Querying Mediated Web Services

M. Sabesan
Dept. of Information Technology
Uppsala University
Sweden.
Tel: +46 18 471 7778
mailto: msabesan@it.uu.se

Tore Risch
Dept. of Information Technology
Uppsala University
Sweden
Tore.Risch@it.uu.se

Gihan Wikramanayake
University of Colombo
School of Computing
Sri Lanka
gnw@ucsc.cmb.ac.lk

Abstract

Web services provide interfaces to web resources described by WSDL interface definitions. The Web Service MEDiator system (WSMED) enables querying data accessible through web services. WSMED allows web service meta-data to be automatically extracted from any WSDL description. Then views can be created in terms of the imported meta-data and queried using SQL. To enhance query performance, WSMED permits to complement the automatically extracted web service meta-data with semantic enrichments. A WSMED prototype is being evaluated over existing web services to verify the effectiveness of the approach. It is being investigated how semantic enrichments and other query optimization methods are useful for efficient querying of mediated web services.

Keywords: Web Services - Views - Query Processing - Binding Patterns

1. Introduction

The growth of the Internet and the emergence of XML for data interchange have increased the importance of web services [27] incorporating standards such as SOAP [24], WSDL [1] and XML Schema [25]. The interface description language WSDL provides an XML format that describes interfaces to a web service as a collection of self-contained descriptions of *operations*. XML Schema is used for describing types and structure of messages used in WSDL operation descriptions. SOAP provides a standardized system communication protocol.

One of the important applications of web services is to search for data accessible through web service operations. However, unlike relational databases web services do not provide any high-level query language or support view capabilities for querying the data. Therefore the user has to access a web service by implementing navigational programs in a high level language such as Java, C that invokes the web service operations to retrieve the desired information. In this work we provide the capability to define declarative queries and views over data extracted from web services.

The WSMED (Web Service MEDiator) engine permits integration of data from heterogeneous remote systems accessible through web services. It permits SQL queries over data produced by distributed web service enabled data sources. Any web service can be accessed by importing its WSDL interface descriptions. The imported WSDL interface descriptions are stored in WSMED as meta-data. To enhance query processing, the meta-data can be complemented with user-provided *semantic enrichments* describing properties of data in the sources not provided by WSDL. The user can define views of data from the web services in terms of the imported meta-data. Applications then query these views transparently as if all the data was stored in a relational database. SQL is used by applications and users for querying these views. Composed SQL views can be defined in terms of the web service views. A mediator query optimizer automatically decides strategies to access wrapped web service operations to minimize the execution cost.

The remainder of this paper is organized as follows. Section 2 formulates the research issues we are addressing. Section 3 discusses related work. Section 4 describes a motivating example used for our study. Section 5 overviews the architecture of WSMED. Finally, section 6 summarizes our approach and outlines future directions.

2. Research issues

The development of a web service based mediator prototype is expected to provide insights into a number of research questions:

1. To what extent can the web service standards, such as WDSL, SOAP, and XML-Schema, be automatically utilized by a mediator engine to query the sources efficiently and scalable?

2. How can views in a high level query language such as SQL be defined in terms of imported web service descriptions?
3. How can the modern query optimization and rewrite techniques be used to provide efficient and scalable access that optimally utilizes the limited data access and update capabilities of different web services?
4. What minimal set of extra semantic enrichment is needed in addition to the current web service standards in order to provide scalable access through the views?

3. Related work

SOAP and WSDL provides standardized basic interoperation protocols for web services but no query or view capabilities. Several systems have been built to query XML in general, e.g. [3] [5] [8] [13] [14], without providing support for web service access. WSMS [17] also studies queries to mediated web services. However, they currently concentrate on pipelined execution of web service queries by importing WSDL descriptions, while we concentrate on utilizing semantic enrichments for scalable execution of database queries to high level views of wrapped web service operations. As some other view integration approaches [2][4][6][7][18][19] we also utilize binding patterns as one of our semantic enrichments to access data sources with limited data capabilities. We are investigating whether the binding pattern mechanism in [12] and other semantic enrichments such as keys, cost estimates, and foreign key relationships can improve the processing time for queries over data accessible through web services.

Chocolate [10] also creates views of web services based on importing WSDL descriptions. Some user modification of the views is possible by adding/deleting/renaming attributes, changing data types, and splitting/merging views, but not the kinds of semantics enrichments we are considering for efficient queries.

