The ICS-FORTH Semantic Web Integration Middleware (SWIM)*

Vassilis Christophides, Gregory Karvounarakis,
Aimilia Magkanaraki, Dimitris Plexousakis
Inst. of Computer Science, FORTH
Vassilika Vouton, P.O.Box 1385,
GR 711 10, Heraklion, Greece
{christop, gregkar, aimilia, dp }@ics.forth.gr

Val Tannen
Dept. of Computer and Information Science
Univ. of Pennsylvania, 200 South 33rd St.
Philadelphia, PA 19104-6389, USA
val@cis.upenn.edu

1 Introduction

A cornerstone issue in the realization of the Semantic Web (SW) vision is the achievement of semantic interoperability among legacy data sources spread worldwide. In order to capture information semantics in a machine processable way, various ontology-based formalisms have been recently proposed (e.g., RDF/S [20, 5], DAML+OIL [27], OWL [10]). However, the vast majority of existing legacy data is not yet in RDF/S or any other SW language [23, 25]. As a matter of fact, most of the data is physically stored in relational database (RDB) systems and actually published on the Web or corporate intranets as *virtual* XML.

SW applications, however, require to view data as *virtual* RDF, valid instance of a domain or application specific RDF/S schema, and to be able to manipulate them with high-level query languages, such as RQL [18] or RVL [24]. Therefore, we need middleware systems that can either republish XML as RDF, or publish RDB data directly as RDF, or - even better - be capable of doing both. Sometimes the practical solution will be to rely just on the virtual XML schema and XML query interface of an existing XML publishing system. At other times, the SW publishing middleware will be built as an alternative to the XML publishing system, taking advantage of direct access to the underlying RDB management system (RDBMS). It is also possible that the SW middleware will have to integrate data in some RDBMS with data in native XML storage.

We need to deal flexibly with all these situations in a uniform framework. A decade of experience with information integration architectures based on mediators [8, 29, 26, 21] suggests that it is highly beneficial to (semi)automatically generate such systems from succinct formal specifications, rather than programming their semantics into low-level code. This greatly enhances the maintainability and reliability of the systems in an environment of often revised and shifting requirements. This paper presents the fundamental ideas for devising a comprehensive framework that allows user communities to

1. specify XML $\rightarrow$ RDF and RDB $\rightarrow$ RDF mappings;
2. verify that these mappings conform to the semantics of the employed SW ontologies;
3. compose RQL queries with these mappings and produce XML or RDB queries;
4. specify further levels of abstraction as RDF $\rightarrow$ RDF views;
5. compose RQL queries with such views;
6. perform query optimizations.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was partially supported by the EU project SeLeNe (IST-2001-39045).

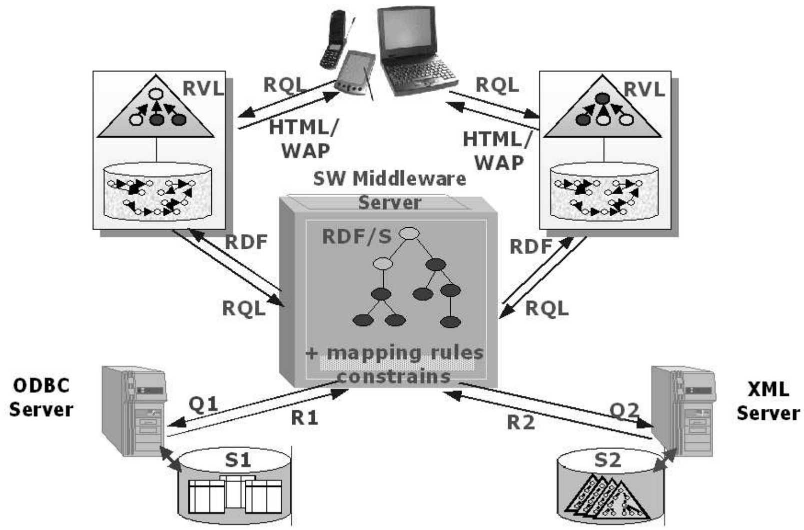

Figure 1: SWIM Architecture

The last requirement is extremely important in such systems. Queries written by humans will rarely have blatant redundancies but queries resulting from automated manipulation/generation are often very ”dumb”. Minimization techniques, sometimes taking advantage of data semantics provided by ontologies expressed in a SW language, can transform such queries into more efficient ones.

