Views for light-weight web ontologies

Raphael Volz
Institute AIFB
University of Karlsruhe
D-76128 Karlsruhe, Germany
volz@aifb.uni-karlsruhe.de

Daniel Oberle
Institute AIFB
University of Karlsruhe
D-76128 Karlsruhe, Germany
oberle@aifb.uni-karlsruhe.de

Rudi Studer
Institute AIFB
University of Karlsruhe
D-76128 Karlsruhe, Germany
studer@aifb.uni-karlsruhe.de

ABSTRACT

The Semantic Web aims at easy integration and usage of content by building on a semi-structured data model where data semantics are explicitly specified through ontologies. However, ontologies and thereby ontology-based applications themselves suffer from heterogeneity. Therefore a new level of data independence is required to allow the customization of information, e.g. towards the needs of other agents, which can be achieved by exploiting database view principles. This paper addresses this issue and presents a new view mechanism for the data models underlying the Semantic Web, RDF and RDFS.

1. INTRODUCTION

The vision of the Semantic Web incorporates distributed content that is accessible through a standardized semi-structured data model (RDF) and at an explicit conceptual level. The conceptual level is not given by a fixed schema, but rather by an ontology that specifies the formal semantics of content. For this purpose, RDF Schema (RDFS) has been devised as a particular vocabulary within RDF and serves as a light-weight ontology language.

The use of ontologies in real-world applications such as community portals has shown that they can enhance interoperability between heterogeneous information resources and systems on a semantic level. However, what has also become clear is that ontologies and thereby ontology-based applications themselves suffer from heterogeneity. This leads to difficulties when several communities try to establish a way of communication while using diverse ontologies. On the one hand, not all information that is accessible within one community (i.e. a department) might be intended to be accessible to other communities. On the other hand, overlapping content might be represented in different ways.

Therefore a new level of data independence is required to allow customization of information towards the needs of other agents, which can be achieved by exploiting database view principles.

Views on RDF(S) data constitute an unexplored terrain. In particular, existing view mechanisms for relational or semi-structured data lack the explicit conceptual model germane to RDF(S), such as inheritance semantics. Existing view mechanisms for object-oriented data lack the flexibility required in an open world, such as lack of typing or partially available attributes.

In this paper we pick up the unique situation of data in the Semantic Web and propose an OO-like view mechanism that allows easy selection, customization and integration of Semantic Web content.

The central objective of our approach is to acknowledge the underlying intention of Semantic Web, i.e. adding explicit formal semantics to Web content. Therefore views are classified in the semantically appropriate location in RDFS inheritance hierarchies. Autonomous agents are thereby enabled to discover the semantics of the view wrt. to the set-inclusion semantics of inheritance.

Second, compose-ability and querying of views is ensured by maintaining the underlying distinction between classes and properties taken in the ontology representation. This leads to a distinction of views on classes and views on properties.

The paper is structured as follows. Section 2 introduces the associated data models as well as the RDFS ontology representation. Section 3 sketches the query language RQL [3], on which our view mechanism is based. Section 4 describes the ideas underlying our approach to views. Section 5 details views on classes and views on properties and addresses the issue of classification of views into the proper position of the inheritance hierarchies. Section 6 reports briefly on our implementation effort. In section 7 we take position to related work. We conclude summarizing our approach and giving future directions.

2. THE SEMANTIC WEB

2.1 RDF — Semi-structured Data

Initially, the Resource Description Framework (RDF) [10] was intended to enable encoding, exchange and reuse of structured metadata describing Web-accessible resources. Data is encoded using so-called resource-property-value triples, which are also called statements.

Individual information objects are represented in RDF using a set of statements describing the same resource. Object identity can be given via an uniform resource identifier (URI) that labels the resource. This object identifier is globally unique.

A set of statements, i.e. a RDF model, constitutes a partially labelled directed pseudograph. The fact that properties can have multiple values, e.g. 'x:email' for the resource 'x:Rudi' in figure 1, allows to combine statements from different RDF models very easily.

The data model distinguishes between two types of values: resources leading to object associations or literals establishing object attributes. In figure 1 'x:Raphael' is a resource, whereas the name 'Volz' is a literal.