[9] [11] propose to use semantic query optimization techniques for optimization of heterogeneous multi-database queries. Some of those methods could be applicable in our environment as well but we aim at minimizing the semantic enrichments needed for scalable query execution.

4. Motivating example

For the initial evaluation we use a publicly available web service that provides access and search to the National Nutrient Database for US Department of Agriculture. It is available through a WSDL description [23]. The database contains information about the nutrient content of over 6000 food items. We illustrate WSMED by the operation *SeachFoodByDescription* to search foods given a food key word or a food group code. The operation returns *ndbnumber*, *longdescription* and *foodgroupcode* as the results. A view (Table 1) is defined in WSMED to execute queries over this web service operation.

ndb	keyword	descr	gpcode
19080	Sweet	Candies, semisweet chocolate	1900
…….	…….	……	…….

Table 1. food view

The following SQL query uses the above to retrieve 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';
```

Figure 1 shows the WSMED view definition of *food*.

```
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 1. SQL view

The view is defined in terms of a generic web service data access query function that wraps the web service operation *foodDescr*. Several different web service *search definitions* are used in the view definition to specify the web service operation calls depending on how a query to the view is specified. For example, the strategies may depend on what view attributes are known or unknown in the query, i.e. the *view binding patterns*. In our example the view binding patterns are:

1. *ffff*- all the attributes of the view *food* are free. That is, the query does not to specify any attribute selection value.
2. *fbff*- a value is specified in the query only for the attribute *keyword*.
3. *fffb*- a value is specified only for *groupcode.*
4. *fbfb*- both the values *keyword* and *groupcode* are specified.

Another important semantic enrichment we use is that the attribute *ndb* turns out to be the key in *food* and specified by calling the system function *declare_key*:

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

This information is important for estimating costs to access web service operations and for unifying redundant web service calls.

5. WSMED system

This section gives an overview of the WSMED system. The *web service schema* describes the internal *web service meta-database* in WSMED that stores WSDL definitions and semantic enrichments. System components sub section describes the sub components of the system.

5.1 Web service schema

Figure 2 shows an extended ER-diagram of the web service schema that represents the WSDL core elements [1] *service, operation,* and *element*.

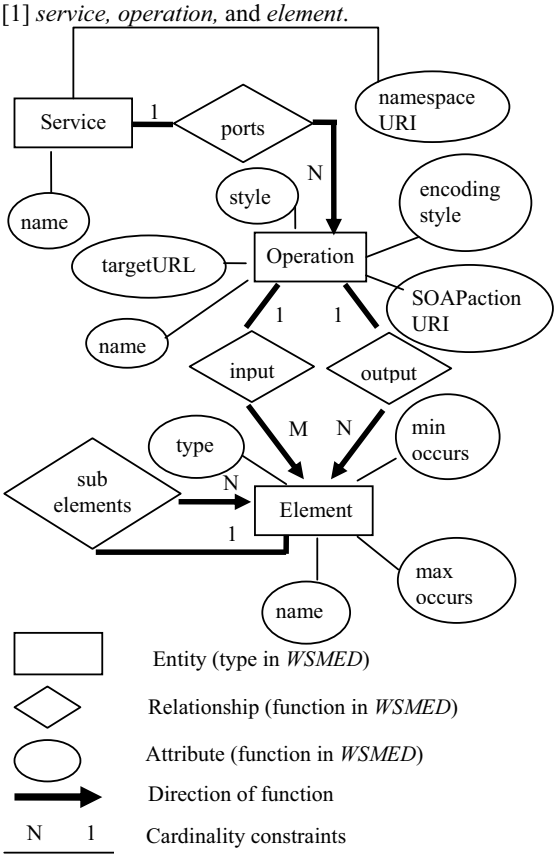


Figure 2. The web service schema

A *service* describes a particular web service and supports a set of *operations* and depicted by the *Service* entity. The *name* attribute corresponds to the name of the web service; *namespaceURI* is a URI reference to identify a web service. Each *operation* named *name* represents a procedure that can be invoked through the web service. The *style* indicates whether the operation is RPC-oriented (messages have parameters and return values) or document-oriented (messages containing documents). The *encoding style* is a URI that indicates the encoding rules for data in the SOAP messages. The *target URL* determines the address of the SOAP message. The *SOAPactionURI* used to identify the intent of a SOAP Message.