Figure 1 sketches the architecture of a SW integration middleware system that we are building, called SWIM. The lower part of the figure depicts data sources, that could be XML repositories or RDBMS. On top of these sources, we have a domain or application ontology for a particular community, expressed, for instance, in RDF/S. Mapping rules can then be used for the integration, i.e., to translate back and forth from RDF/S to the source data models. As a result, through a SWIM server we can view the underlying sources as virtual RDF repositories and use RQL to query these sources as RDF data or even define personalized views on top using RVL. In this context, the main challenge is to choose an expressive, but still tractable logical framework in which the above functionality (1-6) can be effectively supported by appropriate SW (reasoning) services.

This paper only presents our preliminary design for the SWIM framework. We expect to report on many of the technical challenges and engineering decisions in future publications. The remainder of the paper is organized as follows. Section 2 presents a motivating example for cultural data available in RDB or XML sources which can be integrated through an appropriate RDF/S schema. Section 3 presents the internal logical framework of SWIM and its use in the translation and composition of RQL queries. Section 4 touches upon the issue of query optimization by minimization using dependencies while Section 5 presents our conclusions and an outlook for further research.

2 Motivating Example and SWIM Mapping Rules

Let us assume an XML repository with cultural data, a sample of which appears in the left part of Figure 2. This data could be queried using an XML query language, such as XQuery [7]. But now, suppose we add a SWIM server on top of this XML data. For this purpose we design - or import from some community standardization body - an RDF/S cultural schema, as the one depicted in the top part of Figure 2. Now we can formulate queries using an RDF query language by employing only few abstract classes and properties from our mediation RDF/S schema. For example, the following RQL query returns the names of the artists (sculptors or painters) whose work is exhibited in the “Reina Sofia” museum:

```
SELECT Z
FROM   {X}creates.exhibited.denom{Y}, {X}name{Z}
WHERE  Y = "Reina Sofia"
```

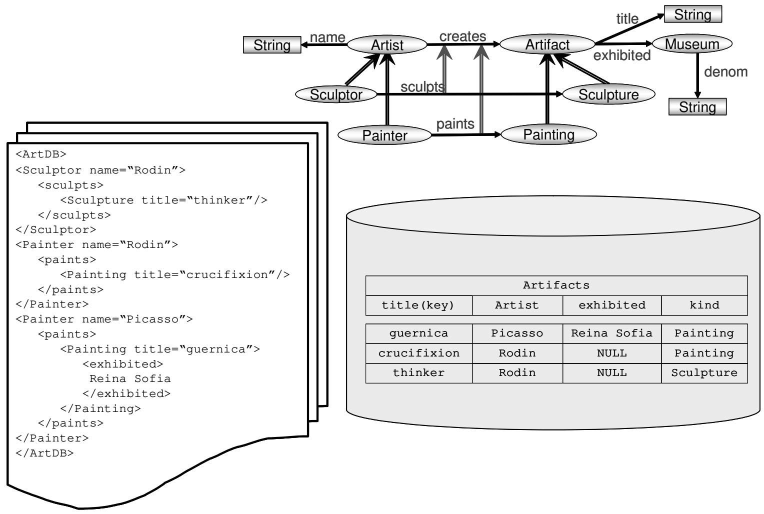

Figure 2: Example of XML/RDF sources and Mediation RDF/S schema

We can observe that the RDF/S layer is completely virtual. The actual data can only be queried using an XML language. Hence, the RQL query we saw needs to be reformulated by the middleware into an XML query. This reformulation should be guided by a formal description of the relationship between the XML and the RDF data, for example a *mapping* from XML to RDF. The question that normally arises is: *how do we express formally such mappings?*

The rich theory developed in the relational case has identified classes of queries and mappings (views) that can be manipulated formally such that various problems like query containment, composing queries with views and rewriting queries with views are algorithmically solvable [1, 15]. These problems can also be solved in the presence of certain classes of relational constraints [1, 11, 14]. We shall try to rely as much as possible on a well-known and robust formalism: conjunctive queries and views, and embedded implicational dependencies [1]. The results about queries and views are easily extended with union, therefore dealing with the positive existential first-order queries also known as non-recursive Datalog. The dependencies can easily be extended with disjunction [12].

