Query Set Specification Language (QSSL)

Michalis Petropoulos Alin Deutsch Yannis Papakonstantinou
University of California, San Diego
{mpetropo,deutsch,yannis}@cs.ucsd.edu

ABSTRACT

Applications require access to multiple information sources and the data of other applications. WSDL-based web services are becoming a popular way of making information sources available on the web and, hence, to applications that need to consume them – often via data integration systems that combine the data of multiple sources. We argue that the function signature paradigm that is used today by web services cannot capture the query capabilities provided by structurally rich and functionally powerful information sources, such as relational databases. We propose the Query Set Specification Language (QSSL) that allows the concise description of sets of parameterized XPath queries. A QSS is embedded in a WSDL specification to form a specialized type of web services, called Data Services. Data Services connect the calls that the source accepts with the underlying schema. QSSL will be enhanced to describe subsets of XQuery expressions beyond XPath ones.

1. INTRODUCTION

Web Services Description Language (WSDL) [5] provides an XML format for describing functions offered via web services. The function signatures typically have fixed numbers of input and output parameters. However, the "function" paradigm is not adequate when the software components behind the web services are databases. One typically associates one function with each parameterized query but this is problematic since databases often allow a large or even infinite set of parameterized queries over their schema. For example, the administrator of a product catalog database may want to allow any query that selects products by a combination of selection conditions on the product's attributes. Assuming the product has, say, 10 attributes, it is obviously impractical to specify 2^{10} function signatures.[1]

In addition, the function paradigm does not state explicitly either the relationship between the input parameters and the output or the semantic connections the available functions have with each other and with the underlying database. We classify such *web services* as *functional* and we argue that they are inappropriate for exporting structurally rich and functionally powerful information sources, such as relational and emerging XML databases.

We present a WSDL extension that enables *Data Services*, which overcome the shortcomings of functional web services. A data service exports the XML Schema [7] of an XML view. The data service also provides a set of parameterized queries that can be executed against the view. Hence the relationship between the input and output parameters is explicit, since the input corresponds to a query and the output to its result. Note that the view typically (but not necessarily) corresponds to a part of the underlying database.

The Query Set Specification Language (QSSL) is a WSDL extension that, given an underlying database, allows the concise and semantically meaningful description of set of parameterized queries. The set may be very large or even infinite, since powerful information sources (such as relational databases) support a large number of parameterized queries. Consequently, QSSL must be able to describe sets of parameterized queries without requiring exhaustive enumeration of them.

QSSL concisely describes sets of tree pattern (subset of XPath) queries. We plan to extend to subsets of XQuery. It lends itself to a compact and intuitive visual notation that forms the basis of an under-development GUI that allows the specification of QSSs.

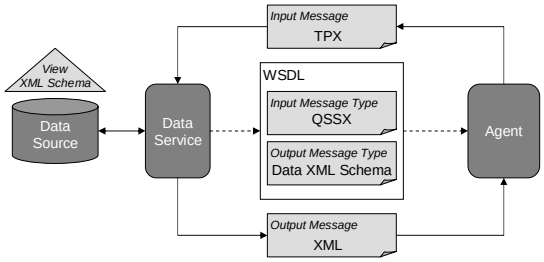

Figure 1. Data Service Architecture.

[1] In the particular example one can resolve the issue simply by allowing some input parameters to be null. This situation is generalized by QSS to capture multiple function signatures in just one WSDL operation [5].

Figure 1 shows the architecture of a data service published by a data source with a given view XML Schema. The query capabilities exported by a data service are published as a WSDL specification [5] that provides an agent with the means to formulate valid and acceptable queries and to be aware of the structure of the result. Notice that we translate QSSs into XML Schemas and we are thus compatible with the WSDL specification. The data service receives an input message from the agent and replies with an output message or a fault. The input message is a tree pattern (TP) query (subset of XPath), defined in Section 2, expressed in the TPX XML format, and the output message is an XML tree. The set of acceptable tree pattern queries (i.e., the set of acceptable input messages) is a Query Set Specification (QSS), defined in Section 2. A QSS describes the possibly infinite set of parameterized queries that are acceptable. The QSS is translated into an XML Schema (QSSX) describing the acceptable TPX messages.

1.1 Example

The running example is based on the XML Schema in Figure 2a that describes the structure of an airline database holding information about flights. The schema describes the flights carried out by one or more airline companies, where each flight has an origin and destination (`from` and `to` elements) and is scheduled at least once per week. In turn, each flight has one or more legs with a code, an origin and a destination and optionally the type of the aircraft used. Note that the schema of the actual airline database may be "richer" but we focus on the part that the database administrator exposes.

