Implementing Views for light-weight Web Ontologies

Raphael Volz, Daniel Oberle, Rudi Studer

Institute AIFB
University of Karlsruhe (TH)
D-76128 Karlsruhe
Germany
email: {volz,oberle,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. The use of ontologies in real-world applications such as community portals has shown that a new level of data independence is required for ontology-based applications. For example, the customization of information towards the needs of specific user communities is often need. This paper extends previous work [22, 21] on this issue and presents a view language for the fundamental data models of the Semantic Web, viz. RDF and RDFS, and how it can be implemented. The basic novelty of the view language is the semantically appropriate classification of views into inheritance taxonomies based on query semantics. Additionally, the underlying distinction between unary predicates (classes) and binary predicates (properties) taken in RDF/S is maintained in the view language. So-called external ontologies allow the integration of multiple source databases, offer control over the publishing of data and enable the generation of views spanning across databases.

1. Introduction

The vision of the Semantic Web incorporates distributed content that is machine understandable by relying on an explicit conceptual level. It builds on RDF [13], which is a semi-structured data model that allows the definition of directed labelled graphs. The required 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. It introduces a property-centric approach and allows to partition data into classes without expressing strict typing. Additionally, inheritance hierarchies on both classes and properties are provided.

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 (e.g. 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.

Contribution of the paper In this paper we show how a view language that picks up the unique situation of data in the Semantic Web and allows easy selection, customization and integration of Semantic Web content can be implemented.

The central objective of this implementation is to acknowledge the underlying intention of the Semantic Web, i.e. adding explicit formal semantics to Web content. Therefore the implementation must ensure that views are classified to the semantically appropriate location in RDFS inheritance hierarchies.

Second, we maintain 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. Hence, views can be composed and used within queries.

Third, our approach supports the construction of external ontologies by grouping views and base entities into new data sets to allow customization and integration of multiple databases towards application demands or other user communities.

Our approach facilitates data integration and usage and paves the way to Semantic Web information systems that nourish from various data sources and feed back into many different data sinks - like our SEmantic portAL (SEAL) [14]. Such a view mechanism 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 metadata bases, authorization and the improved integration of ontologically disparate information sources - to name but a few.

The paper is structured as follows. Section 2 introduces the associated data model and ontology representation language proposed for the Semantic Web and discusses the query language that is used in our approach. In Section 3 the underlying design decision for an apt view language is presented. Then, Section 4 provides a brief look at the proposed view language and presents some example views. Section 5 introduces the notion of external ontologies that support the aggregation and integration of distributed data. Section 6 sketches the implementation issues before we talk about related work in section 7 and conclude.

2. The Semantic Web

2.1. RDF - A semi-structured data model

The underlying data model of the Semantic Web is the Resource Description Framework (RDF) [13]. It is a semi-structured data model that was initially intended to enable the 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 is given via the uniform resource identifier (URI) that labels the resource [1]. This object identifier is globally unique.

A set of statements constitutes a partially labelled directed pseudograph[2] and is commonly called an RDF model. The fact that properties can have multiple values, e.g. 'x:email'[3] 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. A value can either be another resource leading to object associations or may be a literal establishing object attributes. For example, in Figure 1 'x:Raphael' is a resource, whereas the name 'Volz' is a literal.

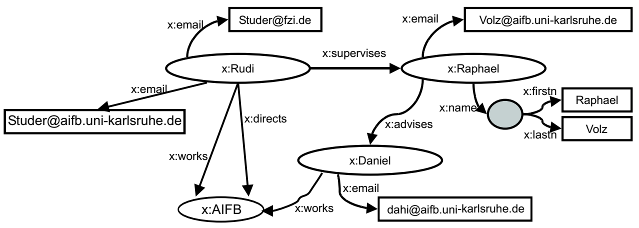


Figure 1. A simple, exemplary RDF model

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. Both are embedded in a class and a property inheritance hierarchy. One of the proposed standards for the Semantic Web is RDF Schema (RDFS)[9][4]

