VLDBJ Manuscript Nr.
(will be inserted by hand later)

Locating and Accessing Data Repositories with WebSemantics*

George A. Mihaila[1], **Louiqa Raschid**[2], **and Anthony Tomasic**[3]**

[1] Department of Computer Science, University of Toronto, 10 King's College Road, Toronto, ON, M5S 1A4, Canada
[2] Robert H. Smith School of Business and UMIACS, University of Maryland, College Park, MD, USA, MD20742
[3] INRIA Rocquencourt, 78153 Le Chesnay, France

Received: July 1999 / Accepted: September, 2000

Abstract. Many collections of scientific data in particular disciplines are available today on the World Wide Web. Most of these data sources are compliant with some standard for interoperable access. In addition, sources may support a common semantics, i.e., a shared meaning for the data types and their domains. However, sharing data among a global community of users is still difficult because of the following reasons: (i) data providers need a mechanism for describing and publishing available sources of data; (ii) data administrators need a mechanism for discovering the location of published sources and obtaining metadata from these sources; and (iii) users need a mechanism for browsing and selecting sources. This paper describes a system, WebSemantics, that accomplishes the above tasks. We describe an architecture for the publication and discovery of scientific data sources, that is an extension of the World Wide Web architecture and protocols. We support catalogs containing metadata about data sources for some application domain. We define a language for discovering sources and querying their metadata. We then describe the WebSemantics prototype.

1 Introduction

Recently, many standardized collections of scientific data, in specific disciplines, have become available on the World Wide Web. For example, many collections of environmental data, located around the world, are now available to scientists [Fra97, GHC, GPC, WDC, NGD]. These sources often comply with a standard for interoperability, e.g. the data is in a relational DBMS. The data may also conform to a common semantics, i.e. each item of data is precisely defined. However, the sharing of information between scientists is still a very difficult process. Sharing is hindered by the lack of mechanisms for *describing and publishing* data sources, and for *discovering* the existence of data relevant to a problem.

The World-Wide Web (WWW) is a system for sharing *documents*. In this system, documents are published and accessed via the HTTP protocol and HTML document format standards. Our goal is to make the access to structured (typed) data sources as easy as access to documents, that is, to give "equal-time" to data on the Internet. Recently XML and XMLSchema [XML96, XS00] have been proposed and widely adopted as a data exchange format and a model for exchanging structured data on the WWW. Resource description languages and mechanisms for resource discovery have also been proposed [MCF97, RDF99, Lyn91, Z39, WID97]. Much of this work has centered on bibliographic collections and resource discovery and information sharing among such specialized collections. There has been some work on extensions to other domains, for example geo-referenced or geo-spatial data [Z39]. However, there is still little support for publishing and describing data sources for structured data on the WWW, and for resource discovery using the metadata of these sources.

In this paper we describe the implementation of a prototype system, WebSemantics (WS), which permits publication and discovery of sources containing typed data, using the WWW and XML. Our approach extends the WWW with a specification for publishing the location, and optionally the metadata (types and domains) of sources, in WS-XML documents. The WS-XML specification is an instance of the XML [XML96] metalanguage. WS then provides a language for discovering relevant published data sources. The language combines features for searching relevant WS-XML documents that publish data sources with features for searching over the metadata describing these sources. This approach smoothly integrates the functionality already existing on the WWW for searching documents with the WS extensions for searching over the metadata.

The contributions of this paper are as follows:

- An architecture which permits publication and discovery of data sources containing structured data.

* This research was partially sponsored by the National Science Foundation grant IRI9630102 and the Defense Advanced Research Projects Agency grant 01-5-28838
** Present affiliation: Common Object Inc., San Mateo, CA
Correspondence to: George A. Mihaila

– A catalog which stores metadata about data sources in a specific application domain.
– A query language that supports the discovery of data sources.
– A description of a prototype implementation of the WS architecture, catalog and query language.

The paper is organized as follows: Section 2 is an overview of the WS architecture. Section 3 describes the query language for the discovery of data sources. Section 4 describes the WS architecture and query processing in more detail. Section 5 describes the WS prototype. Section 6 describes related work. In particular, a comparison is made with resource description and resource discovery for bibliographic collections and the extensions that have been made to accommodate structured data in particular domains. Section 7 concludes the paper. Preliminary results from this paper were presented in [MRT98].

2 Overview of the WebSemantics System

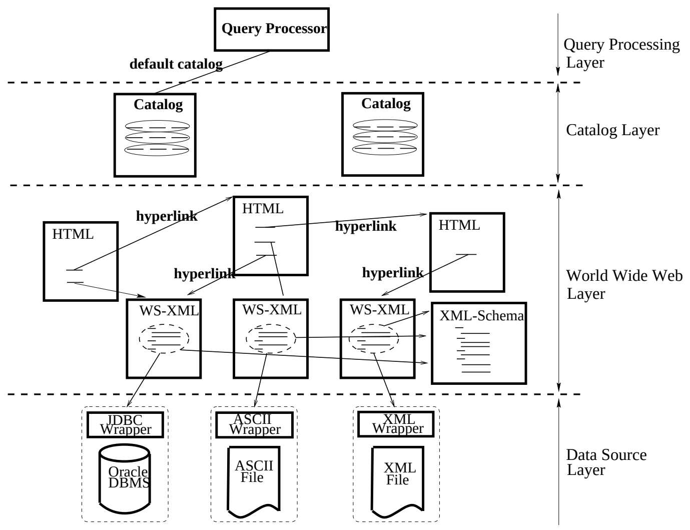


Fig. 1. The WebSemantics layers

In this section we briefly overview the WebSemantics (WS) system. The WS system has a layered architecture of interdependent components (see Figure 1).

The *Data Source Layer* has two components, *data sources* and *wrappers*. Data providers create and manage collections of autonomous *data sources*, that can be accessed over the Internet. These data sources can provide query capability ranging from full DBMS functionality to simple scanning of files. WS assumes uniform access to both kinds of sources, independent of the capability of the sources. This is accomplished using *wrapper* components.