To define XML→RDF mappings we will use an analog to the relational queries just mentioned. We use the same logical shape as that of Datalog rules, but instead of relational atoms, we use XPath atoms in the bodies (this is similar to the XBind queries of [14]). For example, the XPath atom `.//Painting (X,Y)` is satisfied by any valuation that maps `X` and `Y` to element nodes in the XML document, such that `Y` has tag `Painting` and is a descendant of `X`. The heads of the rules define RDF instances in the style of the `VIEW` clause employed by the RDF/S view definition language RVL [24]. So, as part of the mapping we can use rules, such as:

```
Painter(X) :-  (//Painter) (X)      Sculptor(X) :-  (//Sculptor) (X)
```

to define the (direct) extent (i.e., the set of direct instances) of the classes `Painter` and `Sculptor` in the virtual RDF layer. Property extents can be also defined in the same style:

```
paints(X,Y) :-  (//Painter) (X),  (.//Painting) (X,Y)
```

Note that this mapping is not always straightforward, since there usually exist schematic and semantic discrepancies between the source and the middleware schema. For example, class inheritance is not expressed in the XML document. Moreover, properties (let alone property inheritance) `creates`, `paints` and `sculpts` are not used explicitly in the XML document. We expect SWIM to be able to take the RQL query and the XML→RDF mapping given above and produce an XML query (e.g., an XQuery). We will discuss in Section 3 how this reformulation can be done.

In addition of being available in XML, the cultural data may be available through an RDBMS, for instance in a table as illustrated in the right part of Figure 2. As for XML, there is an RDB→RDF mapping which is also expressed in a mixed language, where instead of XPath atoms we can use standard Datalog atoms:

```
Painter(X) :- Artifacts(_, X, _, "Painting")
paints(X,Y):- Artifacts(Y, X, _, "Painting")
name(X,Y)  :- Artifact(_, X, _, "Painting"), Y=X
name(X,Y)  :- Artifact(_, X, _, "Sculpture"), Y=X
```

As in the case of XML, there may also be discrepancies in the RDB→RDF mapping. For instance, in our example, the classification of an Artist to `Painter` or `Sculptor` is determined by the value of the attribute `kind`, i.e., schema information is "encoded" inside data values. Again, the SW middleware should be able to automatically reformulate the RQL query, using this mapping, into a relational query, presumably SQL [17].

3 Query Mediation in SWIM

We need an internal logical framework that captures RDF/S semantics, as well as queries, so that we can "virtually populate" given RDF/S schemas. It should also capture - to any needed extent - the XML and RDB semantics. As we showed in the previous section, Datalog-like rules are very convenient for expressing mappings, even across data models, such as XML→ RDF. Based on the experience of [13, 11] of performing XML query reformulation via translation in a first-order, relational framework, we propose to follow the same approach for RDF, in order to translate both queries and mappings into this framework.

3.1 SWIM Internal Logical Framework

The SWIM internal logic framework employs first-order relations together with some first-order constraints to model RDF/S. It is convenient to use a signature with three sorts: $\mathsf{Resource}, \mathsf{Property}, \mathsf{Class}$[1]. The relations used have the following meaning:

- $\mathsf{C\_EXT}(c,x)$ iff the resource x is in the *proper extent* (i.e., it is a direct instance) of class c. In RDF, class extents can overlap, due to multiple classification of resources.
- $\mathsf{C\_SUB}(c,d)$ iff c is a (not necessarily direct) subclass of d.
- $\mathsf{PROP}(c,p,d)$ iff class c is the domain and class d is the range of property p.
- $\mathsf{P\_EXT}(x,p,y)$ iff (x,y) is in the *proper extent* (i.e., it is a direct instance) of property p. In our model instances of properties are represented as ordered pairs of the resources they connect.
- $\mathsf{P\_SUB}(p,q)$ iff p is a (not necessarily direct) subproperty of q.

