown than when it is known. Based on this the call to *foodDescr* with a known *gpcode* value is placed in the outer loop of a nested loop join. This clearly is a better strategy than the naïve implementation.

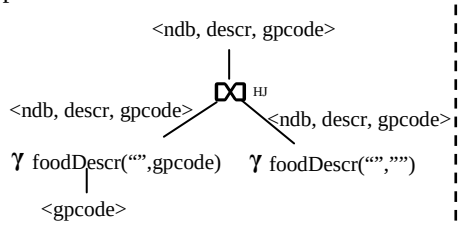

Figure 5a: Execution plan of hash join strategy

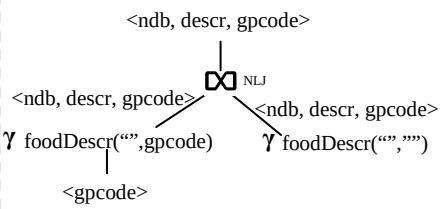

Figure 5b: Execution plan with default cost model

Finally by utilizing key constraints in the WSMED view definition the system will know that the two applications of *foodDescr* can be combined into one call. With this *full enrichment strategy* only one web service operation call is required for execution of the query and no hash join is needed. We notice that this is the fastest and most scalable plan and that it needs no costing knowledge.

6. Related Work

Preliminary results for our method of querying mediated web services were reported in [20].

SOAP [12] and WSDL [5] provide standardized basic interoperation protocols for web services but no query or view capabilities. The SQL 2003 standard [8][26] has facilitates to combine SQL with *XML Query language* (XQuery) [3] to access both ordinary SQL-data and XML documents stored in a relational database. By contrast, we optimize SQL queries to views over data returned by invoking web services and we use semantic query transformations to improve the performance.

The formal basis for using views to query heterogeneous data sources is reviewed in [10][15][25]. As some other information integration approaches, e.g. [11][29], we also use binding patterns as one of our semantic enrichments to access data sources with limited query capabilities. We define semantically enriched declarative views extracting data from the results of each web service operations in terms of an object-oriented query language. In [1] an approach is described for optimizing web service compositions by procedurally traversing ActiveXML documents to select embedded web service calls, without providing view capabilities.

WSMS [22] also provide queries to mediated web services. However, they concentrate on optimizing pipelined execution of web service queries while we utilize semantic enrichments for efficient query processing over multi-level views of web services. XLive [6] is a mediator for integrating heterogeneous sources including web service sources with specific wrappers based on XML standards. In contrast we deploy a generic wrapper that can call any web service.

In particular, unlike the other works, we show that key constraints significantly improve performance of queries to multi-level views of web services with different capabilities.

7. Conclusions and future work

We devised a general approach to query data accessible through web services by defining relational views of data extracted from the result SOAP messages returned by web service operations. Multi-level relational views of web service operations can be defined. The system allows SQL queries over these WSMED views. The view extractions are defined in terms of an object oriented query language. The query performance is heavily influenced by knowledge about the semantics of the specific web service operations invoked and all such information is not provided by standard web service descriptions. Therefore the user can complement a WSMED view with semantic enrichments for better query performance. Our experiments showed that *binding patterns* combined with *key constraints* are essential for scalable performance when other views are defined in terms of WSMED views.

Strategies for parallel pipelined execution strategies of web service operation calls as in WSMS [22] should be investigated. The pruning of superfluous web service operation calls is crucial for performance. The adaptive approaches in [2][17] should be investigated where useless results are dynamically pruned in the early stage of query execution. Currently the semantic enrichments are added manually. Future work could investigate when it is possible to automate this and how to efficiently verify that enrichment is valid. For example, determination of key constraints is currently added manually, and this could be automated by querying the source. Another issue is how to minimize the required semantic enrichments by self tuning cost modeling techniques [16] based on monitoring the behavior of web service calls.

The semantic web is an emerging prominent approach for the future data representations where WSDL working groups are proposing standards to incorporate semantic web representations [21]. It should be investigated how mediate of web services based on such semantic web representations.

Acknowledgements

This work is supported by Sida.

References

[1] S. Abiteboul et al., Lazy query evaluation for active XML, *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Managementof Data*, 227–238, 2004.

[2] R. Avnur, and J. M. Hellerstein, Eddies: Continuously adaptive query processing, *Proc. SIGMOD conference*, 2000.