The second layer is the *World Wide Web Layer*. The WWW is used as a medium for describing and publishing sources. Thus, in order to publish a data source, a provider needs to create a WS-XML document describing the source.

```
<xs:schema xmlns:xs="http://www.w3.org/1999/XMLSchema"
           version="1.0">

  <xs:element name="AirQuality">
    <xs:complexType content="empty">
      <xs:attribute name="date" type="timeInstant"/>
      <xs:attribute name="location" type="string"/>
      <xs:attribute name="CO2percentage" type="float"/>
      ...
    </xs:complexType>
  </xs:element>

  <xs:element name="Rainfall">
    ...
  </xs:element>

</xs:schema>
```

Fig. 2. The `EnvSchema.xml` file: a shared schema for environmental data

The third layer is the *Catalog Layer*. A *catalog* is a specialized repository storing metadata about a collection of data sources in some application domain. For each source, the catalog maintains metadata characterizing the source, such as the set of types exported by the source; the domains for a subset of attributes of some types; a textual description of the data source; the URL of the WS-XML document that published the source, etc.

Finally, the WS *query processor* component, bound to a specific catalog, allows the user to discover and select sources based on the available metadata.

2.1 Motivating Example

Suppose a community of environmental scientists agreed on a common schema, `EnvSchema`, and an associated semantics for various measurements. The schema consists of a set of *types*, e.g., relational table names and attributes. For example, a type AirQuality(date, location, CO2percentage, ...) describes the date and location of the measurement, and the percentages of various gases from a standard set of gases of interest (carbon dioxide, carbon monoxide, etc.). While we use the relational data model to simplify our example, WS does not insist on the use of the relational model. In order to make the schema available to everyone, they publish it on the Web, in the XML file shown in Figure 2. This file conforms to the XML-Schema conventions for describing strongly typed relational data [XS00].

Consider a scientist who measures air quality parameters in Ontario. She measures the concentration of greenhouse gases in the atmosphere and stores the results of her daily measurements in a `DB2` database. The schema of this database follows the shared semantics, `EnvSchema`. In order to make this data source available to WebSemantics users, she publishes a WS-XML document (shown in Figure 3) that specifies the connection

```
<?xml version="1.0"?>
<!DOCTYPE ws SYSTEM "wsxml.dtd">
<ws>
 <source>
  <sci>
     <wrapper wtype="JDBC"/>
     <repository rtype="DB2"
       rlocation="jdbc:db2://server.env.org/ontario"/>
  </sci>
  <metadata>
     <schema>http://www.env.org/EnvSchema.xml</schema>

     <type name = "AirQuality">
         <domain attr="location"
                 domtype="enumeration"
                 values="Halifax Montreal Toronto"/>
         <domain attr="time" domtype="range"
                 minvalue="Jan 1, 1990"
                 maxvalue="Dec 31, 2000"/>
     </type>

     <type name = "Rainfall">
          ...
     </type>

     ...
  </metadata>
  <desc> This repository contains daily measurements
        of air quality parameters in Ontario for
        the year 2000. </desc>
 </source>
</ws>
```

Fig. 3. Describing a data source in an WS-XML document

information for the source (such as the type of wrapper and the location of the data) as well as the subset of types from the shared schema that are available in this source. The complete DTD describing a WS-XML document is given in Appendix A.

Suppose that a second scientist is interested in air quality data for his research. In order to locate data sources of interest, he can execute the following query:

Query 1. *Find all the sources described in WS documents that mention the phrase "air quality".*

select *s*
from *Document d* **such that** *d* **mentions** *"air quality"*,
Source s **such that** *d* **describes** *s;*

This query finds any document that mentions air quality and describes sources. The **mentions** keyword instructs the query processor to submit a query to Web search engines. We will discuss how we can limit the search to WS-XML documents in a later section. Matching documents are processed to extract the information about data sources that are described in that document (the **describes** keyword specifies the relationship between WS-XML documents and the data sources for which they contain metadata). Later, a scientist can also **register** the discovered data sources with a WS catalog. Registering a data source means that its location and metadata are made available to the catalog. Once a WS catalog has been constructed, other scientists can query the catalog in order to identify data sources.

As a sample application domain for WS, we have chosen environmental information, and a community of environmental scientists. However, the underlying architecture and language are not specific to environmental data or scientific data in general. Given a shared schema for some data, WS can be utilized to share this data. For example, given a simple shared schema of the "cars for sale" ads from newspapers, (e.g., make, model, year, and price of a car for sale), a specialized catalog can publish sources of such ads and help users search for such sources.

3 Locating and Querying Data Sources

In this section, we introduce the WebSemantics Query Language (WSQL) which provides the following functions: 1) finding data sources that are published in WS-XML documents or that are registered in existing catalogs; and 2) selecting sources based on their metadata. To accomplish these tasks, the WSQL language integrates constructs borrowed from WebSQL [MMM97] and OQL [C$^+$96]. We present the features of the language through examples.

3.1 Finding Sources on the WWW

Query 1 presented in the previous section illustrated one way of using search engines to locate data sources on the WWW, assuming no previous knowledge about their location. However, if a user has additional information, for example the home page of a particular research institute, she can use this information to restrict the scope of the search to the pages reachable from that home page. The following query would build the desired collection of sources:

Query 2. *Find all the sources described in WS-XML documents reachable from "www.env.org/" that contain the phrase "air quality" in their text.*

select *s*
from *Document d* **such that** *"www.env.org/"* $\rightarrow^*$ *d,*
Source s **such that** *d* **describes** *s*
where *d.text* **contains** *"air quality";*

The first construct in the **from** clause sets the range of the variable *d* to the set of all documents on the "www.env.org" server which are reachable from the root page. The path regular expression '$\rightarrow^*$' means "traverse any number of local links starting from the specified URL". The set of documents that are reachable are further restricted by a predicate in the **where** clause which specifies a string containment condition on *d.text*. The

contains condition is directly evaluated on the reachable documents in the set. This is in contrast to the previous approach of querying search engines, as is done for the **mentions** clause. The second construct in the **from** clause sets the range of the variable s to the set of sources described in the documents which satisfy the predicate.

Path regular expressions, a construct borrowed from WebSQL, are regular expressions over the alphabet of link types: $\rightarrow$ is a link between documents on the same Web server and $\Rightarrow$ is a link to a different Web server.

3.2 Selecting Sources From Catalogs

The previous examples selected sources that were described in WS-XML documents. An alternative is to pick sources directly from specialized WS catalogs.

For each source, the catalog maintains a metadata tuple of the form *Source*(*id*, *description*, *types*, *domains*), where *types* is the set of types exported by the source. A source can export a subset of types identified in the shared schema. The *domains* is a set of active domains of some attributes of some types. Types are represented in the catalog as metadata tuples of the form *Type*(*name*, *attributes*), where *attributes* is a set of (*attr_name*, *datatype*) pairs. The active domain of an attribute is the set of distinct values of that attribute in all the tuples currently present in the source. Active domains are represented as metadata tuples of the form *Domain*(*type*, *attribute*, *values*).