The relations must satisfy some *built-in* RDF/S constraints which are considered by RQL. In particular, the domain and range of a property must be unique, while the subclass and subproperty relations must be reflexive, transitive and satisfy the following *subproperty/subclass compatibility* constraint:

$$\forall\, a,p,b,c,q,d\ \mathsf{P\_SUB}(q,p) \wedge \mathsf{PROP}(a,p,b) \wedge \mathsf{PROP}(c,q,d) \longrightarrow \mathsf{C\_SUB}(c,a) \wedge \mathsf{C\_SUB}(d,b)$$

This means that if q is a subproperty of p, the domain and range of q are subclasses of the domain and range of p, respectively. Finally, we have the *property-class extent compatibility* constraint, i.e., any instance of a property p connects a pair of instances of some subclasses of the domain and range of p, respectively:

$$\forall\, a,p,b,x,y\ \mathsf{PROP}(a,p,b) \wedge \mathsf{P\_EXT}(x,p,y) \longrightarrow \exists\, c,d\ \mathsf{C\_SUB}(c,a) \wedge \mathsf{C\_SUB}(d,b) \wedge \mathsf{C\_EXT}(c,x) \wedge \mathsf{C\_EXT}(d,y)$$

Let Δ_{RDF} be the set of dependencies (constraints) used to axiomatize the internal RDF/S model.

[1] For simplicity reasons, we ignore metaclasses and metaproperties in this discussion but they can be handled easily in the same way.

Theorem 1: It is decidable whether $\Delta_{\text{RDF}} \models d$ and whether $\Delta_{\text{RDF}} \models Q_1 \sqsubseteq Q_2$, where d is an embedded implicational dependency, Q_1, Q_2 are conjunctive queries and $\sqsubseteq$ is query containment.

It is straightforward to translate the information of an RDF/S schema to the SWIM internal framework as a set of relational facts (in Datalog parlance—an extensional database), involving the relations C_SUB, PROP, P_SUB as well as the names of classes and properties in the schema as constants. Some of the facts obtained from the schema in Figure 2:

C_SUB(`Painting`, `Artifact`) PROP(`Artist`, `name`, `String`) P_SUB(`sculpts`, `creates`)

Note that this set of facts will include all C_SUB and P_SUB reflexivity instances and will be "closed" under transitivity and under subproperty/subclass compatibility.

3.2 Translation of RQL Queries

RQL is a powerful language for querying smoothly both RDF/S schemas and their instances. An RQL *conjunctive* query has the form $ans(\bar{X}) :- C_1, \ldots, C_n$ where C_i's are either RQL class or property patterns (as they appear in the RQL `FROM` clause) or equalities involving variables and/or constants and $\bar{X}$ is a tuple of variables or constants (range restrictions [1] are also required). Many RQL queries are in fact conjunctive queries, e.g., the query given in Section 2 can be written:

```
ans(Z):- {X}creates{V}, {V}exhibited{W}, {W}denom{Y},
         {X}name{Z}, Y="Reina Sofia"
```

Conjunctive RQL queries can then be translated into relational conjunctive queries in the SWIM internal logical framework. Indeed, according to the declarative semantics in [18], RQL patterns have the same meaning as conjunctions of relational atoms. For example:

RQL Pattern	*Internal SWIM Translation*
$\{X;\$C\}@P\{Y;\$D\}$	PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y), C_SUB(c,a), C_SUB(d,b), C_EXT(c,x), C_EXT(d,y)
$\{X\}@P\{Y\}$	P_SUB(q,p), P_EXT(x,q,y)