SAC 2003 Melbourne, Florida, USA

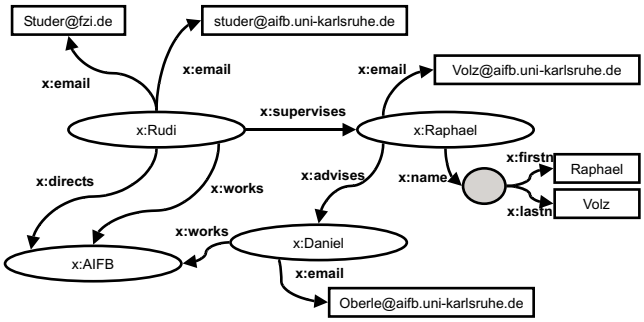

Figure 1: A simple, exemplary RDF model

2.2 RDFS – Light-weight ontologies

Ontologies provide a formal and shared conceptualization of a particular domain of interest. In the Semantic Web a light-weight (in comparison to classical knowledge representation languages) approach is currently in use. Ontologies are constructed from classes and properties that are embedded in a class and a property inheritance hierarchy, respectively. The proposed standard for the Semantic Web is RDF Schema (RDFS)[7]

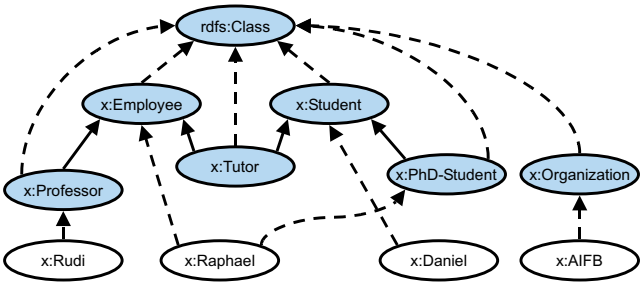

Figure 2: Class hierarchy in RDFS for a simple ontology. Dashed lines denote instantiation, solid lines denote subsumption.

RDFS incorporates a unique notion of object orientation. It introduces classes and a subsumption hierarchy on classes (compare Figure 2). In RDFS subsumption allows for multiple inheritance and has set-inclusion semantics. As the subsumption establishes a partial order, class equivalence can be expressed via a cyclic class hierarchy. The extension of a class is defined by explicit assignment of resources to (possibly multiple) classes.

Unlike in commonly-known object-orientated data models attributes and associations are not defined with the class specification itself. Instead, class properties are first-class primitives themselves. The specification of a property, namely the definition of domain and range constraints, defines the context, i.e. the resource-value-pairs, in which a property may be validly used. in a RDF statement.

RDFS also allows to build a partial-order on properties, e.g. in figure 3 the property 'x:advises' is a specialization of the property 'x:responsible_for'.

3. QUERY LANGUAGE

RQL [3] is the query language used within our approach and is the only RDF query language that takes the semantics of RDFS ontologies into account. The need to be aware of these semantics is the main reason why query languages operating on the syntactic XML-serialization (e.g. XQuery [9]) fail to meet our goals. Due to lack of space, we can only give a short introduction to RQL in this

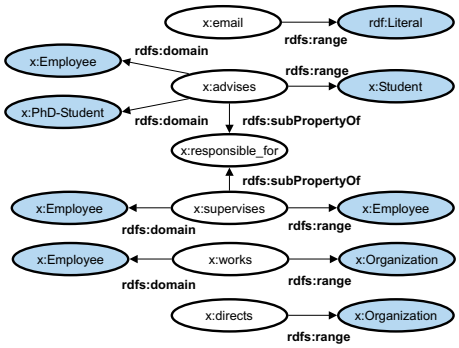

Figure 3: Properties in RDFS for a simple ontology

section. The interested reader may refer to [3] for a more in-depth description.

RQL follows a functional approach. Its basic building blocks are generalized path expressions which offer navigation in the RDF graph. The graph itself is viewed as a collection of elements which can be accessed in generalized path expressions. For example, the following query would return the collection of all pairs of nodes which are related via the property email:

```
SELECT X,Y FROM {X} x:email {Y}
```

RQL queries follow the basic select-from-where construct known from SQL. The construct {X}x:email{Y} is called a basic data path expression, the atom of all path expressions. The variables X and Y, introduced via the {} notation, are bound to the resources and values of those RDF statements that use the property x:email.