For instance, going back to our example with environmental data, suppose one knows the addresses of several catalogs registering relevant data sources. One can select sources of interest from these catalogs with the following query:

Query 3. *Find all data sources containing air quality information for Toronto, identified from a specific list of catalogs.*

select *s*
from *Catalog c* **in** { *"rmi://alpha.env.org/WSCatalog"*,
"rmi://rep.env.ca/Catalog"},
Source s **in** *c.sources*,
Domain d **in** *s.domains*,
where *d.type* = *"AirQuality"* **and** *d.att* = *"location"*
and *d.values* **contains** *"Toronto"*;

The first clause in the **from** section identifies the catalogs to query. WS catalogs are implemented as Java classes accessible through the Remote Method Invocation protocol [RMI], hence they are identified through URLs of the form "rmi://host/service". The second clause binds the variable s to the set of all sources in the catalogs. The set of sources is further restricted by keeping only those sources that are associated with the type "AirQuality", such that the active domain of the attribute "location" contains the string "Toronto", among the the list of values for this domain. For each source s, *s.domains* are the active domains of attributes that are published by the source.

In the previous queries, we assumed that the type information was known. WS also allows queries over all the metadata, including the type names and attributes. Suppose, for example, that we want to find a type which contains an attribute whose name we know approximately. The following query illustrates this situation:

Query 4. *Find all types that have an attribute whose name contains the string "UV".*

select *t*
from *Catalog c* = *"rmi://alpha.env.org/WSCatalog"*,
Type t **in** *c.types*
where *t.attributes* **contains** *"UV"*;

Technically, the expression *t.attributes* evaluates to a definition of the attributes. The **contains** operator works on strings, so the expression *t.attributes* is automatically converted to a string before evaluating the substring condition.

4 WebSemantics Architecture and Interactions between Components

Section 2 presented an overview of the various components of the WS architecture. In this section, we describe the interactions between these components to support the tasks of discovering sources and registering them in a catalog. We then address the scalability of the WS architecture. This includes the issue of maintaining the catalog to be consistent with sources over time. We also discuss how WS uses wrappers based on standard interoperability solutions [SM, XML96], to access a variety of data sources with different functionality.

4.1 Registering Sources in the Catalog

Sources can be registered in the catalog in two ways. First, a data provider may explicitly register a source by submitting the URL for a WS-XML document that describes the source. The second method reflects resource discovery. The query processor evaluates a query *WSQL_query*, in the WSQL language described in Section 3, and identifies a set of sources which are then registered in the catalog. Registering is accomplished through a **register** command with the following syntax:

register into catalog *catalog_address*
(**sources** | **types** | **domains**) *URL*
| (**sources** | **types** | **domains**) *WSQL_query*

Thus, the register command allows types and domains as well as sources to be registered.

4.2 Catalog Maintenance

There are two issues that must be considered regarding the scalability of the catalog in the WS architecture. The first issue is that in a dynamic environment such as the

WWW, the catalog must be maintained to be consistent with the data sources. Over time, the data provided by the sources may change, and some sources may move or disappear. In order to reflect these changes in the catalog, WS supports deletions and updates of sources and their domains.

A **delete** command has a syntax similar to the **register** command. Executing this command unregisters the specified objects, which may include sources, and domains.

The **update** command instructs the catalog to recompute the specified objects. If the original WS-XML document that advertised the source (whose URL is maintained in the catalog) is current, and if the source is current, then the **update** command will be successful. If this WS-XML document is unavailable, then the objects will be unregistered.

The **delete** and **update** commands may be explicitly submitted to the catalog by a data provider. Recall that we also used a method of resource discovery to evaluate a *WSQL_query* query and to register sources in the WS catalog. Thus, we cannot rely on data providers to explicitly provide notification of updates, as these discovered data sources change.

Thus, to remain consistent, the WS catalog must periodically poll *all* the registered sources. For this, the WS-XML documents that publish the sources and that are referenced in the catalog will be periodically consulted via the **update** command. If the documents exist, then the catalog will be refreshed, i.e., sources will be updated or unregistered as appropriate. If the documents do not exist, then the object is unregistered.

The second issue is harnessing the technology that searches the Web, a la search engines, to gather URLs of all the WS-XML documents that publish sources, and construct indexes over these pages. These indexes may store the keywords from the natural language descriptions of the sources. They may also store the metadata extracted from WS-XML documents. The indexes would then be used to register sources into the catalog. This would be a natural extension of the WSQL language, with the capability of search engines, to gather and index the contents of WS-XML documents that describe sources. When we discuss our implementation, we will consider limiting these indexes to WS-XML documents.

4.3 Contacting Sources to Obtain Data

Throughout this paper we reference to strict types á la the relational data model. Since the main focus of WS is source discovery and selection, the system is oblivious to the particular data model (e.g. relational, object-relational, semistructured, etc.) used by the sources. However, if uniform data access is sought, one must implement appropriate wrappers for every such data model. For completeness, we briefly discuss different data sources that are not relational DBMS.

Data sources may vary widely based on their query capability, which could range from supporting complete DBMS query capability, to flat files whose contents may be scanned. Sources vary based on the format used to encode their answers. Also, sources may or may not support strict typing. Wrappers provide an application programming interface to data sources, to handle queries submitted to the data sources, and the answers. A uniform query interface is provided by sources that support generic interoperability standards, e.g., a relational DBMS. However the query interface is not standard across all sources. We now describe some combinations of WS wrappers/sources.

- *Generic* JDBC [SM] drivers provide a standard interface to relational databases. For example, an Oracle JDBC driver [Ora] can be used to access any Oracle data source. WS includes a WS↔JDBC wrapper that interoperates with any JDBC driver. The query capability of the driver/source and the answer format is specified by JDBC. This interoperability solution supports relational queries on the types in the source. A list of all available JDBC drivers is maintained at http://java.sun.com/products/jdbc/jdbc.drivers.html.
- A WS↔XML wrapper interoperates with any data source whose output is encoded as XML. In this case, we assume that the source cannot be queried and the wrapper only supports a scan over the XML output. The WS query engine must provide all additional query capabilities. This includes support for querying semi-structured data [AQM$^+$97, GMW99].
- Many sources accessible over the Web, WebSources [GRVB98], have a limited query capability, compared to a DBMS. Projects such as DISCO [TRV96], Garlic [CHS$^+$95], Information Manifold [KLSS95] and TSIMMIS [GM$^+$95] have developed wrappers for such sources, and provide mediator capability to submit appropriate queries to these wrappers. WS would rely on the mediators developed in these projects to access limited capability sources.
- Many collections of data are encoded as delimiter separated ASCII text in files, with particular native formats [GPC, WDC, GHC]. Custom WS wrappers need to understand the particular format of each source. The query capability of these wrappers is a scan of the specific type in the source.