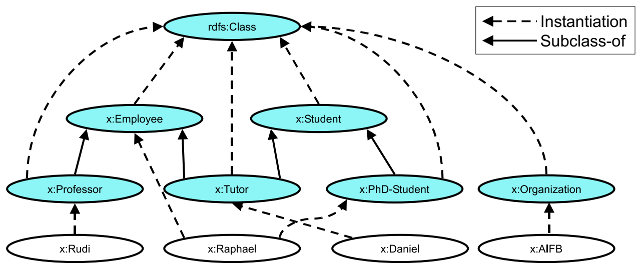


Figure 2. Class hierarchy in RDFS for a simple ontology

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 classes. A given resource can belong to several class extensions since multiple instantiation is allowed.

[1]This can also be omitted creating so-called anonymous resources, e.g. the resource pointed to by 'x:name' in Figure 1

[2]We can speak of pseudographs since multiple edges between (possibly identical) nodes are allowed.

[3]For the sake of brevity, we use "x" to abbreviate a uniform resource identifier (URI) using XML namespaces. In a real world scenario, the "x"es would be replaced by URIs.

[4]A more expressive language in style of description logics is currently finalized by another W3C working group.

Attributes and associations are not defined with the class specification itself. Instead, such class properties are defined as first-class primitives, so-called properties, which exist on their own. Thereby classes do not specify types. Instances may have further (unspecified) properties and may also not use properties that were specified to be valid for their particular classes.

The definition of a property may include the specification of (multiple) domains and ranges. This defines the context, i.e. the class instances, in which a property may be validly used in an RDF statement. Multiple assignments have to be understood conjunctively. Therefore, a property can only be validly used on resources that are instances of all classes simultaneously. For example, in Figure 3 the property 'x:advises' has two domain classes: 'x:Employee' and 'x:PhD-Student', thus 'x:advises' is correctly instantiated in Figure 1 on 'x:Raphael' in Figure 1 (also cf. Figure 2 for class membership). The consistency of the constraints is maintained by entailing the appropriate class membership for resources when they are used in a property instantiation [12].

If the domain or range of a property is not defined, no entailments are made for the resource-value pair, e.g. this applies to 'x:responsible_for' in Figure 3. Properties may be placed into a subsumption hierarchy as well, e.g. in Figure 3 the property 'x:advises' is a specialization of the property 'x:responsible_for'. Property subsumption establishes a partial-order and has set-inclusion semantics as well. A query to the extension of the property 'x:responsible_for' to the data set of Figure 1 would therefore yield the two tuples (('x:Rudi','x:Raphael'), ('x:Raphael','x:Daniel')).

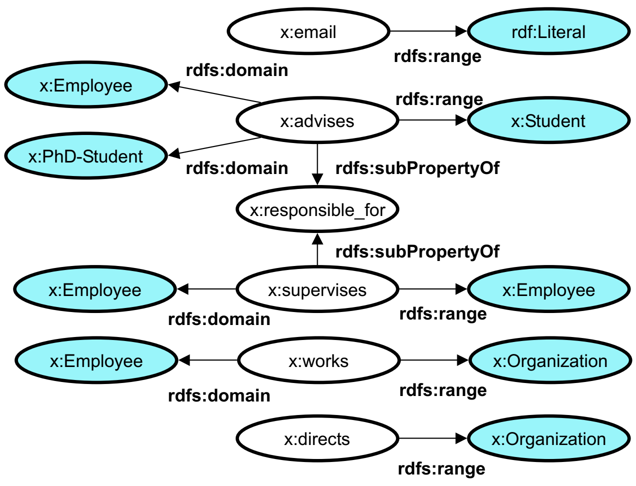


Figure 3. Properties in RDFS for a simple ontology

2.3. Query Language

We chose to extend RQL [3] with view primitives. RQL 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) also fail to meet our goals[5].

Due to lack of space, we can only give a short introduction to RQL in this section. The interested reader may refer to [3] for a more in-depth description.