[3] S.Boag, D.Chamberlin, M.F. Fernández, D.Florescu, J.Robie, and J.Siméon, XQuery 1.0: An XML Query LanguageW3C Candidate Recommendation, *published online at http://www.w3.org/TR/xquery/*, 2006

[4] D.Booth, H.Haas, F.McCabe, E.Newcomer, M.Champion, C.Ferris, and D.Orchard, Web Services Architecture,W3C Working Group Note, *published online at http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/* , 2004

[5] E.Christensen, F.Curbera, G.Meredith, and S. Weerawarana, *Web services description language (WSDL) 1.1.*, W3C, http://www.w3.org/TR/wsdl, 2001.

[6] T.Dang Ngoc, C.Jamard, and N.Travers , XLive : An XML Light Integration Virtual Engine, *Proc. of BDA*, 2005

[7] D.C. Fallside, and P.Walmsley, XML Schema Part 0: Primer Second EditionW3C Recommendation, *published online at http://www.w3.org/TR /xmlschema-0/*, 2004

[8] A.Eisenberg, and J.Melton, SQL/XML is Making Good Progress, *ACM SIGMOD Record*, 31(2), June 2002

[9] G. Fahl, and T. Risch, Query Processing over Object Views of Relational Data, *The VLDB Journal* , 6(4), 261-281, 1997.

[10] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A.Rajaraman, Y. Sagiv, J.D. Ullman, V. Vassalos, and J.Widom, The TSIMMIS Approach to Mediation: Data Models and Languages, *In Journal of Intelligent Information Systems*, 8(2): 117-132, 1997

[11] H.Garcia-Molina, J.D Ullman, and J.Widom, *Database Systems: The Complete Book*, ISBN 0-13-098043-9, Prentice Hall, 1047-1069, 2002.

[12] Google SOAP Search API (Beta), *published online at http://code.google.com/apis /soapsearch/*

[13] M.Gudgin, M.Hadley, N.Mendelsohn, J.Moreau, and H.Frystyk Nielsen, SOAP Version 1.2 Part 1: Messaging Framework,W3C Recommendation, *published online at http://www.w3.org/TR/soap12-part1/* ,2003

[14] L.M.Haas, D. Kossmann, E. Wimmers, and J .Yang, Optimizing queries across diverse data sources, *Proc. Very Large Database Conference(23rd VLDB)*, 1997

[15] A.L.Halevy, Answering queries using views: A survey, *VLDB Journal*, 4(10), 270-294, 2001.

[16] Z.He, B.S.Lee, and R.Snapp, Self-Tuning Cost Modeling of User-Defined Functions in an Object-Relational DBMS, *ACM Transactions on Database Systems*, 30(3), 812-853, 2005.

[17] Z.G.Ives, A.Y.Halvey, and D.S.Weld, Adapting to Source Properties in Processing Data Integration Queries, *Proc. SIGMOD conference*, 2004

[18] W. Litwin, and T. Risch, Main Memory Oriented Optimization of OO Queries using Typed Datalog with Foreign Predicates, *Proc. IEEE Transactions on Knowledge and Data Engineering*, 4(6), pp. 517-528, 1992

[19] SAAJ Project, *published online at https://saaj.dev.java.net/*

[20] M.Sabesan, T.Risch, and G.Wikramanayake, Querying Mediated Web Services**,** *Proc. 8th International Information Technology Conference (IITC 2006*), 2006

[21] Semantic Web Activity, W3C Technology and Society domain, *published online at http://www.w3.org/2001/sw/*

[22] U.Srivastava, J.Widom, K.Munagala, and R.Motwani, Query Optimization over Web Services, *Proc Very Large Database Conference(VLDB 2006)*, 2006

[23] The Castor Project, *published online at http://www.castor.org/index.html*

[24] The Web Services Description Language for Java Tool kit(WSDL4J), *published online http://sourceforge.net/projects/wsdl4j*

[25] J.D.Ullman, Information Integration Using Logical Views, *Proc. 6th International Conference on Database Theory (ICDT '97)*, 19-40, 1997.

[26] XML-Related specifications (SQL/XML), *published online at http://www.sqlx.org/SQL-XML-documents/5FCD-14-XML-2004-07.pdf*, 2005

[27] Web Service USDAData, *published online http://ws.strikeiron.com/USDAData?DOC&page=proxy*

[28] WSDL document for USDAData web service, *published online http://ws.strikeiron.com/USDAData?WSDL*

[29] V.Zadorozhny, L.Raschid, M.E.Vidal, T.Urban, and L.Bright, Efficient Evaluation of Queries in a Mediator for WebSources, *Proc. of the 2002 ACM SIGMOD international conference on Management of data*, 85-96, 2002.