The WS-XML document describing a data source must provide sufficient information that would enable WS to recognize the query capability, format, etc. of the source, so that the WS query processor can use an appropriate WS wrapper that can interoperate with the source. An XML DTD of the WS-XML document specification is included in Appendix A and describes how this information is provided, in order to publish the source.

We do not address the issue of data access to the variety of sources described above, and we refer the reader to related work on wrapper mediator architectures for data integration [AQM$^+$97, GMW99, FMLS99, Vid00].

4.4 Obtaining Metadata from Sources

A WS-XML document specifies the types in a data source. We assume that a community of users share a common semantics of types and domains for some application domain. Once a source is discovered, its types (specified in the shared schema) will be registered in the catalog. Domain information, corresponding to a particular attribute, may also be provided in the WS-XML document describing the types in the source. Available domain information will be stored in the catalog. Alternately, a query (scan) could be evaluated on the source using the appropriate WS wrapper to obtain domain information, assuming that the source will support such queries. As discussed previously, this would depend on the query capability of the source. Both type and domain information in the catalog must be periodically refreshed.

4.5 Query Processing

The WSQL language is a declarative query language. In order to evaluate a query, the query processor needs to compile it into some sort of evaluable representation. In this section we introduce an algebra and then we briefly describe the algorithm used by the query processor to generate an operator tree starting from a calculus query.

Consider a set **att** of *attributes* and the set **dom** of all metadata tuples. Given a subset $U = \{A_1, \cdots, A_n\} \subset$ **att**, a *tuple over* U is a mapping $t : U \longrightarrow \mathbf{dom}$. We usually write a tuple t in the form $\langle A_1 : a_1, A_2 : a_2, \cdots, A_n : a_n\rangle$ and denote by $t.A_i$ the value $t(A_i)$.

A *relation of sort* U is a finite set R of tuples over U. We denote by $Rel(U)$ the set of all relations of sort U. Finally, we denote by Rel the set of all relations and we assume a mapping $sort : Rel \longrightarrow 2^{\mathbf{att}}$ stating the sort of each relation. We introduce the *tuple product* operation: if $u = \langle A_1 : u_1, \cdots, A_n : u_n\rangle$ and $v = \langle V_1 : v_1, \cdots, V_n : v_n\rangle$ are tuples over two disjunct sets U and V, then the tuple $u \times v : U \cup V \longrightarrow \mathbf{dom}$ is defined by $u \times v = \langle U_1 : u_1, \cdots, U_n : u_n, V_1 : v_1, \cdots, V_n : v_n\rangle$.

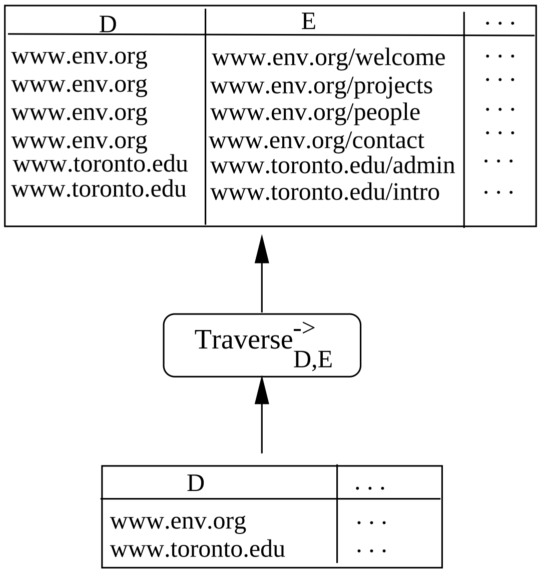
We now define operators for the various constructs in the WSQL language.

- The **select** and **where** clauses of WSQL queries are translated to operators from traditional algebra, i.e., (*Project* and *Select*), respectively.
- The **from** clause is non-traditional, compared to an SQL query. Variables are introduced in this clause and their ranges are defined by means of *range expressions*.
  The range expressions are classified as (1) regular expressions for document navigation; (2) keyword matching; (3) source descriptions occurring in WS-XML documents; and (4) field selection (types, attributes, domains, etc.).
  We introduce an operator for each type of range expression.

Each of these operators, with the exception of the last one, accepts as input a relation R and produce as output a relation R' that has exactly one extra attribute whose values depend on one or two attributes in the input relation (see Fig. 4). The naming convention for each operator is $op^{param}_{iAtt;oAtt}$ where $iAtt$ is the *input attribute*, $oAtt$ is the *output attribute* and *param* is a parameter. An operator $op^{param}_{A;B}$ reads each tuple t of its input relation and produces one or more output tuples t' that have the same attributes as t plus an extra attribute named B whose value depends on $t.A$ — the value of the attribute A in the input tuple t.

Definition 1. *We introduce the following four* range operators. *They correspond to the 4 types of range expressions previously described.*

- $Traverse^{r}_{D;E} : Rel \longrightarrow Rel$, *defined by*
  $Traverse^{r}_{D;E}(R) = \{t \times \langle E : e\rangle | t \in R$ *and there is a path matching the regular expression* r *from the document* $t.D$ *to the document* e $\}$
  This operator corresponds to regular expressions for document navigation.
- $Search_{W;D} : Rel \longrightarrow Rel$, *defined by*
  $Search_{W;D}(R) = \{t \times \langle D : d\rangle | t \in R\}$ *and document* d *mentions keyword* $t.W$.
  This operator corresponds to keyword matching;
- $Extract_{D;S} : Rel \longrightarrow Rel$, *defined by*
  $Extract_{D;S}(R) = \{t \times \langle S : s\rangle | t \in R\}$ *and document* $t.D$ *describes source* s.
  This operator corresponds to extracting source metadata from WS-XML documents;
- $GetField^{field}_{X;Y} : Rel \longrightarrow Rel$, *defined by*
  $GetField^{field}_{X;Y}(R) = \{t \times \langle Y : y\rangle | t \in R$ *and* $y \in t.X.field\}$.
  This operator corresponds to field selection (types, attributes, domains, etc.);
- $ConstantSet^{A}_{X} \in Rel$, *defined by*
  $ConstantSet^{A}_{X} = \{\langle X : x\rangle | x \in A\}$, *where* $A \subset \mathbf{dom}$ *is a set of domain constants.*



D	E	...
www.env.org	www.env.org/welcome	...
www.env.org	www.env.org/projects	...
www.env.org	www.env.org/people	...
www.env.org	www.env.org/contact	...
www.toronto.edu	www.toronto.edu/admin	...
www.toronto.edu	www.toronto.edu/intro	...