In the above RQL patterns, X, Y are resource variables, $\$C, \$D$ are class variables (and can be replaced with constant class names), and $@P$ is a property variable (that also can be replaced by a constant property name). Using these patterns, the RQL conjunctive query above translates internally to the following Datalog rule:

$$\begin{aligned} ans(z) \;\; :- \;\; & \mathsf{P\_SUB}(q_1, \texttt{creates}),\ \mathsf{P\_EXT}(x, q_1, v), \\ & \mathsf{P\_SUB}(q_2, \texttt{exhibited}),\ \mathsf{P\_EXT}(v, q_2, w), \\ & \mathsf{P\_SUB}(q_3, \texttt{denom}),\ \mathsf{P\_EXT}(w, q_3, \text{"}\texttt{Reina Sofia}\text{"}), \\ & \mathsf{P\_SUB}(q_4, \texttt{name}),\ \mathsf{P\_EXT}(x, q_4, z) \end{aligned}$$

3.3 Composing Queries with Mappings

Starting with the internal translation of the query, we perform an interesting *partial evaluation* using the RDF schema information, i.e., we evaluate first the schema-part of the query, namely the P_SUB expressions. This is related to partial evaluation of Datalog programs [4]. Because some atoms (e.g., P_SUB$(q_1,$ `creates`$)$) match more than one fact in the schema, what was a single conjunctive query now becomes a (non-recursive) Datalog program. Here is one of the rules in our example (the other two feature `sculpts` and `creates`):

$$\begin{aligned} ans(z) :- \; & \mathsf{P\_EXT}(x, \texttt{paints}, v),\ \mathsf{P\_EXT}(v, \texttt{exhibited}, w), \\ & \mathsf{P\_EXT}(w, \texttt{denom}, \text{"}\texttt{Reina Sofia}\text{"}),\ \mathsf{P\_EXT}(x, \texttt{name}, z) \end{aligned}$$

The next step is to translate into the SWIM internal framework the heads of the rules that define the mappings. For example, a rule (expressed in RVL syntax) defining the extent of the class `Painter` has the head

`Painter(X)`. We translate this into $\mathsf{C\_EXT}(\mathtt{Painter}, x)$. In the same style we can translate the rule defining the extent of the property `paints(X,Y)` into $\mathsf{P\_EXT}(x, \mathtt{paints}, y)$. Thus, the mapping becomes a (non-recursive) Datalog-like program with XPath atoms for the XML→RDF case and a plain non-recursive Datalog program for the RDB→RDF case. The composition of the query and the mapping is now simply the composition of two Datalog programs.

To finish the reformulation, we must still eliminate the intermediate predicates $\mathsf{C\_EXT}, \mathsf{P\_EXT}$ because they are not part of the data sources. This is done with standard matching/substitution but it may increase (square, in fact) the number of rules. In the examples we have looked at so far, however, the resulting union of conjunctive queries can be minimized significantly because many of the rules are unsatisfiable and hence can be discarded.

4 RQL Query Reformulation and Optimization

Continuing the example from Section 3.3, we compose the query with the mapping for the RDB→RDF case. After eliminating the intermediate predicates $\mathsf{C\_EXT}$ and $\mathsf{P\_EXT}$ we obtain a Datalog program with eight rules. Six of these rules, however, are unsatisfiable because their bodies equate distinct constants. Moreover, standard conjunctive query minimization [1] applies to the remaining two rules. The final reformulated query, after optimizations, for the RDB→RDF case is the following union of conjunctive query (a non-recursive Datalog program with two rules):

```
ans(z) :- Artifacts(x, z, "Reina Sofia", "Painting")
ans(z) :- Artifacts(x, z, "Reina Sofia", "Sculpture")
```

Similar transformations are performed in the case of the XML→RDF mapping. We also encounter six unsatisfiable rules: for example in a rule containing both `(//Sculpture)`(y) and `(.//Painting)`(x,y) there is no valuation for y since an XML element cannot have two different tags (i.e., `Sculpture` and `Painting`). The reformulated query for the XML→RDF case is given below:

```
ans(z) :- (//Painter)(x),  (./@name)(x, z),
          (//Painter)(x),  (./Painting)(x, y),
          (//Painting)(y), (./@exhibited)(y, "Reina Sofia")

ans(z) :- (//Sculptor)(x),  (./@name)(x, z),
          (//Sculptor)(x),  (./Sculpture)(x, y),
          (//Sculpture)(y), (./@exhibited)(y, "Reina Sofia")
```

However, the problem of deciding satisfiability of rules with XPath atoms is a difficult one. We expect that the techniques developed in [14] will help with this problem and more generally with the minimization of such queries.