RQL is a typed language following a functional approach (in style of OQL) and aims at querying RDF at the semantic level. 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 such 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 and is the atom of all path expressions. The variables X and Y are bound to the resources and values of those RDF statements that use the property x:email. The {} notation is used in path expressions to introduce variables.

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. The following query shows these features:

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

This query returns the identifiers of all students advised by other students. Since the class PhD-Student is a subclass of Student the above query would return "x:Daniel" for the RDF model depicted in Figure 1.

Furthermore, RQL supports set operators, such as union, intersection and difference. Boolean operations like $=$, $<$, $>$ can be used for selection in where-clauses.

3. Design of a view language

Views provide a new level of data independence and allow the required selection, customization and integration of data that is required for many Semantic Web applications.

[5]Additionally one would have to take care of the multiple syntactic variants that exist to represent the same data due to the RDF specification.

From the perspective of classical databases it is natural to consider views as arbitrary stored queries. Users should not be able to make a distinction between views and base data and should be able to state other queries or views on top of them. This mandates that the structure of views corresponds to the structure of base data.

However, in classical approaches to views (independent of the particular data model), no conceptual description of views are provided. Hence, the semantics of the view remain unclear to the agent. Since the Semantic Web builds on this very semantics, the semantics of the views must be described by an ontology and views must be embedded in the appropriate location of the inheritance hierarchies. This imposes the following requirements on a view language for ontology-based RDF data:

1. The structure of views must correspond to the structure of data. This results in the distinction between views on classes and views on properties. Hence, views can only involve queries which return either unary (views on classes) or binary (views on properties) tuples. RQL queries that return n-ary relations are therefore not allowed in view definitions.

2. The semantics of views have to be specified by an ontology and should be embedded in the respective inheritance hierarchies according to their semantics by classification. This classification must be part of every view definition.

3.1. Classification of views

Views have to be embedded into the inheritance hierarchies by explicit assignment. This imposes further work on the user. Therefore this burden should be taken from the user by deducing the appropriate classification from analysis of the query.

Unfortunately, this information cannot be deduced in all cases due to the undecidability of the general problem [17, 4]. However, the classification can be deduced automatically for many important queries. In our approach the required (and undecidable) analysis whether a particular query belongs to one of those special cases is avoided by introducing a special convenience syntax for each case. If these syntaxes are used the appropriate classification is automatically generated for the user.

Figure 4 provides an overview about the classification which is semantically correct for each algebraic operator. The illustrated classifications are motivated by the set-inclusion semantics of inheritance.

Selection always reduces the initial set and creates subsets. Therefore the class or property which is subject of the selection subsumes the view.

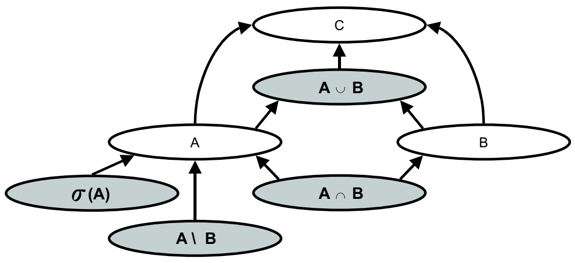


Figure 4. Placement of views in the hierarchy

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

Union unifies the extensions of classes or properties. Hence, the unified classes or properties are subsets of the view. Consequently, the view subsumes the unified classes or properties. The view itself is subsumed by the least common element in the respective inheritance hierarchy of all unified elements.

Intersection is a subset of the extents of all intersected classes or properties. Therefore every intersected class or property subsumes the view based on the intersection operation.

Join, negation and cross product do not introduce sub- or supersets wrt. to the interconnected classes or properties. Hence, the semantics have to be stated manually by the user. One exception are views on classes, where queries have to return unary results. Here, the join operator decomposes to selection[6]. The cross product operator does not have any effect wrt. the row that provides the unary result of the query[7]. Nevertheless the user has to stated even in this case which class or property is subject of the selection.

3.2. View Population and updates

Views that are created via this automatic classification are also updatable, since updates can be delegated to the involved classes or properties.