D	...
www.env.org	...
www.toronto.edu	...

Fig. 4. An example range operator: $\text{Traverse}^{\rightarrow}_{D;E}$ produces for each input tuple a set of output tuples containing pairs of URLs of WWW documents connected by a local link.

The ConstantSet$_X^A$ operator takes no input and produces a relation with just one attribute X having as values all the domain constants in the set A. In addition to range operators we use the traditional relational operators Select and Project with their usual semantics.

From Def. 1 we note that a range operator $op_{iAtt,oAtt}^{param}$ is only defined on relations R such that $sort(R) \supseteq iAtt$. This leads to the following definition of a well-formed operator tree.

Definition 2. *An operator tree is a rooted tree $T = (V, E, \lambda)$, where λ is a labeling function mapping each node to an operator such that for each node v, $\lambda(v)$ is an operator whose arity coincides with the number of children of v.*

We define recursively the functions $input, output : V \rightarrow 2^{\mathbf{att}}$ giving for each node the sorts of its input and output:

- *if v is a leaf node, then $\lambda(v) = ConstantSet_X$; we define $input(v) = \emptyset$ and $output(v) = \{X\}$;*
- *for every interior node, define $input(v) = \bigcup_{(v,w)\in E} output(w)$;*
- *if v has exactly one child w, then $\lambda(v)$ is an operator $op_{iAtt,oAtt}$; we define $output(v) = output(w) \cup \{oAtt\}$.*
- *if v has two or more children, then $\lambda(v) = \times$; we define $output(v) = input(v)$;*

We call the tree well-formed *if for each interior node v with a label of the form $op_{iAtt,oAtt}$, we have $iAtt \subseteq input(v)$.*

Given a WSQL query Q, our goal is to produce a well-formed operator tree such that the output of its root operator is the answer to the query Q. In the remainder of this section, we describe our tree generating algorithm on an example.

Consider the following WSQL query that identifies all the data sources described in documents on a specific server:

select *s.id*, *s.types*
from Document *d* **such that** "www.env.org/" $\rightarrow^*$ *d*,
Source *s* **such that** *d* **describes** *s*;

The algorithm builds the tree in a bottom-up fashion.

1. The first step is generating a ConstantSet operator for each constant in the query.
2. Next, for each range expression E in the **from** clause, generate the corresponding operator $op(E)_{iAtt}^{oAtt}$, where *iAtt* and *oAtt* are attributes corresponding to the variables and constants in E.
   The mapping between the particular type of range operator and the corresponding range expression has been described.
   This operator is then connected to the top-most operator *op* for which $output(op) \supseteq iAtt$.
3. In the final step, a Project operator is added at the root of the tree.

The resulting tree is shown in Figure 5.

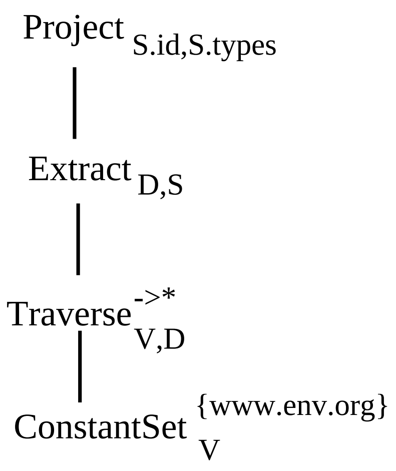


Fig. 5. An operator tree

5 Implementation

In this section, we describe the status of our prototype implementation of the WS system. The WS prototype has three main components: a Query Processor, a Catalog Server, and Wrappers. All components are implemented as a collection of Java classes, organized in a WS Class Library, which facilitates their integration in Java application programs. We also implemented two Java applets facilitating the user interaction with the system, for ad-hoc WSQL queries, and catalog visualization and administration.

5.1 The Query Processor

The query run-time system uses the Graefe iterator model [Gra93]). where each operator is implemented by a physical algorithm called an *iterator*. Each iterator implements three methods: *open*, which prepares the operator for producing data; *next*, which produces an output element; and *close*, which performs final housekeeping. Once an operator tree is built, the evaluation is initiated by invoking the *open* method on the root iterator, then repeatedly invoking *next* until there are no more items, and finally invoking *close*, all on the root iterator. The root iterator will call the appropriate methods on its input iterator(s), and so on, down to the leaf iterators, which execute the methods directly. Each time an iterator needs an input element, it calls its input iterator(s) to produce one.

It is worth noting here that the iterators that correspond to range operators produce a list of output elements for each input element and they store this list in an internal buffer from which they deliver one item at a time on each invocation of the *next* method.

For example, the Traverse^r iterator reads a URL from the input tuple, then starts exploring its Web neighborhood using a labeled graph traversal algorithm (adapted from [MW95]) in order to find all the paths matching the path regular expression r. For each such path, the iterator produces an output tuple containing the URL of its endpoint.