The database administrator allows queries that are having any combination of the following conditions:

- The name of the airline company is specified
- The origin and destination of one or more flights is optionally specified
- A day of the week is specified
- The origin of zero or more legs is optionally specified
- The destination of zero or more legs is optionally specified.
- The aircraft used for zero or more legs is optionally specified.

Notice that one may also specify combinations of origin, destination, and aircraft for legs. For the sake of the example, we also allow one to check whether a flight has a leg (existential condition).

The queries may return "airline" or "flight" elements.

This document presents the process of exporting such query capabilities using a data service. More specifically, Section 2 defines the query language and the query set specification language, and shows how QSSL accommodates recursive schemas. Section 3 describes the XML syntax of TP queries and QSSs. Section 4 presents possible QSSL extensions and related work is discussed in Section 5.

2. SPECIFYING QUERIES AND QUERY SETS

We consider data services that support queries defined by a class of XPath expressions consisting of node tests, navigation along the child axis '/' and the descendant axis '//', and predicates denoted by '[]'. The established convention for representing this class of XPath expressions is to use *tree pattern* (*TP*) queries [2, 14]. We believe that support for tree pattern queries is a minimum data service requirement, since tree patterns are widely used in current applications, and since they are crucial building blocks of more expressive query languages such as XQuery [4]. Moreover, tree patterns provide an excellent visual paradigm which enables graphical user interfaces for constructing applications that produce and consume data services. For example, the XPath expression

```
flights/airline[name='Delta']/flight[from=
'JFK'][to='LAX'][day='MON'][leg[to='LAS']]
```

is represented by the tree pattern query in Figure 2b.

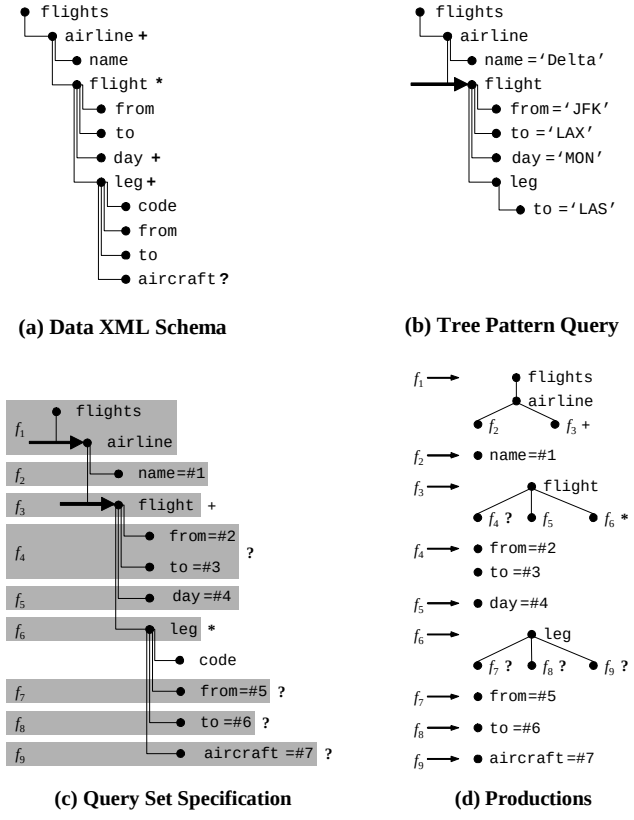

Figure 2. Airline Example.

The arrow pointing to the `flight` element node denotes the *result node* of the tree pattern query.

2.1 Query Set Specifications

We define a data service by specifying the set of tree pattern queries it supports. First, we introduce *parameterized tree patterns (PTPs)*, which are TP queries where the constants are replaced with parameters. A PTP query specifies an infinite set of TP queries, each TP corresponding to a parameter instantiation. A data service exports a possibly infinite set of such parameterized tree pattern queries. This set is succinctly encoded using a *Query Set Specification (QSS)*.