The optimizations we have seen so far do not take into account the specifics of the RDF/S semantics considered by RQL. However, once we have encoded this semantics into the relational dependencies Δ_{RDF} (see Section 3.1) we can use Δ_{RDF} in minimizing queries. For example, by translating into the internal model and by using minimization under dependencies done with the Chase&Backchase algorithm [11] it is possible to show that the conjunctive RQL queries of the form

```
ans(X,@P,Y) :- {X;$C}@P{Y;$D}, rest(X,@P,Y)
```

minimize to (the internal translation of):

```
ans(X,@P,Y) :- {X}@P{Y}, rest(X,@P,Y)
```

thus eliminating several redundant scans over the class variables $\$C$ and $\$D$ (`rest(X,@P,Y)` stands for a boolean predicate whose variables are X, $@P$ and Y only). It should be stressed that if we just translate these queries into SWIM internal conjunctive queries, the results are not equivalent in the absence of Δ_{RDF}. The examples we saw in this section serve as a guide for design decisions regarding what kind of optimization facilities need to be incorporated into SWIM.

5 Conclusions and Future Work

In this paper we presented the principles underlying the design of SWIM (Semantic Web Integration Middleware) and described the components that achieve semantic integration by mapping XML and relational data to RDF. The unifying framework proposed relies on the use of Datalog-like rules for expressing the mappings and reformulating RQL queries. Furthermore, this framework permits the optimization of RQL queries as well as their composition with the specified mappings in order to produce XML or relational database queries. Previous projects sharing similar motivations are described in [2, 3], [28] and [16]. Our approach is closest to that of [2, 3], while using a more expressive language for the specification of mappings and a different ontology query language. The papers [22, 6] present formal specifications of mappings from less structured schemas such as XML and relational to more structured schemas of the same level of complexity as RDF. Languages similar to our Datalog with XPath atoms are also used, for example, in [9, 19]. Finally, compared to the Datalog framework for RDF/S-based query mediation of [28], SWIM ensures the compositionality of queries with views and mappings, as well as, supports advanced optimization and verification services.

Several issues require further investigation. Specifically, we have dealt so far with the case of conjunctive RQL queries and conjunctive RVL view definitions. In both these cases we obtain a translation into non-recursive Datalog programs to which we can apply well-known optimization techniques and for which the problem of determining the consistency of the mappings is decidable. We intend to study the conditions under which similar results can be obtained for a broader class of RQL queries and RVL view definitions. Another issue is the exploitation of knowledge about the source schemas and data in order to perform further optimizations during the reformulation process. SWIM's internal model can also accommodate constraints such as the ones expressible in OWL [10]. It will be interesting to study the optimization potential that stems from the use of such constraints (e.g., uniqueness or disjointness constraints) in query reformulation / minimization.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-Based Integration of XML Web Resources. In *Proc. of the International Semantic Web Conference (ISWC)*, Sardinia, Italy, June 2002.

[3] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Querying XML Sources Using an Ontology-Based Mediator. In *Proceedings of International Conf. on Cooperative Information Systems (CoopIS)*, pages 429–448, Irvine, California, USA, Nov. 2002.

[4] K. Benkerimi and J. Lloyd. A Partial Evaluation Procedure for Logic Programs. In *Proceedings of the North American Conference on Logic Programs*, Austin, TX, USA, 1990. MIT Press.

[5] D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation, Mar. 27, 2000.

[6] A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of the 20th Int. Conf. on Conceptual Modeling (ER 2001)*, pages 270–284, Yokohama, Japan, Nov. 2001.