Since RDF uses object identifiers in form of URIs, updates on views could be propagated to base data. This requires that objects are preserved. Hence, views must be populated with base data. The generation of objects together with new instance identifiers, which is chosen in

[6]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.

[7]Since duplicates are irrelevant with respect to the interpretation of the result since we regard the result as a set of instances

many object-oriented view languages (e.g. [5]), is therefore not applied in our language.

Object preservation allows the propagation of updates in many cases. Views could be updated if they don't involve joins (such as involved in generalized path expressions), or aggregation functions. The question where to put newly generated instances, known as the so-called interface resolution conflict [8], does not arise with RDFS ontologies since multiple instantiation is allowed.

For views based on the above-mentioned operations all updates are therefore propagated to the involved base classes or properties (cf. Figure 5). For example, if object x is deleted from $\cup(A, B)$ then x is deleted from both A and B. Any updates through a view are ultimately visible for views based on the union and intersection operations, since they automatically meet the query conditions afterwards. The latter is not necessarily true for selection-based views, since the updated data may not meet the query conditions anymore.

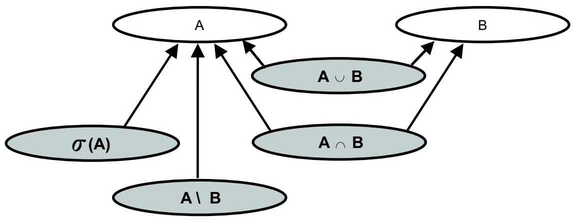


Figure 5. Update propagation

Updates are generally not permitted against elements of the ontology itself. Modifying or deleting classes and subclass relationships may invalidate view definitions, e.g. if a class is deleted. Inserts do not necessarily impose problems since they do not alter existing information. However, for union-based views, where the least common subsuming element is sought, a reclassification could be necessary. The same kind of problems arise with updates on property definitions. This problem is germane to schema evolution and not specific to the Semantic Web.

4. View Language

In this section we will describe the language constructs for creating class and property views as well as for external ontologies.

4.1. Views on classes

The definition of views on classes involves two components: First, users have to define an arbitrary RQL query. This query must return a set of resources, viz. unary tuples. This set of resources constitutes 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. The basic syntax for the definition of class views (cf. [22]) therefore involves these two components. For example, one could characterize the class of all "Wis. Mitarbeiter" consisting of PhD-Students that are employed and advise students at the same time:

```
CREATE CLASS VIEW x:WisMitarbeiter
  SUBCLASSOF x:Employee
  SUBCLASSOF x:PhD-Student
  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 decide where to propagate updates.
2. *Manual Classification* As users can combine arbitrary algebraic operations in the view[8] the semantic characterization of the view cannot be given automatically since this problem is undecidable [17, 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. Consequently, the view language features convenience syntaxes which allow to define views via selection and set operations (i.e. union, intersection and difference). Multiple convenience syntaxes for selection are available, viz. rename, arbitrary selection and difference[9].

Convenience syntaxes are normalized into the above mentioned general syntax. The classification of the view, i.e. the creation of necessary subclassof statements, is automatically generated during the transformation. Besides, the appropriate RQL-query is created. Views which are defined in this way are additionally subject to the previously discussed updatability.

For example a class view "x:Scientist" which consists of professors as well as PhD-Students is created by the following definition:

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

[8]except for the default projection that fixes the arity

[9]As already mentioned, the difference operator can always be rewritten into an equivalent selection.

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 omitted in the translation. The generation of the RQL query which is used in the translated definition is straightforward:

```
CREATE CLASS VIEW x:Scientist USE
(SELECT X FROM x:Professor{X})
UNION
(SELECT X FROM x:PhD-Student{X})
```

The remaining convenience syntaxes can be found in [22].

4.2. Views on Properties

The declaration of views on properties involves the following:

1. an arbitrary, binary query[10]
2. the definition of the views' domains and ranges
3. embedding of the view into the property hierarchy

The following definition creates a new property view which relates all PhD-Students with the email addresses of 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 updatability of views based on arbitrary queries cannot be automatically ensured. Furthermore, the consistency of the view definition with respect to constraints and property inheritance must be ensured at compile time (cf. section 6.4).

As the definition of property views involves quite a lot of information from the user, several convenience syntaxes are supported by our system, simplifying the construction of property views. Similar to views on classes, these short hand notations support automatic classification of the view into the hierarchy. Additionally, appropriate domain and range constraints are generated.

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
```

This can be automatically translated to the following standard property view definition:

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

5. External ontologies

5.1. Motivation

Many typical Semantic Web applications such as community portals are characterized by the fact that they rely on more than one information source and collect information from many distributed sources in the web. Distributed information can be aggregated and combined easily due to the characteristics of the RDF model. The integrated information can be understood if all information providers have used the same ontology to mark up their data. Hence, information that is not presented according to the ontology of the consumer cannot be understood. This mandates a means to transform data such as provided by database views.

Accordingly, views should not only be applied on one data source, but on the integrated data of several data sources instead. This requires the ability to integrate information from several sources. A set of view definitions can then transform data outside of a particular data source.

The aggregated and transformed data is often intended to be republished as a new information artefact. Therefore it is necessary to be able to control which elements should be republished. This requires the adoption of the ontology. For example, the ontology should only talk about the aspects of the data that are visible, e.g. for security reasons.

5.2. Primitives

Figure 6 depicts the features offered by external ontologies. First, users can import classes and properties from multiple RDF databases and other external ontologies. Second, users may state new views on top of the integrated data

[10]Each tuple must be either (resource × resource) or (resource × literal) to reflect that literals cannot be in the domain of RDF properties

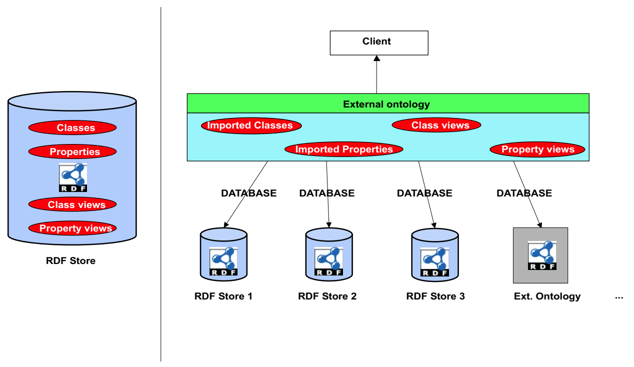


Figure 6. Classical and external ontologies approach

sources. View queries have access to all data not only the imported classes and properties. This allows to transform data without making the (raw) data itself visible to users.

One the one hand, this provides *external schemata* in the sense of the ANSI SPARC three-level architecture for databases, where applications or users can access a database through a specified subschema, e.g. to issue queries. On the other hand, this differs from the ANSI SPARC approach since external ontologies are hosted and specified completely outside of the data sources.

5.3. Example

The following example provides an external ontology that captures an administration perspective for a scientific department. This includes all information about people who actually receive payments. The view x:WorkingStudents grasps that students might employed by another faculty than the one they are enrolled in. The latter information would not be possible within an isolated faculty database.

```
CREATE EXTERNAL ONTOLOGY
x:HumanResources
DATABASE x:Faculty_1
DATABASE x:Faculty_2
IMPORT CLASS x:Scientist
IMPORT PROPERTY x:email
IMPORT PROPERTY x:supervises

CREATE CLASS VIEW x:WorkingStudents
ON x:Student INTERSECT x:Employee
```

The interested reader may refer to the normative syntax defined in [21].

6. Implementation

6.1. Storage of RDF

We use a standard relational database to store the physical RDF data. For each class, a separate unary relation is created, in which the particular instances of the class are stored. Furthermore, separate binary relations are created for each property, all triples[11] are then sorted on a per property basis and stored as (resource, value) pairs in these relations.