Definition 1 (Query Set Specification). A QSS is a 5-tuple $<F, \Sigma, P, S, R>$, where:

- F is a finite set called the *tree fragment names*.
- Σ is a finite set, disjoint from F, called the *element node names*.
- P is a finite set of *productions* of general form $f \rightarrow tf_1, \ldots, tf_n$ where $f \in F$ is a tree fragment name and each tf_i is a *tree fragment*. A *tree fragment* is a labeled tree consisting of:
  - *Element nodes* with labels from Σ. Leaf element nodes may be additionally labeled with a parameterized equality predicate of the form $=\#i$, where $\#i$ is a *parameter* and i is an integer.
  - *Tree fragment nodes* n labeled with a name $name(n) \in F$ and an occurrence constraint $occ(n) \in \{1, ?, +, *\}$. Tree fragment nodes can only appear as leaf nodes of a tree fragment. We often omit the occurrence constraint '1'.
  - *Edges* e either of *child* type, denoted by straight lines, or of *descendant* type, denoted by dashed lines.
- $S \in F$ is the start tree fragment name.
- $R \in \Sigma$ is a set called *result node names*. ■

Example. The QSS describing the airline data service from our motivating example is $A=<F, \Sigma, P, S, R>$, where $F=\{f_1, \ldots, f_9\}$, $\Sigma=\{$`flights`, `airline`, `name`, `flight`, `from`, `to`, `day`, `leg`, `aircraft`$\}$, P is the set of productions shown in Figure 2d, $S=f_1$ and $R=\{$`airline`, `flight`$\}$. ■

A compact visual representation of this QSS is given in Figure 2c, where tree fragments are depicted by shaded boxes with occurrence constraints to their right. This visual representation is the basis of our under-development GUI for specifying QSSs by displaying the XML schema and using drag-and-drop actions.

Given the similarity between QSSs and extended context-free grammars [1], we define the set of parameterized tree pattern queries described by a QSS analogously to the language generated by a grammar. A QSS defines the set of PTPs whose result node is in R and whose pattern is yielded by a sequence of *derivation steps* starting from the start fragment name S. At any step, given a tree fragment node n, the derivation step replaces n with the tree fragments on the right hand side of a production that has n on the left hand side. Depending on the occurrence constraint labeling n, the derivation step might replace it more than once or not at all. More specifically, if $occ(n)=1$, then n is replaced with the corresponding tree fragments exactly once, and they all become children of n's parent. If $occ(n)=?$, then n is nondeterministically either deleted or relabeled with $occ(n)=1$ before replacement. If $occ(n)=+$, then for a nondeterministically chosen $k \geq 1$, n is replaced with k copies of n, all siblings, with occurrence labels set to 1. If $occ(n)=*$, then n is labeled nondeterministically with $occ(n)=?$ or $occ(n)=+$ first. The parameters introduced in every step are freshly renamed such that their name is unique across the tree fragment obtained so far.

A TP query is *accepted* by a QSS A if and only if it corresponds to an instantiation of the parameters of a PTP query from the set defined by A. We denote with $TP(A)$ the set of TP queries accepted by A.

Example. Figure 3 shows the sequence of derivations steps, denoted by the $\Rightarrow$ symbol, that obtains the corresponding PTP query pq of the TP query q in Figure 2b. Note how the third derivation step replaces f_4 with the corresponding tree fragments, and how the fourth derivation step deletes f_7 and f_9. After the final derivation step, the node labeled with the `flight` result node name is chosen, thus forming a PTP query pq. When pq's parameters `[#1,#2,#3,#4,#5]` are instantiated with the constants `['Delta', 'JFK',`

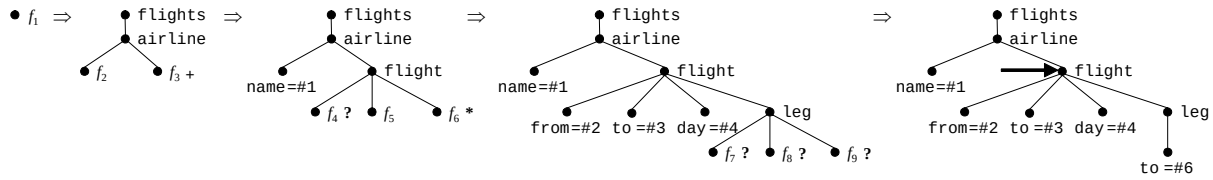

Figure 3. Example Derivation.

'LAX', 'MON', 'LAS'], we obtain the TP query q from Figure 2b. Therefore, *q* is accepted by *A*. ■

When the XML Schema is recursive, it describes documents of arbitrary depth. On these documents, there are TP queries of arbitrary pattern height with non-empty answer and it makes sense to export them in a data service.

Despite its fixed size (determined by the XML schema), a QSS can specify such arbitrarily deep TP queries.