[7] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery: An XML Query Language. W3C Working Draft, May 2003. See http://www.w3.org/TR/xquery/.

[8] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your Mediators Need Data Conversion! In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 177–188, Seattle, WA, USA, June 1998.

[9] S. Cluet, P. Veltri, and D. Vodislav. Views in a Large Scale XML Repository. In *Proc. of International Conf. on Very Large Databases (VLDB)*, pages 271–280, Roma, Italy, Sept. 2001.

[10] M. Dean, D. Connolly, F. V. Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. OWL Web Ontology Language Reference Version 1.0, W3C Working Draft. Technical report, W3C, Dec. 12, 2002.

[11] A. Deutsch, L. Popa, and V. Tannen. Physical Data Independence, Constraints, and Optimization with Universal Plans. In *Proc. of International Conf. on Very Large Databases (VLDB)*, pages 459–470, Edinburgh, Scotland, UK, Sept. 1999.

[12] A. Deutsch and V. Tannen. Optimization Properties for Classes of Conjunctive Regular Path Queries. In *Proc. of International Workshop on Database Programming Languages*, pages 21–39, Frascati, Italy, 2001.

[13] A. Deutsch and V. Tannen. MARS: A System for Publishing XML from Mixed and Redundant Storage. In *Proc. of International Conf. on Very Large Databases (VLDB)*, Berlin, Germany, Sept. 2003.

[14] A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints. In *Proc. of International Conf. on Database Theory (ICDT)*, pages 255–241, Siena, Italy, Jan. 2003.

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

[16] A. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: data management infrastructure for semantic web applications. In *Proc. of International World Wide Web Conf.*, pages 556–567, Budapest, Hungary, 2003.

[17] ISO/IEC 9075: Information technology – Database Languages – SQL, 1999.

[18] G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle. Querying the Semantic Web with RQL. *Computer Networks*, 42(5):617–640, Aug. 2003.

[19] L. Lakshmanan and F. Sadri. XML Interoperability. In *Proc. of the International Workshop on the Web and Databases (WebDB)*, San Diego, California, USA, June 2003.

[20] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, Feb. 1999. Available at http://www.w3.org/TR/REC-rdf-syntax.

[21] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of International Conf. on Very Large Databases (VLDB)*, pages 251–262, Bombay, India, Sept. 1996.

[22] B. Ludäscher, A. Gupta, and M. Martone. Model-Based Mediation with Domain Maps. In *Proc. of IEEE International Conf. on Data Engineering (ICDE)*, Heidelberg, Germany, Apr. 2001.

[23] A. Magkanaraki, S. Alexaki, V. Christophides, and D. Plexousakis. Benchmarking RDF Schemas for the Semantic Web. In *Proc. of the International Semantic Web Conference (ISWC)*, Sardinia, Italy, June 2002.

[24] A. Magkanaraki, V. Tannen, V. Christophides, and D. Plexousakis. Viewing the Semantic Web Through RVL Lenses. In *Proc. of the Int'l Semantic Web Conf. (ISWC)*, Sanibel Island, Florida, Oct. 2003.

[25] L. Mignet, D. Barbosa, and P. Veltri. The XML web: a first study. In *Proc. of International World Wide Web Conf.*, pages 500–510, Budapest, Hungary, May 2003.

[26] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *Proc. of IEEE International Conf. on Data Engineering (ICDE)*, pages 251–260, Taipei, Taiwan, Mar. 1995.

[27] F. van Harmelen, P. Patel-Schneider, and I. Horrocks. Reference description of the DAML+OIL ontology markup language. http://www.daml.org/2001/03/reference.html, Mar. 2001.

[28] W. Nejdl and B. Wolf and C. Qu and S. Decker and M. Sintek and A. Naeve and M. Nilsson and M. Palmér and T. Risch. EDUTELLA: A p2p networking infrastructure based on rdf. In *Proc. of International World Wide Web Conf.*, Honolulu, Hawaii, USA, May 2002.

[29] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, 1992.