Please note, that this approach also creates six dedicated relations that store the ontology. Two of them are unary relations which store all class and property definitions. Besides, there are four binary relations storing the class and property hierarchies as well as the constraints for properties.

In our experience this physical representation outperforms the naive storage of RDF on a per triple basis within a single ternary relation with respect to read access. If updates are dominant for a RDF system the ternary representation is faster since it does not involve the generation and deletion of physical tables in the database.

Additionally the storage of data on a per class and property basis simplifies the integration of RDF data with legacy data. The latter can be done via the definition of unary and binary relational SQL views on top of existing database content. To realize this data access has to be realized via a catalogue component which determines the correct physical relation where property and class data is actually stored.

6.2. Realization of RDF Semantics

In order to speed up query processing all entailments supported by the RDF semantics specification [12] are materialized. This involves primarily the computation of transitive closure of the class and property hierarchies. All implicit members of classes and properties (e.g. instances of subclasses and subproperties) are stored in separate physical relations. The same is done for other entailments, e.g. for domain and range of properties.

This materialization of implicit information has to be maintained in case of updates. Insertions are easy to handle and simply require to compute the new closure resulting from the addition. Modifications and deletions are harder to handle. Trivially, we could simply recompute the whole materialization. More elaborate mechanisms such as implemented in the Sesame system [7] keep track of all the dependencies between statements (statement B was derived from statement A, etc.) and use this knowledge of the dependencies to determine which set of statements have to be

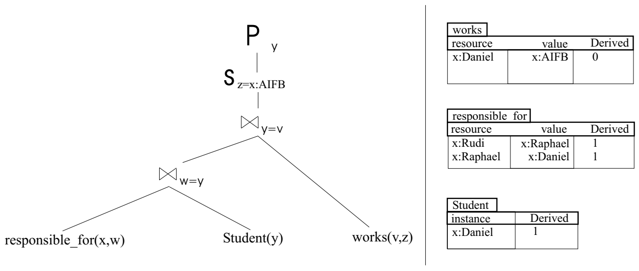
[11]except for those creating the class instance relationship, i.e. (x, rdf:type, <class>)

removed together. Naturally, this makes adding new statements slightly slower (since dependencies have to be computed and stored), but removing statements is a lot quicker.

6.3. Interpretation of queries

The implementation of the query language is straightforward because operations can be mapped easily to those offered by the relational algebra of the underlying database. Hence, operations are pushed down to the database. Consider the following query as an example. It returns all students working at AIFB, who are advised or supervised by someone:

```
SELECT Y
FROM {x} x:responsible_for
{y : Student}.works {z}
WHERE z = "x:AIFB"
```



works

resource	value	Derived
x:Daniel	x:AIFB	0

responsible_for

resource	value	Derived
x:Rudi	x:Raphael	1
x:Raphael	x:Daniel	1

Student

instance	Derived
x:Daniel	1

Figure 7. Query graph and involved physical db relations

First, the parser analyzes the syntax of the query, then the semantics of the query are captured via graph constructions. Figure 7 depicts the query graph of the above query. Then, this graph is translated into corresponding SQL queries, which are sent to the database engine:

```
SELECT responsible_for.value
FROM responsible_for, student, works
WHERE responsible_for.value =
student.instance AND
responsible_for.value =
works.resource AND
works.value = "x:AIFB"
```

During graph construction, the various shortcuts of RQL are expanded and variable dependencies are determined. The computation of the extents of classes and properties would involve the expensive computation of the transitive closure of the respective inheritance hierarchies. To avoid this and speed up read access, the transitive closure is computed with each update and materialized as mentioned before.

6.4. Realization of views

Views are created in a straightforward manner. First the provided convenience syntaxes are normalized into the general forms, creating the appropriate query, classification and constraints. Then the adequate tuples in the subclass-of and subproperty-of relations are created, respectively. For views on properties, additional tuples are created in the appropriate tables that store property constraints. Then, a relational view is created for each view whose query is the SQL query which was translated from the RQL query. Alternatively, we could store the normalized query in a dedicated relation and expand the queries issued by users with view queries in style of [20].