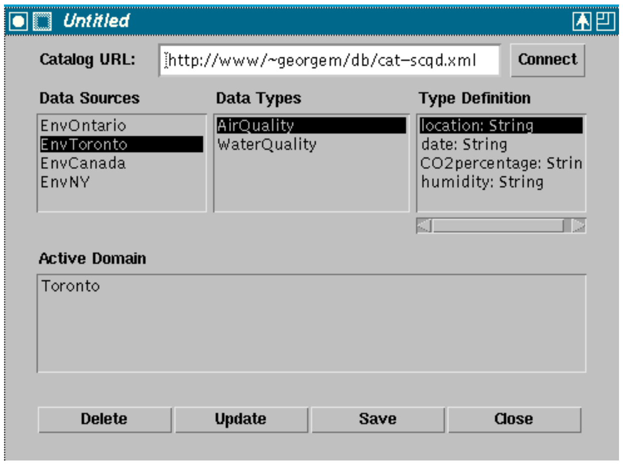
The Search iterator reads a text pattern from the input tuple, then sends this pattern to an index server (currently AltaVista) and produces an output tuple for each answer retrieved from the server. In order to reduce the search space to WS-XML documents, a special keyword ("scitnamesbew" which is "websemantics" spelled backwards) is included in all WS-XML documents. This keyword is also added to the list of keywords requested in the query before submitting the query to a search engine. In our experience, this keyword does not occur in any other document in any language and has allowed us to narrow the search to relevant documents.

The Extract iterator reads a URL from its input tuple, retrieves the corresponding WS-XML document, and extracts all the source descriptions from it, producing an output tuple for each description.

5.2 Catalog Server

In the current prototype, we have implemented a centralized catalog to store the metadata for a small collection of data sources. The catalog currently supports catalog construction, querying and simple maintenance tasks (deleting and updating individual sources).

During catalog construction, when a new source is registered, the catalog downloads the corresponding WS-XML document and extracts types and domain metadata exported by each source. In some cases, domain information can also be obtained from the source. The communication with the various types of sources is uniformly handled by the WS wrapper interface. As mentioned earlier, all sources may not support this capability.



http://www/~georgem/db/cat-scqd.xml

EnvOntario
EnvToronto
EnvCanada
EnvNY

AirQuality
WaterQuality

location: String
date: String
CO2percentage: Strin
humidity: String

Toronto

Fig. 6. The ViewCat Applet

During the catalog query phase, the catalog answers queries on sources, types and domains, by accessing the data stored in the catalog. The query task is handled by the **ViewCat** applet (shown in Figure 6). On startup, the applet connects to the catalog specified in the command line and displays a list of all the registered sources. Alternatively, users can connect to another WS catalog by typing its address in the **Catalog URL** text area and pressing the **Connect** button. There are two types of catalog addresses:

– a URL of a WS-XML file
  (eg. "http://www.env.org/catalog.xml");
– an RMI url of a remote catalog
  (eg. "rmi://www.env.org/Catalog")

In the first case, the built-in catalog loads and parses the specified file, registering all the sources described therein. In the second case, the applet contacts the specified remote catalog through the remote method invocation protocol [RMI].

Viewing the catalog contents

Upon selection of a source, the set of types available in that source is displayed in a listbox. The lower text area displays the connection information for that source. Upon selection of a type, the set of its attributes (and their datatypes) is displayed in the "Type Definition" listbox. Finally, the active domain of any attribute in the current source can be computed by selecting the attribute of interest from the list. For example, in Figure 6, the domain for attribute `location` is displayed.

Updating the catalog

The applet provides a convenient way to delete and update the metadata about individual sources, as well as save the current catalog contents.

– the **Delete** button removes the currently selected source from the catalog;
– the **Update** button instructs the catalog to re-load the metadata about the selected source from the WS-XML document that was used when the source was registered;
– the **Save** button writes the current content of the catalog in a local WS-XML file (named "catalog.xml"); this file can later be used for catalog initialization;
– the **Close** button quits the applet.

For a more detailed description of the WS architecture, the WSQL query language and its formal semantics, as well as information on calibrating the performance of the prototype we refer to [Mih00].

6 Related Work

Our research is similar to and depends upon work in many areas. Research in mediation technology has proposed several techniques for describing, integrating or accessing structured data on the Internet. WS is based on the assumption that wrappers and mediators will support interoperability. Ongoing research in the information retrieval and digital library community has generated a number of distributed searching protocols and metadata standards in order to facilitate resource discovery for document-like objects and there are some extensions to structured data as well. WS will perform resource discovery for sources with structured data. Finally, various techniques for indexing and querying the

Web have been proposed. WS must extend these techniques to WS-XML pages that publish data sources.

We now consider wrapper mediator architectures, as proposed in [ACPS96, B+97, C+95, G+96, GRVB98, KLSS95, R+89, P+96, TRV96, Wie92]. These systems differ widely in the capabilities of mediators and in the capabilities of wrappers. Wrappers developed for Garlic [C+95] and the Information Manifold [KLSS95] assume the location of the data sources, types, and wrapper capability, are embedded within the wrappers. Wrappers for DISCO [TRV96], TSIMMIS [GM+95, VP97] and for Web accessible WebSources [GRVB98] use declarative languages to export types and query capabilities. Another proposal [WID97] focuses on CORBA IDL access via WWW protocols, instead of query based access, to sources. The WS architecture can be extended to include such data sources.

In all these projects, the knowledge of the location of the sources is embedded within the mediators and there is no resource discovery task. Our research is based on the premise that WS-XML documents will be used to publish the location of components (wrappers and data sources). We thus provide a (single) query language to locate data sources (based on metadata) and to access data from the sources. We note that at present, WS does not deal with limited capability wrappers. Our current prototype interacts with databases, using JDBC, and with data in flat files.

The importance of the World Wide Web as a repository of information has generated an increasing interest in the research community for the design of high-level, declarative languages for querying it. WebSQL [MMM97] integrates textual retrieval with structure and topology-based queries. Instead of trying to model document structure with some kind of object-oriented schema, as in [CACS94, QRS+95], a minimalist relational approach is taken: each Web document is associated with a tuple in a virtual `Document` relation and each hypertext link with a tuple in a virtual `Anchor` relation. In order to query these virtual tables, one must first define computable sub-domains, either using keyword matching or through controlled navigation starting from known URLs. Another Web query language, W3QS [KS95] includes the specification of syntax and semantics of a SQL-like query language (W3QL) that provides simple access to external Unix programs, advanced display facilities for gathered information, and view maintenance facilities. We borrow concepts from these languages in the design of WSQL.

An emerging proposal for a standard for exchange of types and structured data, XMLSchema [XML96, XS00] specified in XML, has been gaining wide attention. WS uses XMLSchema to specify the schema and metadata of sources. When query languages for XML are developed, they can be exploited by WS.