Example. The recursive XML Schema in Figure 4a captures the structure of a family tree. Figure 4b shows a TP query that returns the persons found at any depth that are named "Kevin" and were born in "NY" such that at least one of his descendants is married to a person also named "Kevin" and also born in "NY". Recall that the dotted lines in Figure 4b denote descendant edges.

A QSS that accepts, among others, the corresponding PTP query of the TP query in Figure 4b is shown in Figure 4c. Note the last node, labeled with the tree fragment name f_2, representing the recursion in the schema. Formally, the above QSS is defined as $B=<F, \Sigma, P, S, R>$, where $F=\{f_1,\ldots, f_8\}$, $\Sigma=\{$`familyTree`, `person`, `name`, `place`, `spouse`, `children`$\}$, P is the set of productions shown in Figure 4d, $S=f_1$ and $R=\{$`person`$\}$. Note how the recursion in productions f_2, f_5 and f_8 allows for derivations of arbitrary length. It is easy to see that the PTP query corresponding to the TP query in Figure 4b is obtained by a sequence of derivation steps using the production associated to f_2 twice. ■

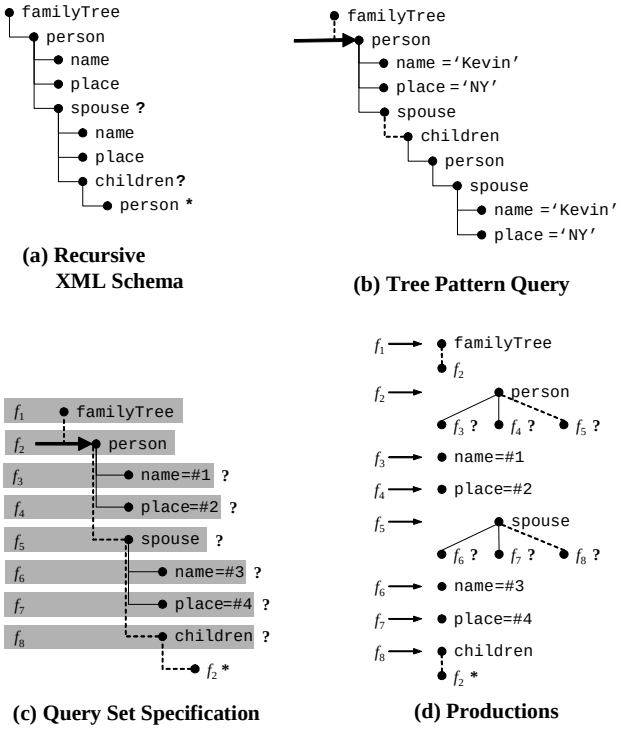

Figure 4. Family Tree Recursive Example.

2.2 Reasoning about Data Services

Aside from facilitating the development of applications that are clients of the data service, QSSs allow reasoning about data services. Below are examples of data service properties we would like to verify.

- *Membership of a query in a data service.* The most basic problem is to check if a client TP query *q* is accepted by a data service described by QSS *A*, i.e. $q \in \mathrm{TP}(A)$.
- *Subsumption of data services:* given services described by QSSs A_1 and A_2, check if $\mathrm{TP}(A_1) \subseteq \mathrm{TP}(A_2)$.
- *Totality of a data service:* does the data service described by QSS *A* accept all possible TP queries?
- *Overlap of data services*: given services described by QSSs A_1 and A_2, check if $\mathrm{TP}(A_1) \cap \mathrm{TP}(A_2) \neq \emptyset$.

Of course, revisiting the analogy between QSSs and extended context-free grammars, we could reduce these problems to decision problems on grammars. However, while the membership problem can be solved in this way, the other problems in the list reduce to well-known problems that are undecidable even for standard context-free grammars. Fortunately, it turns out that a QSS can be translated to an equivalent top-down nondeterministic unranked tree automaton [3] (the translation is straightforward and omitted due to space limitations). QSSs therefore describe regular tree languages, for which all problems listed above are decidable [3].[2]

A practically important question is whether a client query can be answered using a finite subset of the queries described by a QSS. This is related to the problem of answering queries using limited query capabilities [18].

2.3 WSDL and XML Syntax

Our proposal for specifying data services is compatible with the standard Web Service Specification Language WSDL [5] in the sense that any QSS can be translated into a WSDL specification.

[2] This observation should not come as a surprise given the similar result stating that DTDs, who look strikingly similar to extended context-free grammars, actually describe regular tree languages [17].