RQL permits the interpretation of the superimposed semantic descriptions offered by one or more ontologies. For instance, the inheritance hierarchy is considered when accessing class extents. Also path expressions can be concatenated by a ".", which is just a syntactic shortcut for an implicit join condition.

```
SELECT Y FROM Student{X}.x:advises{Y}
```

This query returns the identifiers of all students advised by other students. Since class subsumption is taken into account, the result would contain the PhD-Student "x:Raphael" from the RDF model depicted in Figure 1.

Furthermore, RQL supports set operators, such as union, intersection and difference and means for pattern matching in where-clauses.

4. VIEW LANGUAGE DESIGN

From the perspective of relational and object-oriented databases it is natural to consider views as arbitrary stored queries. This is not apt for RDF due to several reasons:

1. Such views are not related to the semantic description provided by an ontology. Especially the fact that queries are given a name does not say anything about semantics of the query result.

2. Query results might be n-ary relations. This does not fit in the structure of an RDF model, which is a graph.

For those reasons we cannot rely on previous work about views for other semi-structured data models. The views of [17] consist of object collections only, associations between objects - which are

fundamental to RDF - are not considered. [2] does consider such edges, but does not meet the first issue. Our approach addresses those issues in its fundamental design.

The first issue results in the distinction between views on classes and views on properties. The creation of views extends the ontology as new (although virtual) classes or properties are created.

The semantics of these extensions can only be understood if these newly defined resources are related to established (possibly view) classes and properties by declaration of the subproperty and subclass relation. This can happen in two ways: As the semantics of an operation that defines a view should be properly represented, this relation should be established automatically. For important cases this can be done at the time of view creation by applying description-logic ideas to the analysis of the query definition. This feature is briefly discussed in section 5 along with consistency issues. On the other hand this information cannot be deduced in many cases due to undecidability [14, 4]. It may therefore be done via explicit assignment of these relations by the user. Here, the implementation can only guarantee basic consistency such as compatibility of domain and range classes in properties.

The latter issue results in the limitation that the definition of views can only involve queries which return either unary (views on classes) or binary (views on properties) tuples. This ensures that views are functional such that they can be used within queries and composed into other views. The user can choose arbitrary (previously unused) identifiers for his views.[1].

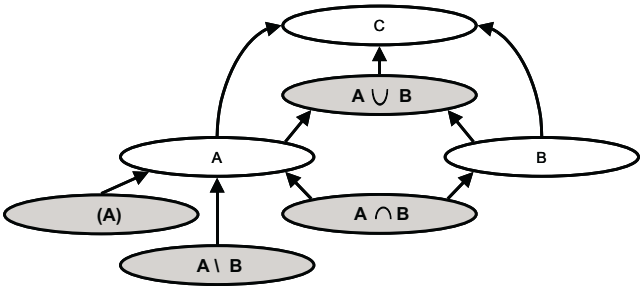

Figure 4: Placement of views in the hierarchy

Automatic classification

Figure 4 provides an overview about the classification which is semantically correct for each algebraic operator. This classification is independent of the type of views.

The illustrated classifications in Figure 4 are motivated by the set-inclusion semantics of inheritance in RDF: Selection always reduces the initial set and creates subsets. Therefore the class which is subject of the selection subsumes the view.

The difference operator can always be rewritten into an equivalent selection (involving negation) on the minuend. Therefore the minuend subsumes the view.

For union the extentions of the unified entities are subsets of the view consequently the view subsumes the unified entities. The view itself is subsumed by the least common superentity of all unified entities. For the intersection operator the opposite fact hold. The intersection is a subset of the extents of all intersected entities. Therefore every intersected entity subsumes a view based on the intersection operation.

For class views, several additional premises hold. Since class views can only be defined via queries that give unary results, operators like joins or the cartesian product are not relevant here. Joins in a query automatically decompose to a selection wrt. the unary result of the query. [2] The cartesian product creates at least binary results. Duplicates are automatically removed in the interpretation of the result since we regard the result as a set of instances.