The problem of retrieving information from multiple sources has also received considerable attention in the context of bibliographic data. In order to overcome the difficulties generated by the differences in data representation and query mechanisms, the library community has developed the Z39.50 protocol [Lyn91, Z39], an interoperability standard allowing highly specific searches of distributed data sources. The tasks supported by Z39.50 are resource discovery, query, and retrieval and result presentation. This standard has been extended to accommodate other types of data, in particular, geo-referenced or geo-spatial data. The underlying data model has been extended with GEO objects, and attribute sets have been defined appropriately. We note that different application domains are identified by different attribute sets in Z39.50.

More recently, metadata standards such as the Dublin Core and WebDAV [DC, Web] have been proposed in order to facilitate resource discovery for document-like objects. Current efforts are being made for the representation of Dublin Core metadata through metadata standard exchange formats such as RDF [RDF99], which is an extension of XML. WebDAV (Web Distributed Authoring and Versioning) protocol is an Internet standard and defines metadata (called "properties" in the protocol). The WebDAV Working Group is also investigating DASL (DAV Searching and locating). WS aims to complement these document-centric systems by proposing a metadata-based infrastructure supporting location and access to structured data.

A number of distributed searching protocols for directory information such as LDAP and whois++ [How95, Who96] have been proposed. The content-based source selection features exhibited by these systems are similar in purpose with those present in WS, although they are supported in different ways. The whois++ system for example, maintains a hierarchical collection of *centroids* (lists of words extracted from the textual fields in the sources). Our current architecture is based on a centralized store which provides an index into sources that have been discovered and registered in the catalog. This approach to catalog construction using the source discovery language is at the expense of scalability. This issue will be addressed in future work.

Distributed information retrieval systems, for example the Harvest/Essence information retrieval based system [B+95] are also related to our work. Essence is a customizable information extraction system that is used to extract and structure mostly textual information from documents in Harvest. It exploits the formats of common file types and extracts contents of files. The result is a summary object (a SOIF record) [SOI]. Collections of SOIF records are indexed and organized into *brokers*, a kind of mediator. Brokers provide information retrieval search on their associated SOIF records. The information stored in SOIF records is similar to the metadata about sources maintained by WS catalogs. One difference is that WS stores connection information for the source, including the location of the source and the type of wrapper. Thus WS can contact discovered sources to obtain metadata, if the source supports a query capability.

There are several other projects that deal with scaling to large numbers of resources. InfoSleuth [B+97] proposes information brokering and domain ontologies as two ways to handle data sources. We believe that domain ontologies are a promising extension to the catalog server, since they provide a way to structure domain information (types). The Diorama project develops a methodology and toolkits for intelligent integration and access of heterogeneous information sources in enterprise-wide networking environments. The Diorama system consists of several components that extract properties from unstructured data or collect data through other information brokers/mediators, and dynamically convert and assemble gathered information into DIOM objects. The WS uses existing wrapper technology to connect to sources and uses WWW documents to publish locations of components. To summarize, our contribution is the design of a query language to discover and access data sources.

7 Conclusion

We have presented a system, WebSemantics, that provides an infrastructure for describing and locating sources containing structured data on the Internet. The system consists of a multi-layered architecture of interdependent components. The *Data Source Layer* contains *sources*, which contain data stored in either an active repository (such as a DBMS) or a passive collection of files. It also contains *wrappers*, which are software components that isolate the differences in query capabilities and data exchange formats between data sources. To facilitate the publication of data sources we propose the use of WS-XML documents in the *WWW Layer*, providing data source connection information (data source and wrapper information), a textual description of the data source content, and type and domain metadata. This provides an easy way to publish sources and allows the use of information retrieval techniques for the location of relevant data sources. The third layer, the *Catalog Layer*, consists of *catalogs*, which are specialized repositories storing metadata about data sources. Finally, the *Query Processing Layer* provides a query processor to select sources based on the published metadata.

We defined a declarative query language, WSQL, which facilitates the discovery and registration of data sources in catalogs. It has features to query the metadata in the catalogs to select sources. We then defined an algebra to express each WSQL query as an evaluable tree of operators. Finally, we described the current status of our prototype implementation of the WS system.

We now summarize the limitations of our current prototype.

Our current prototype does not support catalog maintenance as sources change over time. It also does not address the important issue of access control. The right to register/delete sources should be restricted to authorized users. These factors affect scalability.

The current catalog computes the active domain of a newly registered source by executing data extraction queries on these sources. Of course, this is only possible for sources with full query capabilities. For sources with limited query capabilities, we assume the domains are provided by the data publisher, in the WS-XML file that describes the source.

We are currently operating on the assumption of a single catalog. This catalog can be extended in future work so that catalogs can reference each other. This would require an extension to the semantics of the query language. The principal advantage would be that catalogs could be "crawled" in a way similar to the crawling of WWW pages. At the same time, we will explore distributed information retrieval and indexing strategies, to build a searchable index of all WS-XML documents on the WWW.

Finally, we expect to extend the prototype to handle *quality* metadata. This metadata goes beyond domain information to describe the quality of a source, e.g., when was the contents of a source last updated, or what is the granularity of the measurements in the source.

References

[ACPS96] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmaniam. Query caching and optimization in distributed mediator systems. In *Proceedings of the ACM SIGMOD'96*, pages 137–148, 1996.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semistructured data. In *International Journal on Digital Libraries 1(1):68-88*, 1997.

[B+95] C. Bowman et al. The Harvest information discovery and access system. *Computer Networks and ISDN Systems*, 28:119–125, 1995.

[B+97] B. Bohrer et al. Infosleuth: Semantic integration of information in open and dynamic environments. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 195–206, 1997.

[C+95] M. Carey et al. Towards heterogeneous multimedia information systems: the Garlic approach. Technical report, IBM Almaden Research, 1995.

[C+96] R.G.G. Cattell et al. *The Object Database Standard - ODMG 93, Release 1.2.* Morgan Kaufmann, 1996.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proc. ACM SIGMOD'94*, pages 313–324, 1994.

[CHS+95] Michael. J. Carey, Laura M. Haas, Peter M. Schwarz, Manish Arya, William F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic, John Thomas, J. H. Williams, and Edward L. Wimmers. Towards heterogeneous multimedia information systems: The Garlic approach. In *Research Issues in Data Engineering*, pages 124–131, Los Alamitos, Ca., USA, March 1995. IEEE Computer Society Press.

[DC] Dublin core metadata initiative. http://purl.org/DC.

[FMLS99] D. Florescu, A. Manolescu, A. Levy, and D. Suciu. Query optimization in the presence of limited access patterns. In *Proceedings of the ACM SIGMOD Conference*, 1999.