For views on properties additional consistency checks have to ensure that the returned tuples are actually instances of the specified domain and range constraints and that these constraints are compatible with respect to the constraints of super properties. While the latter is checked at compile time (cf. [16]), the first is implemented via a kind of type casting. It ensures that the domain and range can only take values that are in the extent of the specified classes. Hence, the RQL expressions of the following form are appended to the WHERE-clause of the RQL query for all specified domain and range constraints[12] before it is compiled into SQL:

```
AND DOMAIN IN (SELECT M FROM
CONSTRAINT_URI_i{M} )
```

6.5. Realization of External Ontologies

Since many user queries can only be answered by the integrated data, a materialized approach is chosen. Hence, all imported data is replicated and views are materialized. User queries are then processed in the normal manner.

The inheritance hierarchies of the external ontology have to be adopted to those classes and properties and views that are visible to the users. This is done by Algorithm 1 for the class hierarchy[13]. Following Figure 8, all inheritance information and is gathered from all sources and augmented with the information about the classification of views in the first step. Then, the full transitive closure of this merged inheritance hierarchy is computed. In the final, third step only the links that connect visible nodes remain.

The property hierarchy is computed in a similar fashion. Furthermore, the domain and range constraints for properties must be adopted to the visible classes, cf. [16] for the proposed solution.

[12] with the exception of literal ranges

[13] This algorithm is simplified, since the implementation can avoid unnecessary computation of the transitive close and has only to consider all upwards links from visible nodes, hence the algorithm can bring visible nodes into a topological order and do an incremental computation (unlike the presented algorithm)

Algorithm 1 Computation of the class hierarchy

```
Require: $IC$ set of imported classes, $CV$ set of class views, $S$ set of RDF sources
  $subclassof = \{\}$
  for all $s \in S$ do
    $subclassof = subclassof \cup s.subclassof$
  end for
  for all $v \in CV$ do
    $subclassof = subclassof \cup v.subclassof$
  end for
  $subclassof = subclassof^*$
  $newsubclassof = \{\}$
  for all $c_1 \in (IC \cup CV)$ do
    for all $c_2 \in (IC \cup CV)$ do
      if $(c_1, c_2) \in subclassof$ then
        $newsubclassof = newsubclassof \cup (c_1, c_2)$
      end if
    end for
  end for
Ensure: new class hierarchy ($newsubclassof$)
```

6.6. System Environment

The view language is currently implemented as an extension of the open-source RQL database system Sesame [7]. PostgreSQL is chosen as the relational database. Several technical details like the employed indexing techniques and translation of class and property URIs to the physically used names for tables have not been presented here.

A first implementation, described in [16] provides an in-memory variant of an earlier version of the view language and has been realized with the SiLRI F-Logic engine [10].

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 the approach to views taken for the relational data model.

Also, we cannot rely on previous work about views for other semi-structured data models. The views of [23] consist of object collections only. Associations between objects - which are fundamental to RDF - are not considered. [2] do consider such edges, but do not provide semantic descriptions for views. Nevertheless our proposal is the only one that takes a superimposed conceptual model into account, viz. the ontology. Besides, 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, 17, 18]). We have combined many aspects presented there. The proposition of external

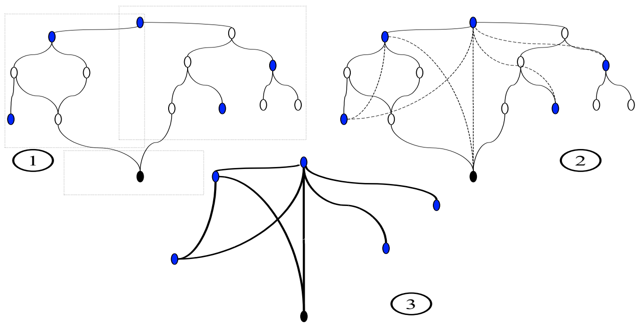


Figure 8. Steps in the computation of the class hierarchy. (1) merging all subclassof statements from all source databases, (2) computation of the transitive closure, (3) Keeping only links between visible classes

ontologies and the terminology is similar to [8]. The idea to classify views in the hierarchy was first proposed in [18]. Other approaches, e.g. [5], do not mix view and class hierarchies. Like the majority of object-oriented approaches to views we mainly support object-preserving views. Many object-oriented approaches 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 was also proposed in [1, 18]. However, views can only be introduced in this context and not alternatively be added to the base database.