[1]Note, that this automatically excludes all identifiers used within the specifications of RDF and RDFS

5. VIEWS IN DETAIL

As mentioned in the previous section we make the basic distinction between views on properties and views on classes which is also made in the underlying representation of RDF Schema. Due to lack of space, we can only give examples here. A more detailed technical report that also presents the BNF-syntax of the language can be found at http://kaon.semanticweb.org/views/.

5.1 Views on classes

The definition of views on classes involves two components: First, users must define an arbitrary RQL query which returns unary tuples of resources. These tuples constitute the instances that are in the extent of the view. Second, the view must be properly classified in the class hierarchy. This enables the understandability of the view by using the semantics of the subclass relationship.

```
CREATE CLASS VIEW <URI>
    { SUBCLASSOF <SUPER_URI_i> }
    USE
    SELECT INSTANCE
    FROM ...
    [ WHERE ... ]
```

Figure 5: Definition framework for views on classes

For example, one could characterize the class of all ”Wis. Mitarbeiter”, which is the job title PhD-Students usually have in Germany since they are employed and advise other students:

```
CREATE CLASS VIEW x:WisMitarbeiter
    SUBCLASSOF x:Employee
    USE
    (SELECT X
    FROM x:Employee{X})
    INTERSECT
    (SELECT X
    FROM x:PhD-Student{X},
    {Y} x:advises {Z}
    WHERE X = Y)
```

The fact that users can use arbitrary unary RQL queries within this syntax has two main consequences.

1. *Restricted Updatability* as it is impossible to propagate inserts towards classes defined in such a way. Nevertheless modifications and deletions are possible since object identifiers are known. This kind of updates is propagated directly to the underlying base data. Of course, instances might disappear from the view if they do not meet the query conditions anymore.

2. *Manual Classification* As users can combine arbitrary algebraic operations in the view[3] the semantic characterization of

[2]Since joins can be rewritten to a selection on the cartesian product. As we can only regard one row of the relation the results of the cartesian product vanish leaving the selection behind.

[3]except for the final projection that fixes the arity

the view can not be given automatically since this problem is undecidable [14, 4].

The latter fact leads to the introduction of additional possibilities to define views on classes conveniently where the classification can automatically be determined from query semantics.

Convenience definitions

Consequently we ammend our syntax by convenience definitions which allow to define views via selection and set operations and generate the standard class view definition automatically by transformation. Especially the classification of the view via the subclassof clause is automatically generated.

Selection views. The convenience operation provided by the system views on classes is selection, where a subset of the members of a certain class is selected. The user is still able to pose arbitrary RQL queries. The system ensures that the result of the query is a selection on the base class (via a mandatory ON clause). Technically, this is done by translating the ON clause into the generated query by appending the following expressions to the where-clause of the query itself:

```
AND INSTANCE IN
   (SELECT M FROM BASE_URI{M} )
```

This expression provides a kind of type casting which ensures that the variable INSTANCE can only take members of the base class as a value.

The following definition creates a new class "x:Supervisors", which is populated with those employees who supervise someone.

```
CREATE CLASS VIEW x:Supervisors
ON x:Employee
    USE
    SELECT INSTANCE
    FROM {INSTANCE} x:supervises
```

The following standard class view definition (cf. 5) is compiled from the provided view specification:

```
CREATE CLASS VIEW x:Supervisors
SUBCLASSOF x:Employee
USE
    SELECT INSTANCE
    FROM {INSTANCE} x:supervises
    WHERE INSTANCE IN
    (SELECT M FROM x:Employee{M})
```

Following the discussion in section 4 the view is made a subclass of x:Employee. Additionally the above mentioned type casting expression was appended to the where clause.

Difference views. A second convenience syntax is defined for the difference operator. For example, the creation of virtual class "x:Unemployed-Students" which is populated with all students that have no job, can be stated as follows:

```
CREATE CLASS VIEW x:Unemployed-Students
ON x:Student
MINUS x:Employee
```

Again, the translation of the convenience syntax to the standard view definition involves generation of the subpropertyof statement which corresponds to the minuend. Additionally the appropriate RQL query must be generated.[4]

Union view. Another convience syntax allow to define class views via the union of classes. For example a class view "x:Scientists" which consist of professors as well as PhD-Students is created by the following definition:

```
CREATE CLASS VIEW x:Scientists
ON x:Professor
UNION x:PhDStudent
```

The translation of this definition into the standard form involves the computation of the least common super class of the unified classes.