In general, a WSDL specification describes the format of the messages that a service sends or receives[3] using element declarations and type definitions drawn from the XML Schema type system [7]. A WSDL specification describing a data service restricts a general WSDL specification in several ways, since the communication between the agent and the data service is always synchronous and is carried out in a request/response fashion [16]. The input message represents the query received from the service, and the output message the result sent from the service. Both message types are described using the XML Schema type system.

A QSS can be automatically translated into a WSDL specification using the well-known fact that XML Schemas describe regular tree languages themselves. We omit the details of the translation algorithm, but illustrate on an example. This example makes a convincing case for presenting users with a concise and visually intuitive representation such as a QSS instead of the less readable XML syntax of the WSDL.

Example. Appendix A shows the WSDL specification of the QSS from Figure 2c. The schemas that describe the input and the output messages of the data service are imported in the beginning of the specification. The first schema describes the parameterized queries supported by the data service. A QSS is expressed in XML Schema format (QSSX) in order to be contained in a WSDL specification. QSSX is an XML Schema that acceptable TPX queries conform to. The QSSX syntax for the QSS in Figure 2c is shown in Appendix B. The second schema reveals the structure of the underlying database and presents a choice group consisting of the result node names in the result node names set of a QSS. Appendix C shows the XML Schema for the QSS in Figure 2c. As in the case of QSS and QSSX syntax, TP queries need to be expressible in XML format in order to be contained in messages described by a WSDL specification. The XML syntax of TP queries, called TPX, is a subset of XQueryX [12], the XML syntax of XQuery. The TPX query equivalent to the TP query in Figure 2b is given in Appendix D. The XML Schema that defines the TPX language is presented in Appendix E. ■

3. QSSL EXTENSIONS

In the future, QSSL will be enhanced to describe subsets of XQuery expressions beyond XPath ones, as well as additional constraints that restrict the co-occurrence of tree fragments.

[3] A WSDL specifies many additional communication details: synchronicity, how sets of messages are grouped into one operation, etc., all of which are orthogonal to our proposal.

In Figure 4c, for example, the QSS only indicates that a parameterized equality predicate on the name and on the birth place of a person can optionally be part of an acceptable PTP query. The QSS does not have the ability to succinctly express that these two predicates are mutually exclusive, or express that at least one of them must be part of an acceptable PTP query. It can achieve the desired effect by explicitly listing all acceptable combinations, but this defeats the purpose of QSSL.

In order to express these constraints, QSS can be enriched with a set of *replacement constrains* including *atLeast*, *atMost* and *xor*.

For example, *atLeast*(1,{f_3, f_4}) expresses that in a derivation at least one of f_3 and f_4 must be replaced. *xor*({f_3, f_4},{ f_6, f_7}) expresses that either the parameterized predicates on the name and on the place of a person or on the name and on the place of a spouse are part of an acceptable PTP query, but not both.

4. RELATED WORK

In the past, the database community has conducted research on the related problems of answering queries using views [9], capability-based query rewriting [8, 18] and computation of query capabilities [19]. One approach assumes that a source exports a relational view with *n* attributes, and query capabilities are described as *binding patterns* [9]. Each binding pattern attaches a *b* (bound) or an *f* (free) adornment on each attribute of the exported view. Adornment *b* means that a value for the attribute is required in a query, while *f* means that a value is optional. The set of adornments can be enriched adding *u*, where a value for an attribute is not permitted, *c*[*s*], where a value for an attribute is required and must be chosen from the set of constants *s*, and *o*[*s*], where a value in *s* is optional [19]. Note that each binding pattern defines a query template. Query capabilities described as binding patterns are characterized as *negative*, because they restrict the set of all possible queries against the exported view. Wrappers exporting binding patterns are called *thin*, because of their limited functionality to execute the input query against the underlying source.

Another approach describes sets of (parameterized) queries using the expansions of a Datalog program [11, 18]. In this work, it is shown that Datalog is not enough to cover even all yes/no conjunctive queries over a schema. It consequently showed that the RQDL extension can describe large sets, such as the set of all conjunctive queries over a schema. QSSL and data services also attempt to describe the capabilities of sources that support large sets of queries and aim to fuel the research on the problems considered in [8, 9,

18, 19] for the XML data model and the XQuery language [4].