Generally those approaches do not fit an open world such as the Semantic Web. This is mainly due to the explicit typing of classes and local assignment of properties to classes taken in object-oriented databases. Web criteria such as the ability to base views on multiple data sources are not met either. Also property hierarchies are unknown to object-oriented models.

8. Discussion

We have presented a view mechanism that picks up the unique situation of data in the Semantic Web and implementation issues surrounding this mechanism. 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 [15].

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 [14, 19] 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 metadata 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. Currently, we are extending our view language towards a more expressive Web ontology language, DLP [11], which represents the intersection of Datalog and Description Logics. Additionally, we are investigating how updates can be consistently integrated for this extended Web ontology language. Furthermore, the materialization of views is of great importance in Web scenarios where read access to data is dominant, 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.

Acknowledgements We thank Steffen Staab for discussion and work on previous versions of the paper. This work was funded by the EU in the WonderWeb project (IST-2001-33052).

References

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

[2] S. Abiteboul, R. Goldman, J. McHugh, V. Vassalos, and Y. Zhuge. Views for semistructured data. In *Proc. of the Workshop on Management of Semistructured Data, Tucson, Arizona*, May 1997.

[3] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, K. Tolle, B. Amann, I. Fundulaki, M. Scholl, and A.-M. 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] E. Bertino. A View Mechanism for Object-Oriented Databases. In *Proc. of Intl. Conf. on Extending Database Technology (EDBT)*, pages 136–151, March 1992.

[6] E. 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] J. Broekstra, A. Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proc. of 1st Int. Conf. on the Semantic Web (ISWC)*, Sardinia, Italy, 2002.

[8] C. D. C. S. Dos Santos and S. Abiteboul. Virtual schemas and bases. In *Proc. Extending Database Technology (EDBT)*, 1994.

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

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

[11] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logics. In *Proc. of WWW-2003*, Budapest, Hungary, 05 2003.

[12] P. Hayes. RDF Semantics. http://www.w3.org/TR/rdf-mt/, January 2003.

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

[14] A. Maedche, S. Staab, R. Studer, Y. Sure, and R. Volz. Seal tying up information integration and web site management by ontologies. In *IEEE Data Engineering Bulletin*, volume 25, March 2002.

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

[16] D. Oberle and R. Volz. Implementation of a view mechanism for ontology-based metadata. Technical Report 422, University of Karlsruhe (TH), 2002. http://www.aifb.uni-karlsruhe.de/WBS/dob/pubs/KAON-Views.pdf.

[17] E. 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.

[18] M. H. Scholl, C. Laasch, and M. Tresch. Views in Object-Oriented Databases. In *Proc. 2nd Workshop on Foundations of Models and Languages of Data and Objects*, pages 37–58, Sept. 1990.

[19] 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 - Proc. of the 9th International World Wide Web Conference, Amsterdam, The Netherlands, May, 15-19, 2000*. Elsevier, 2000.

[20] M. Stonebraker. Implementation of integrity constraints and views by query modification. In *Proc. of Int. Conf. on Management of Data (SIGMOD)*, pages 65–78. ACM, 1975.

[21] R. Volz. External ontologies in the semantic web. In *Proc. of the 20th British National Conference on Databases (BNCOD)*, Coventry, UK, July 2003.

[22] R. Volz, D. Oberle, and R. Studer. Views for light-weight web ontologies. In *Proc. of ACM Symposium of Applied Computing (SAC)*, Melbourne, Florida, USA, 03 2003.

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