With respect to our ontology example in figure 2 no common superclass of "x:Professor" and "x:PhD-Student" can be found. Therefore the view has no super class and the subclassof statement is ommitted in the translation. The generation of the RQL query which is used in the translated definition is straightforward.[5]

Intersection view. The intersection operator conceptually follows the principles used for Union View, but here the translation of the convenience syntax to a standard definition involves the generation of at least two subclassof statements as well as the appropriate RQL query.[6]

5.2 Views on Properties

Views on properties can be defined using arbitrary queries which return binary tuples[7]. Besides the query itself, several additional information is required to define a view on properties. This involves *(a)* the definition the domains and ranges of the view, *(b)* embedding the view into the property hierarchy and *(c)* forcing the query to return binary tuples.

```
CREATE PROPERTY VIEW <URI>
    { SET DOMAIN <DOMAIN_URI_i> }
    { SET RANGE <RANGE_URI_i> }
    { SUBPROPERTYOF <SUPER_URI_i> }
    USE
    SELECT DOMAIN, RANGE
    FROM ...
    [ WHERE ... ]
```

Figure 6: Definition framework for views on properties

Since the query can involve arbitrary joins and aggregation the update-ability of views based on arbitrary queries cannot be automatically ensured. Additionally, the consistency of the view definition with respect to constraints and property inheritance must be ensured at compile time.

View consistency

Consistency of domain and ranges. First, it must be ensured that only tuples are returned that meet the constraints specified by the domain and range definition. Technically, this is implemented similarly to the type casting used for selection views on classes.

[4]For $M \setminus S$: `(SELECT X FROM M{X}) MINUS (SELECT X FROM S{X})`

[5]For $M \cap S$: `(SELECT X FROM M{X}) UNION (SELECT X FROM S{X})`

[6]For $M \cap S$: `(SELECT X FROM M{X}) INTERSECT (SELECT X FROM S{X})`

[7]Each tuple must be either (resource $\times$ resource) or (resource $\times$ literal) to reflect that literals cannot be in the domain of RDF properties

Consistency of the property hierarchy. The domain and range definitions of the view must be compatible, i.e. connected in the class hierarchy, to the domain and range definitions of the super properties. We can check this at compile time via appropriate consistency rules (cf. [13]).

An example. The following definition creates a new property view which relates all PhD-Students with email addresses of the advised students.

```
CREATE PROPERTY VIEW x:mails_of_advised
    SET DOMAIN x:PhD-Student
    SET RANGE  rdf:Literal
    SUBPROPERTYOF x:email
    USE
    SELECT DOMAIN, RANGE
    FROM x:PhD-Student{DOMAIN}.
    x:advises{Y}. x:email{RANGE}
```

The definition is consistent. This is due to the fact that no domain constraints have been specified for the super property x:email. No conflicts arise for the range since it is the same. The query is modified to the following form to ensure domain and range compatibility:

```
    SELECT DOMAIN, RANGE
    FROM x:PhD-Student{DOMAIN}.
         x:advises{Y}. x:email{RANGE}
    WHERE DOMAIN IN
    (SELECT M FROM x:PhD-Student{M} )
```

Here an exception for casting the range applies since the range is a literal. The domain is ensured to be in the extent of x:PhD-Student by an appended where clause.

Convenience definitions

As we can see the definition of views on properties involves quite a lot of information from the user. Again, a set of short hand notations simplifies the construction of property views by users.

Renaming properties. The following definition creates a new property view that is populated with the extension of a base property :

```
CREATE PROPERTY VIEW <VIEW_URI> ON <BASE_URI>
```

Domain and range constraints as well as subproperty relationships are taken from the base property by copying the definitions made there. The required RQL query is automatically generated.[8] Such views are updateable since no joins or aggregations are involved.

Refining properties. A special definition selects only those instances from the underlying property extent that match the extents of the specified domains and ranges. Here the consistency of domains and ranges must be ensured again and can rely on the previously mentioned means. The generation of the query is straightforward since it only involves appending the above mentioned typecasting expressions to the where-clause of the query.

The view is a specialization of the base property since it is defined via a selection on the extent of the base property. Therefore the base property subsumes the view. Additionally the view can be updated.