On the industrial level, the effort is focused on turning relational database systems to web services providers by exporting data definition and manipulation operations via web services. These operations are either fixed or parameterized queries expressed in SQL or SQL/XML [6], stored procedures, or functions. Typically, a web service exporting a fixed query takes as input the name of the database operation, and possibly a parameter instantiation, and outputs either an XML document or a serialized object in a given programming language. No schema information of the underlying database is given, either for the input or the output. A list of systems implementing this architecture includes IBM's Document Access Definition Extension (DADx) for DB2 [10], Oracle's Database Web Services specification [13], Microsoft's SQL Server 2000 Web Services Toolkit [21] and BEA's WebLogic Workshop [20]. There is also an effort on consuming web services within the SQL query language, thus integrating relational data with web services.

The W3C Web Services Description Working Group [16] describes usage scenarios that focus on various types of communication using messages and demonstrate how they can be carried out using web services. The technical issues focus on the direction of communication, i.e., request-response, solicit-response or one-way, whether a web service is synchronous or asynchronous and whether it supports conversations, rather than what query capabilities a database exports.

Finally, previous work defines a preliminary and restricted version of QSSL that supports the generation of web-based query forms and reports for semistructured data [15]. Only finite sets of parameterized queries can be encoded no formal semantics is given, and there is not an algorithm that translates a QSS to an XML Schema.

5. REFERENCES

[1] J. Albert, D. Giammarresi, D. Wood**:** *Normal Form Algorithms for Extended Context-Free Grammars*, Theoretical Computer Science 267, pp. 35-47, 2001.

[2] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, D. Srivastava: *Minimization of tree pattern queries*, ACM SIGMOD, 2001.

[3] A. Brüggemann-Klein, M. Murata, and D. Wood: *Regular Tree and Regular Hedge Languages over Unranked Alphabets: Version 1, April 3, 2001*, Technical Report HKUST-TCSC-2001-0, The Hong Kong University of Science and Technology, 2001.

[4] D. Chamberlin et al.: *XQuery 1.0: An XML Query Language*, W3C Working Draft, 2002. http://www.w3.org/TR/xquery/

[5] R. Chinnici et al.: *Web Services Description Language (WSDL) v. 1.2*, W3C Working Draft, 2003. http://www.w3.org/TR/wsdl12/

[6] A. Eisenberg, J. Melton: *SQL/XML is Making Good Progress*, SIGMOD Record 31(2), 2002.

[7] D. C. Fallside: *XML Schema Part 0: Primer*, W3C Recommendation, 2001. http://www.w3.org/TR/xmlschema-0/

[8] L. M. Haas, D. Kossmann, E. L. Wimmers, J. Yang: *Optimizing Queries Across Diverse Data Sources*, VLDB, 1997.

[9] A. Y. Halevy: *Answering Queries Using Views: A Survey*, VLDB Journal 10(4): 270-294, 2001.

[10] G. Hutchison: *DB2 and Web Services: The Big Picture*, January 2003. http://www7b.software.ibm.com/dmdd/zones/webservices/

[11] A. Y. Levy, A. Rajaraman, J. D. Ullman: *Answering Queries Using Limited External Processors*, PODS, 1996.

[12] A. Malhotra et al.: *XML Syntax for XQuery 1.0 (XQueryX)*, W3C Working Draft 7 June 2001. http://www.w3.org/TR/xqueryx

[13] K. Mensah, E. Rohwedder: *Oracle9i Database Web Services*, White Paper, November 2002. http://otn.oracle.com/tech/webservices/htdocs/dbwebservices/Database_Web_Services.pdf

[14] G. Miklau, D. Suciu: *Containment and Equivalence for an XPath Fragment*, PODS, 2002.

[15] Y. Papakonstantinou, M. Petropoulos, V. Vassalos: *QURSED: Querying and Reporting Semistructured Data*, ACM SIGMOD, 2002.

[16] W. Sadiq et al.: *Web Service Description Usage Scenarios*, W3C Working Draft, 2002. http://www.w3.org/TR/ws-desc-usecases/

[17] L. Segoufin, V. Vianu: *Validating Streaming XML Documents*, PODS, 2002.

[18] V. Vassalos, Y. Papakonstantinou: *Describing and Using Query Capabilities of Heterogeneous Sources*, VLDB, 1997.

[19] R. Yerneni, C. Li, H. Garcia-Molina, J. Ullman: *Computing Capabilities of Mediators*, ACM SIGMOD, 1999.