[Fra97] Michael J. Franklin, editor. *SIGMOD Record*, volume 26, March 1997. Special Section on Environmental Information Systems.

[G+96] G. Gardarin et al. IRO-DB: A distributed system federating object and relational databases. In O.A. Bukhres and A.K. Elmagarmid, editors, *Object-Oriented Multidatabase Systems : A solution for Advanced Applications.* Prentice Hall, 1996.

[GHC] The global historical climatology network (GHCN). http://www.ncdc.noaa.gov/ol/climate/research/ghcn/ghcn.html.

[GM+95] H. Garcia-Molina et al. Integrating and accessing heterogeneous information sources in TSIMMIS. In *Proceedings of the AAAI Symposium on Information Gathering*, pages 61–64, Stanford, California, March 1995.

[GMW99] R. Goldman, J. McHugh, and J. Widom. From semistructured data to xml: Migrating the lore data model and query language. In *Proc. of the 2nd Intl. Workshop on the Web and Databases (WebDB '99)*, 1999.

[GPC] Global precipitation climatology centre (GPCC). http://www.dwd.de/research/gpcc/.

[Gra93] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2), june 1993.

[GRVB98] Jean-Robert Gruser, Louiqa Raschid, María Esther Vidal, and Laura Bright. Wrapper generation for web accessible data sources. In *CoopIS'98*, pages 14–23, New York, NY, August 1998.

[How95] Timothy A. Howes. The lightweight directory access protocol: X.500 lite. Technical report, University of Michigan, July 1995.

[KLSS95] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Proc. of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, Stanford, CA, March 1995.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A query system for the World Wide Web. In *Proceedings of VLDB'95*, pages 54–65, 1995.

[Lyn91] Clifford A. Lynch. The z39.50 information retrieval protocol: An overview and status report. *Computer Communication Review*, 21(1):58–70, 1991.

[MCF97] Meta Content Framework using XML, June 1997. http://www.w3.org/TR/NOTE-MCF-XML.

[Mih00] George Andrei Mihaila. *Publishing, Locating, and Querying Networked Information Sources.* PhD thesis, University of Toronto, September 2000.

[MMM97] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. *Journal of Digital Libraries*, 1(1):68–88, 1997.

[MRT98] George Mihaila, Louiqa Raschid, and Anthony Tomasic. Equal Time for Data on the Internet with WebSemantics. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*, pages 87–101, Valencia, Spain, March 1998.

[MW95] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comp.*, 24(6):1235–1258, 1995.

[NGD] National geophysical data centre (NGDC). http://ngdc.noaa.gov/.

[Ora] Oracle home page. http://www.oracle.com.

[P$^+$96] Y. Papakonstantinou et al. Capabilities-based query rewriting in mediator systems. In *Proceedings of PDIS'96*, 1996.

[QRS$^+$95] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Deductive and Object-Oriented Databases, Proceedings of the DOOD '95 Conference*, pages 319–344, Singapore, 1995. Springer.

[R$^+$89] M. Rusinkiewicz et al. Query processing in a heterogeneous multidatabase environment. In *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, 1989.

[RDF99] Resource Description Framework (RDF), March 1999. http://www.w3.org/RDF/.

[RMI] Java(tm) Remote Method Invocation (RMI). http://java.sun.com/products/jdk/rmi/.

[SM] Sun Microsystems. The JDBC(tm) database access API. http://splash.javasoft.com/jdbc.

[SOI] The summary object interchange format (soif). http://harvest.transarc.com/Harvest/brokers/soifhelp.html.

[TRV96] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of DISCO. In *Proceeding of ICDCS'96*, 1996.

[Vid00] M.E. Vidal. *A Mediator for Scaling up to Multiple Autonomous Distributed Information Sources.* PhD thesis, University Simón Bolívar, Venezuela, 2000.

[VP97] V. Vassalos and Y. Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In *Proc. of VLDB'97*, pages 256–265, 1997.

[WDC] World data centre for greenhouse gases (WDCGG). http://jcdc.kishou.go.jp/wdcgg.html.

[Web] World wide web distributed authoring and versioning home page. http://www.ics.uci.edu/ ejw/authoring/.

[Who96] The NSF Whois++ testbed project, March 1996. http://www.ucdavis.edu/whoisplus/.

[WID97] Web Interface Definition Language (WIDL), 1997. http://www.webMethods.com.

[Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, March 1992.

[XML96] Extensible Markup Language (XML), November 1996. http://www.w3.org/XML.

[XS00] XML Schema (W3C working draft), February 2000. http://www.w3.org/TR/xmlschema-0.

[Z39] Library of congress maintenance agency page for international standard z39.50. http://lcweb.loc.gov/z3950/agency/agency.html.

A A DTD for the WS-XML document format

```
<!DOCTYPE ws [

   <!-- The root element -->
   <!ELEMENT ws (source)+>

   <!-- A data source description -->
   <!ELEMENT source (sci, metadata, desc)>

   <!-- The source connection information -->
   <!ELEMENT sci (wrapper, repository)>

   <!-- The type of wrapper required to access
        this source -->
   <!ELEMENT wrapper  EMPTY>
   <!ATTLIST wrapper
             wtype      CDATA  #REQUIRED>

   <!-- The type and location of
        the data source -->
   <!ELEMENT repository  EMPTY>
   <!ATTLIST repository
             rtype      CDATA #REQUIRED
             rlocation  CDATA #REQUIRED>

   <!-- Metadata for the source: which shared schema
        it conforms to and which are the data types
        available in the source -->
   <!ELEMENT metadata (schema, type+)>

   <!-- The URL of the XML-Data document describing
        the shared schema -->
   <!ELEMENT schema (#PCDATA)>

   <!-- The name of a relational type from the shared
        schema. Optionally, one can also include
        domain information for selected attributes -->
   <!ELEMENT type (domain*)>
   <!ATTLIST type
             name   CDATA #REQUIRED>

   <!-- Domain of an attribute -->
   <!ELEMENT domain EMPTY>
   <!ATTLIST domain
             attr      CDATA #REQUIRED
             domtype   (enumeration | range)
             values    CDATA #IMPLIED
             minvalue  CDATA #IMPLIED
             maxvalue  CDATA #IMPLIED>

   <!-- An English description of
        the source's content -->
   <!ELEMENT desc  (#PCDATA)>

]>
```

This article was processed by the author using the LaTeX style file *cljour2* from Springer-Verlag.