WSDL data type	**WSMED data type**	**WSDL data type**	**WSMED data type**
anyURI	Charstring	Integer	Real
baseBinary	Charstring	Language	Charstring
Boolean	Boolean	Long	Integer
Byte	Integer	Name	Charstring
Date	Date	NCName	Charstring
dateTime	Charstring	NegativeInteger	Real
Decimal	Real	NMTOKEN	Charstring
Double	Real	NMTOKENS	Charstring
Duration	Charstring	NonnegativeInteger	Real
ENTITIES	Charstring	nonPositiveInteger	real
ENTITY	Charstring	NormalizedString	Charstring
Float	Real	NOTATION	Charstring
gDay	Charstring	positiveInteger	real
gMonth	Charstring	QName	Charstring
gMonthDay	Charstring	Short	integer
gYear	Charstring	String	Charstring
gYearMonth	Charstring	Time	Time
hexBinary	Charstring	Token	Charstring
ID	XS_ID	unsignedByte	integer
IDREF	XML	unsignedInt	integer
IDREFS	XML	unsignedLong	integer
Int	Integer	unsignedShort	integer

Table 2: Mappings between WSDL and WSMED data types

The *ports* relationship represents the association between a service and its operations. Each operation has a number of *input* elements and *output* elements. It is an abstract definition of the data being transmitted and is associated with a data type, i.e. the input and output elements define the *signature* of the operation. Complex data elements may consist of other *sub-elements* where each sub-element has a data type, along with a name and the number of minimum and maximum occurrences within the super element. The WSMED uses a *conversion table* (Table 2) for type conversion from/to a XML

Schema data type to/from the corresponding data type in WSMED.

4.2 System components

The WSMED prototype is illustrated by Figure 3. For high level and scalable query execution over the data provided by a web service, WSMED requires representation of its WSDL meta-data in the internal database of WSMED. The *WSDL Importer* can import any WSDL description using the tool kits *WSDL4J* [22] and *Castor* [20]. For a given *URL,* the retrieved WSDL document is parsed and converted into the format used by WSMED's predefined *web service schema* for WSDL (Figure 2). The extracted meta-data is stored as *web service descriptions* in WSMED's internal database.

In addition to the web service descriptions, WSMED furthermore keeps optional WSMED enrichments in its local store extending the basic web service schema. These WSMED enrichments include information necessary to efficiently translate SQL queries into web service operation calls using different strategies depending on the semantics of the wrapped operations. A functional and object-oriented query language [15] [16] is used for defining the SQL views in terms of calls to one or several web service operations.

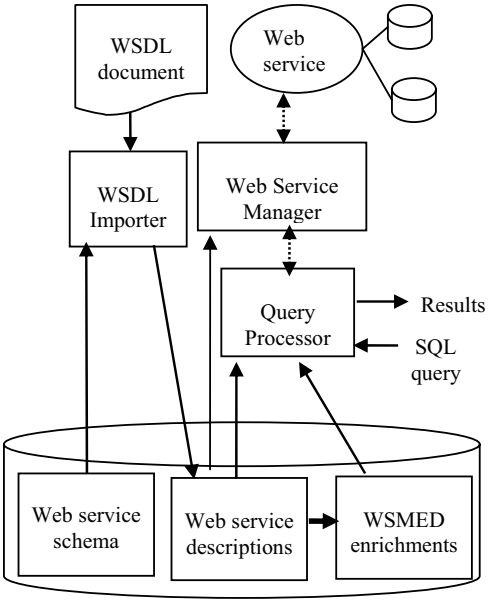


Figure 3: WSMED system components

The *query processo*r exploits the web service descriptions and WSMED enrichments to process queries. The query processor calls the *web service manager* component, implemented using the APIs *SAAJ* [21] and *JDOM* [26]. The web service manager is accountable for invoking web service calls in order to retrieve the result for the user query. For this, it creates a message format able to invoke the web service operation using the SOAP [24] protocol over the communication network. Then the structure of the SOAP reply message is converted by the web service manager into the format used by the query processor for further processing.

Figure 4 illustrates architectural details of the query processor. The *calculus generator* produces a domain calculus expression from an SQL query. This expression is passed to the *query rewriter* for further processing to produce an equivalent but simpler domain calculus expression. The query rewriter calls the *view expander* to process the WSMED view and search definitions to translate an SQL query over the WSMED view into a corresponding calculus query directly calling web service operations. An important task for the query rewriter is to perform common sub-expression elimination between different sub-queries calling the same web service operation in order to minimize the number of calls. This requires knowledge about what argument or result in a web service operation can be used as a *key*.