[8]That is: SELECT DOMAIN, RANGE FROM {DOMAIN} base_uri {RANGE}

For example, the following definition refines the property "x:email" from Figure 1 to carry only email addresses of Students.

```
CREATE PROPERTY VIEW x:student-mail
SET DOMAIN x:Student
ON x:email
```

Applying the above-mentioned arguments leads to the translation of the convenience definition to the following standard property view definition:

```
CREATE PROPERTY VIEW x:student-mail
SET DOMAIN x:Student
SET RANGE rdf:Literal
SUBPROPERTYOF x:email
USE
SELECT DOMAIN, RANGE
FROM {DOMAIN} x:email {RANGE}
WHERE DOMAIN IN
    (SELECT M FROM x:Student{M})
```

Other selections and set operations. Several other convenience definitions for the algebraic operations of selection and set operations are defined, which are not presented here due to lack of space. They are conceptually similar to the convenience operations for classes and follow the same pattern of automatic definition of domains and ranges, automatic classification in the property hierarchy and query generation which was shown for the above convenience definition for property views.

6. IMPLEMENTATION

We give a brief survey about our ongoing implementation effort. The first prototype of the view mechanism is already implemented within KAON Server, a multi-user capable, transactional web ontology repository. KAON Server is part of the open-source KAON tool suite [6].[9] Clients are able to use the object-oriented KAON-API built upon the W3C's RDF-API to access ontologies and corresponding RDF data. The KAON-API offers the transparent access to views which is also constituted in the query language.

View definitions are stored in RDF syntax. This approach allows to reuse the data structures provided to store data for managing schematic information as it is done for the specification of ontologies.

As soon as a view is created by the user consistency checks are performed. For this purpose, we defined a set of axioms expressing consistency. This involves some inference processes are started as we use a logic programming system, namely SiLRI [8], to check the axioms.

SiLRI is also used to implement the query engine. Therefore the agreggation functions which were specified for RQL are not implemented yet. Our prototype does not support updates yet, thus views remain read-only.

7. RELATED WORK

There is a large body of work on views for the relational data model. These results are already incorporated in many database textbooks. Our approach differs substantially from relational world. Few research has actually focused on views for semi-structured data. The approach presented in [17] consist of object collections only and does not allow object links. [2] reflects the importance of

[9]http://kaon.semanticweb.org/

such links. Nevertheless our proposal is the only one that takes a superimposed conceptual model into account, viz. the ontology. Additionally consistency constraints such as situated by the RDF(S) data model are not considered in those approaches.

There is a large amount of work done on views for object-oriented data (e.g. [5, 1, 14, 15]). Generally those approaches do not fit the Semantic Web despite the conceptual similarity of object-oriented data models to web ontologies. This is mainly due to the explicit typing of classes and local assignment of properties to classes. Additionally web criteria such as the ability to base views on multiple data sources are not met. Also property hierarchies are not known to object-oriented models.

Nevertheless we have combined many aspects presented there. The idea to classifiy views in the hierarchy was first proposed in [15]. Other approaches, i.e. [5], do not mix view and class hierarchies. Like the majority of object-oriented approaches to views we mainly support object-preserving views, although not discussed here. Many object-oriented allow to support other forms of view population such as set-tuples (akin to the relational world) and of object-generation. The latter was first presented in [5] which present arguments such as its usefulness for simulated schema evolution. Some formation of external schemata proposed in [1, 15] however views can only be introduced in this context and not be added to the base database.

8. DISCUSSION

We have presented a view mechanism that picks up the unique situation of data in the Semantic Web. We propose a view mechanism that allows easy selection, customization and integration of Semantic Web content. Our approach acknowledges the underlying intention of the Semantic Web - to add explicit formal semantics to Web content - and exploits the semantics of view definitions as far as possible to classify views into the semantically appropriate position in the entity hierarchies provided by RDFS. This allows agents to understand the semantics of the views autonomously. If the vocabulary of another ontology is used in the view definitions otherwise disparate ontologies are integrated by establishing is-a links between the classes and properties of both vocabularies leading to a proper articulation of both ontologies [12].