[20] *BEA WebLogic Workshop* http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/workshop/

[21] *Microsoft SQL Server 2000 Web Services Toolkit* http://www.microsoft.com/sql/techinfo/xml/

APPENDIX

A. WSDL Specification of a Data Service

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="FlightsService"
    targetNamespace="http://airline.wsdl/flights/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://airline.wsdl/flights/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd1="http://airlineQSSX/"
    xmlns:xsd2="http://airlineSchema/">

    <import location="airlineQSSX.xsd" namespace="http://airlineQSSX/"/>
    <import location="airlineSchema.xsd" namespace="http://airlineSchema/"/>

    <message name="queryFlightsRequest">
        <part name="query" type="xsd1:query"/>
        <part name="result" type="xsd2:result"/>
    </message>
    <message name="resultFlightsResponse">
        <part name="result" type="xsd2:result"/>
    </message>

    <portType name="FlightsPortType">
        <operation name="queryFlights" variety="Input-Output">
            <input message="tns:queryFlightsRequest" name="queryFlightsRequest"/>
            <output message="tns:resultFlightsResponse" name="resultFlightsResponse"/>
        </operation>
    </portType>
</definitions>
```

B. QSSX Syntax

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://airlineQSSX/"
            xmlns:qssx = "http://airlineQSSX/">

  <xsd:annotation>
    <xsd:documentation>The root element of a TPX query</xsd:documentation>
  </xsd:annotation>
  <xsd:element name = "query">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:group ref = "qssx:f1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:annotation>
    <xsd:documentation>
      The following element groups correspond to the productions of Figure 4d
    </xsd:documentation>
  </xsd:annotation>

  <xsd:group name = "f1">
    <xsd:sequence>
      <xsd:element name = "step">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name = "identifier" fixed = "flights"/>
            <xsd:choice>
              <xsd:sequence>
                <xsd:annotation>
                  <xsd:documentation>
                    The airline element is chosen as the result node
                  </xsd:documentation>
                </xsd:annotation>
                <xsd:element name = "predicatedExpr">
                  <xsd:complexType>
                    <xsd:sequence>
                      <xsd:element name = "identifier" fixed = "airline"/>
                      <xsd:group ref = "qssx:f2"/>
                      <xsd:element name = "predicate" maxOccurs = "unbounded">
                        <xsd:complexType>
                          <xsd:sequence>
```

```
                        <xsd:group ref = "qssx:f3"/>
                      </xsd:sequence>
                    </xsd:complexType>
                  </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="axis" use="required" type="xsd:string" fixed = "CHILD"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>
                The flight element is chosen as the result node
              </xsd:documentation>
            </xsd:annotation>
            <xsd:element name = "step">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name = "predicatedExpr">
                    <xsd:complexType>
                      <xsd:sequence>
                        <xsd:element name = "identifier" fixed = "airline"/>
                        <xsd:group ref = "qssx:f2"/>
                      </xsd:sequence>
                    </xsd:complexType>
                  </xsd:element>
                  <xsd:group ref = "qssx:f3" maxOccurs = "unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="axis" use="required" type="xsd:string" fixed = "CHILD"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name = "axis" use = "required" type = "xsd:string" fixed = "CHILD"/>
    </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:group>

<xsd:group name = "f2">
 <xsd:sequence>
  <xsd:element name = "predicate">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name = "function">
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element name = "identifier" fixed = "name"/>
        <xsd:element name = "constant" type = "xsd:string"/>
       </xsd:sequence>
       <xsd:attribute name="name" fixed="EQUAL"/>
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:group>

<xsd:group name = "f3">
 <xsd:sequence>
  <xsd:element name = "predicatedExpr">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name = "identifier" fixed = "flight"/>
     <xsd:group ref = "qssx:f4" minOccurs = "0"/>
     <xsd:group ref = "qssx:f5"/>
     <xsd:group ref = "qssx:f6" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:group>

<xsd:group name = "f4">
```

```
    <xsd:annotation>
      <xsd:documentation>
        Similar to f2 element group
      </xsd:documentation>
    </xsd:annotation>
    ...
  </xsd:group>

  <xsd:group name = "f5">
    <xsd:annotation>
      <xsd:documentation>
        Similar to f2 element group
      </xsd:documentation>
    </xsd:annotation>
    ...
  </xsd:group>

  <xsd:annotation>
    <xsd:documentation>
      A choice appears in group f6, because leg element might appear as an identifier
      if groups f7, f8 and f9 are not replaced, or as a pedicated expression otherwise
    </xsd:documentation>
  </xsd:annotation>
  <xsd:group name = "f6">
    <xsd:sequence>
      <xsd:element name = "predicate">
        <xsd:complexType>
          <xsd:choice>
            <xsd:element name = "identifier" fixed = "leg"/>
            <xsd:sequence>
              <xsd:element name = "predicatedExpr">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name = "identifier" fixed = "leg"/>
                    <xsd:group ref = "qssx:f7" minOccurs = "0"/>
                    <xsd:group ref = "qssx:f8" minOccurs = "0"/>
                    <xsd:group ref = "qssx:f9" minOccurs = "0"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:group>

  <xsd:group name = "f7">
    <xsd:annotation>
      <xsd:documentation>
        Similar to f2 element group
      </xsd:documentation>
    </xsd:annotation>
    ...
  </xsd:group>

  <xsd:group name = "f8">
    <xsd:annotation>
      <xsd:documentation>
        Similar to f2 element group
      </xsd:documentation>
    </xsd:annotation>
    ...
  </xsd:group>

  <xsd:group name = "f9">
    <xsd:annotation>
      <xsd:documentation>
        Similar to f2 element group
      </xsd:documentation>
    </xsd:annotation>
    ...
  </xsd:group>
</xsd:schema>
```