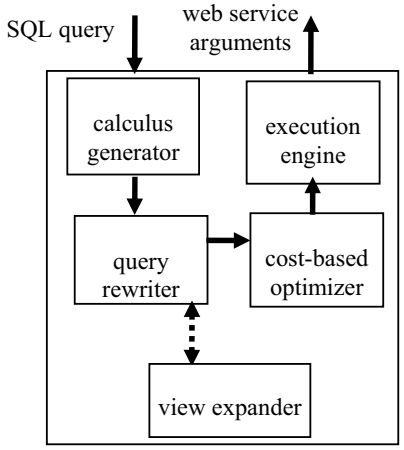
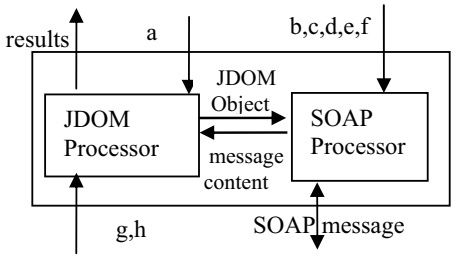


Figure 4: Query Processor

The *cost-based optimizer* uses a generic web service cost model to produce and optimize the user query execution plan represented as an *algebra expression*. 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 descriptions of operations in the web service meta-database to call the web service operations in the execution plan. A call to the web service manager is specified by *web service properties* such as

SOAPactionURI, style, encodingstyle, namespaceURI, and targetURL (Figure 2). Furthermore, it contains the actual parameters of the operation, called the *input elements*.



a - input elements b - SOAPactionURI
c - style d -encodingstyle
e - namespaceURI f - targetURL
g - type h-maxoccurs

Figure 5: Web service manager

As shown by Figure 5 , the web service manager uses two sub components to create a SOAP message: The *JDOM processor* and the *SOAP Processor.* JDOM processor is implemented using JDOM APIs while SOAP processor using SAAJ (SOAP with Attachments API for Java) APIs.

The SOAP processor creates a request SOAP message and sends it over the network to invoke the web service operation call. The SOAP body is the essential part of the SOAP message and it provides a simple mechanism for exchanging mandatory information intended for the web service operation call. The JDOM processor creates a JDOM object that represents the mandatory information for the SOAP body. The SOAP processor requires additional information for SOAP message creation. The XML format of the JDOM object is shown by Figure 6. The input elements to the JDOM processor contain names of elements such as *SearchFoodByDescription*, *FoodKeywords,* and *FoodGroupCode,* and the values for the elements, e.g. *Sweet* and *1900*.

```
<SearchFoodByDescription>
 <FoodKeywords>Sweet</FoodKeywords>
 <FoodGroupCode>1900</FoodGroupCode>
</SearchFoodByDescription>
```

Figure 6: The request XML document

The SOAP processor creates the final SOAP message given the JDOM object (Figure 5) and the web service properties and sends it over the network to invoke the web service operation call. The response of the remote web service call is received as a SOAP message. Contents of the SOAP message is extracted by the SOAP processor and sent it to the JDOM processor as a Java object. Then the JDOM processor converts the message content into a JDOM object. The JDOM object represents the XML format as shown in Figure 7.

The JDOM processor requires the output elements' properties (Figure 2) of the web service operation call such as *type* and *maxoccurs* to constructs the result object of the web service call by extracting information from the JDOM object. The type of output elements is used by the JDOM processor for the conversion of XML data format and data format used by WSMED while *maxoccurs* is used to determine the result object structure. The JDOM processor retrieves the values for *type* and *maxoccurs* from the web service meta-database. After that the result object is sent back to the execution engine.

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XML
Schema">
  <soap:Body>
   <SearchFoodByDescriptionResponse
    xmlns="http://www.strikeiron.com/">
   <SearchFoodByDescriptionResult>
    <SearchByKeywordsOutput>
      <NDBNumber>02044</NDBNumber>
      <LongDescription>Basil, fresh </LongDescription>
      <FoodGroupCode>0200</FoodGroupCode>
    </SearchByKeywordsOutput>
   </SearchFoodByDescriptionResult>
   <ResponseStatus>
      <response_code>0</response_code>
   <response_string>Success</response_string>
   </ResponseStatus>
   </SearchFoodByDescriptionResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 7: The response XML document

The execution engine performs further post processing specified by the algebra such as filtering and data transformation before the query result is delivered to the user.

6. Conclusions and future work

WSMED provides primitives for defining relational views of web service data and supports SQL queries over the views. We described the architecture of the WSMED prototype system to evaluate different implementation alternatives. The architecture utilizes publically available subsystems such as WSDL4J, JDOM,SAAJ and Castor to allow the WSMED query engine to access any web service based on web service meta-data in a WSDL description.