From our perspective, a view mechanism is an important step in putting the idea of the Semantic Web into practice. Based on our own experiences with building Semantic Web based community portals [11, 16] and knowledge management frameworks [6] we devise that view mechanisms for ontology-based semi-structured data will be a crucial cornerstone to achieve many different, exciting objectives.

Examples for such objectives will be personalized access to meta-data bases in community portals, authorization and the improved integration of ontologically disparate information sources — to name but a few.

For the future much remains to be done. We are currently investigating how updates can be consistently integrated. Additionally the materialization of views is of great importance in Web scenarios, we are therefore also investigating how such materialized views can be incrementally maintained in presence of updates. We also plan to adapt the implicit classification approach to allow full description-logic style subsumption which might have benefits for using views in query rewriting.

Acknowledgement:. This work was funded through the WonderWeb programm (EU-IST-2001-33052). We thank our colleague Steffen Staab for his contributions to the paper.

9. REFERENCES

[1] Serge Abiteboul and Anthony Bonner. Objects and Views. In *Proc. Intl. Conf. on Management of Data*, pages 238–247. ACM SIGMOD, May 1991.

[2] Serge Abiteboul, Roy Goldman, Jason McHugh, Vasilis Vassalos, and Yue Zhuge. Views for semistructured data. In *Proceedings of the Workshop on Management of Semistructured Data, Tucson, Arizona*, May 1997.

[3] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, K. Tolle, Bernd Amann, Irini Fundulaki, Michel Scholl, and Anne-Marie Vercoustre. Managing RDF metadata for community webs. In *(WCM'00), Salt Lake City, Utah*, pages 140–151, October 2000.

[4] C. Beeri. Formal Models for object oriented databases. In *Proc. 1st Intl. Conf. on Deductive and object-oriented databases*, pages 370–396, 1989.

[5] Elisa Bertino. A View Mechanism for Object-Oriented Databases. In *Advances in DB-Technology, Proc. Intl. Conf. on Extending Database Technology (EDBT)*, number 580 in Lecture Notes in Computer Science, pages 136–151, Vienna, Austria, March 1992. Springer.

[6] Erol Bozsak and al. Kaon — towards a large scale semantic web. In *Proc. of 3rd Int'l Conference on Electronic Commerce and Web Technologies (EC-WEB 2002)*, Aix-en-Provence, France, September 2002.

[7] Dan Brickley and R. V. Guha. Resource description framework (RDF) schema specification 1.0. Internet: http://www.w3.org/TR/2000/CR-rdf-schema-20000372/, 2000.

[8] S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for RDF. In *QL98 - Query Languages Workshop*, December 1998.

[9] S.J. DeRose. Xquery: A unified syntax for linking and querying general xml documents. *Query Languages 1998*, 1998.

[10] O. Lassila and R. Swick. Resource description framework (RDF) model and syntax specification. Internet: http://www.w3.org/TR/REC-rdf-syntax/, 1999.

[11] Alexander Maedche, Steffen Staab, Rudi Studer, York Sure, and Raphael Volz. Seal tying up information integration and web site management by ontologies. In *IEEE Data Engineering Bulletin*, volume 25, March 2002.

[12] Prasenjit Mitra, Gio Wiederhold, and Martin L. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proc. of Extending Database Technology (EDBT) 2000*, pages 86–100, 2000.

[13] Daniel Oberle and Raphael Volz. Implementation of an view mechanism for ontology-based metadata. Technical Report 422, Institute AIFB, University of Karlsruhe (TH), 2002.

[14] Elke A. Rundensteiner. MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases. In *Proc. 18th Intl. Conf. on Very Large Data Bases (VLDB)*, pages 187–198, Vancouver, Canada, 1992. ACM SIGMOD.

[15] Marc H. Scholl, Christian Laasch, and Markus Tresch. Views in Object-Oriented Databases. In *Proc. 2nd Workshop on Foundations of Models and Languages of Data and Objects*, pages 37–58, September 1990.

[16] S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer, and Y. Sure. Semantic community web portals. In *WWW9 - Proceedings of the 9th International World Wide Web Conference, Amsterdam, The Netherlands, May, 15-19, 2000*. Elsevier, 2000.

[17] Y. Zhuge and H. Garcia-Molina. Graph structured views and their incremental maintenance. In *Proc. 14th Int. Conf. on Data Engineering*, 1998.