C. Result XML Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://airlineSchema/">
  <xsd:element name = "result">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element ref = "airline"/>
        <xsd:element ref = "flight"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name = "airline">
    <xsd:annotation>
      <xsd:documentation>
        Element declaration as it appears in the data XML Schema
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>

  <xsd:element name = "flight">...</xsd:element>

  <xsd:element name = "leg">...</xsd:element>
</xsd:schema>
```

D. TPX Query

```
<query xmlns = "http://www.db.ucsd.edu/tpx">
  <step axis = "CHILD">
    <identifier>flights</identifier>
    <step axis = "CHILD">
      <predicatedExpr>
        <identifier>airline</identifier>
        <predicate>
          <function name = "EQUALS">
            <identifier>name</identifier>
            <constant datatype = "CHARSTRING">Delta</constant>
          </function>
        </predicate>
      </predicatedExpr>
      <predicatedExpr>
        <identifier>flight</identifier>
        <predicate>
          <function name = "EQUALS">
            <identifier>from</identifier>
            <constant datatype = "CHARSTRING">JFK</constant>
          </function>
        </predicate>
        <predicate>
          <function name = "EQUALS">
            <identifier>to</identifier>
            <constant datatype = "CHARSTRING">LAX</constant>
          </function>
        </predicate>
        <predicate>
          <function name = "EQUALS">
            <identifier>day</identifier>
            <constant datatype = "CHARSTRING">MON</constant>
          </function>
        </predicate>
        <predicate>
          <predicatedExpr>
            <identifier>leg</identifier>
            <predicate>
              <function name = "EQUALS">
                <identifier>to</identifier>
                <constant datatype = "CHARSTRING">LAS</constant>
              </function>
            </predicate>
          </predicatedExpr>
        </predicate>
      </predicatedExpr>
    </step>
  </step>
</query>
```

E. XML Schema for TPX Syntax

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
            targetNamespace = "http://www.db.ucsd.edu/tpx">
    <xsd:group name = "expression">
        <xsd:choice>
            <xsd:element ref = "constant"/>
            <xsd:element ref = "function"/>
            <xsd:element ref = "predicatedExpr"/>
            <xsd:element ref = "step"/>
            <xsd:element ref = "identifier"/>
        </xsd:choice>
    </xsd:group>
    <xsd:element name = "query">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:group ref = "expression"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "predicatedExpr">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:group ref = "expression"/>
                <xsd:element ref = "predicate" maxOccurs = "unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "predicate">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:group ref = "expression"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "identifier" type = "xsd:string"/>
    <xsd:element name = "constant">
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:extension base = "xsd:string">
                    <xsd:attribute name = "datatype" type = "xsd:string"/>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "function">
        <xsd:complexType>
            <xsd:choice minOccurs = "0" maxOccurs = "unbounded">
                <xsd:group ref = "expression"/>
            </xsd:choice>
            <xsd:attribute name = "name" use = "required" type = "xsd:string" fixed = "EQUAL"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "step">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:group ref = "expression"/>
                <xsd:group ref = "expression"/>
            </xsd:sequence>
            <xsd:attribute name = "axis" use = "required">
                <xsd:simpleType>
                    <xsd:restriction base = "xsd:NMTOKEN">
                        <xsd:enumeration value = "CHILD"/>
                        <xsd:enumeration value = "DESCENDANT"/>
                        <xsd:enumeration value = "SLASHSLASH"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```