The performance of queries varies very substantially depending on what processing strategy is used. Therefore, we are investigating query processing strategies using our prototype and existing web services. The query performance turns out to be very much dependent on knowledge about some semantics of the specific web service operations invoked not provided by WSDL and therefore the user can provide these semantic enrichments to WSDM for better query optimization. We investigate how to utilize capability based query optimization [4] to process the SQL queries over semantically enriched views of web services.

Future work includes how to minimize the required semantic enrichments by using heuristic methods or by adaptive query processing techniques based on monitoring the behavior of web service calls.

So far we assume all web service operations used in queries are side effect free. A further future research issue is semantic enrichments to allow SQL updates of web service data views.

References

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

[2] M.Duschka, *Query Planning and Optimization in Information Integration*, Ph.D. Thesis, Stanford University, Computer Science Technical Report STAN-CS-TR-97-1598, 1997.

[3] L.Ennser, C.Delporte, M.Oba, and K.Sunil, Integrating XML with DB2 XML Extender and DB2 Text Extender, *http://www.redbooks.ibm.com/redbooks/pdfs/sg246130.pdf , IBM Corp*., 2001.

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

[5] G.Gardarin, A.Mensch, and A.Tomasic, An Introduction to the e-XML Data Integration Suite, *Proc. 8th International Conference on Extending Database Technology (EDBT '02)*, 2002, pp. 297–306.

[6] A.Gupta, L.Haas, and Y. Papakonstantinou, Capabilities-Based Query Rewriting in Mediator Systems, *Proc. 4th International Conference on Parallel and Distributed Information Systems (PDIS '96)*, 1996, pp 170-181.

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

[8] A. Halverson, V.Josifovski, G.Lohman, H.Pirahesh, and M. Mörschel, ROX: Relational Over XML, *Proc. 30th VLDB Conference (VLDB '04)*, 2004, pp 264-275.

[9] C.Hsu, and C.A.Knoblock, Semantic Query Optimization for Query Plans of Heterogeneous Multidatabase Systems, *IEEE Transactions on Knowledge and Data Engineering ,* 12(6), 2000, pp 959-978.

[10] V.Josifovski, S.Massmann, and F.Naumann, Super-Fast XML Wrapper Generation in DB2: A Demonstration, *Proc. International Conference of Data Engineering, (ICDE'03)*, 2003, pp 756-758.

[11] C.Li, Describing and Utilizing Constraints to Answer Queries in Data-Integration Systems, *Proc. IJCAI 2003 workshop on Information Integration on the Web (IIWeb-03)*, 2003, pp 163-168.

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

[13] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, Lore: A Database Management System for Semistructured Data, *SIGMOD Record*, 26(3), ACM Press , New York, USA, 1997, pp 54-66.

[14] J.Naughton, et al. , The Niagara Internet Query System, *IEEE Data Engineering bulletin*, 24(2) , 2001, pp. 27-33.

[15] T.Risch and V.Josifovski, Distributed Data Integration by Object-Oriented Mediator Servers, *Concurrency and Computation: Practice and Experience J.*, 13(11), John Wiley & Sons, 2001, pp 933-953.

[16] T.Risch, V.Josifovski, and T.Katchaounov, Functional Data Integration in a Distributed Mediator System, in P.Gray, L.Kerschberg, P.King, and A.Poulovassilis (eds.): *Functional Approach to Data Management - Modeling, Analyzing and Integrating Heterogeneous Data,* ISBN 3-540-00375-4,Springer,2003,pp 211-238.

[17] U.Srivastava, J.Widom, K.Munagala, and R.Motwani, Query Optimization over Web Services, *http://hake.stanford.edu/wsms/index.htm*, 2005.

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

[19] V.Zadorozhny, L.Raschid, T.Urhan, M.E.Vidal, and L.Bright, Efficient Evaluation of Queries in a Mediator for Web Sources, *Proc. International conference on Management of data (SIGMOD '02)*, 2002, pp 85-96.

[20] http://www.castor.org/index.html.

[21] https://saaj.dev.java.net/.

[22] http://sourceforge.net/projects/wsdl4j.

[23] http://ws.strikeiron.com/USDAData?WSDL

[24] http://www.w3.org/TR/soap12-part1/.

[25] http://www.w3.org/TR/2004/REC-xml-20040204/

[26] http://www.jdom.org/

[27] http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/