Towards Views in the Semantic Web

Raphael Volz, Daniel Oberle, Rudi Studer

Institute AIFB
University of Karlsruhe (TH)
D-76128 Karlsruhe
Germany
{lastname}@aifb.uni-karlsruhe.de

Abstract. The Semantic Web is an extension of the current Web in which information is given well defined meaning, better enabling computers and people to work in cooperation. The technical foundation of the Semantic Web is RDF, a semi-structured data model resembling directed labelled graphs. Semantics of data are given by ontologies that are encoded using a specialized vocabulary called RDF Schema. First efforts in actually implementing Semantic Web applications with RDF show that views on RDF data can be very beneficial to gain the intended semantic interoperability allowing RDF data to be filtered and restructured. This paper carefully analyzes the situation constituted by the data model, comparing RDF Schema with well known object-oriented data models. It introduces a specialized view mechanism for ontology based RDF models.

1 Introduction

The Semantic Web is one of today's hot topics and is about bringing ”[...] structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users.” [BL00]. To enable the Semantic Web, pages are supplemented with semi-structured meta-data which provide formal semantics for web content. Formal semantics are given by referring to an ontology, which provides shared domain models, understandable to both human and machine by providing a shared conceptualization of a specific domain.

The technological basis for representing data in the Semantic Web is RDF [LS99], a semi-structured data format that resembles directed labelled pseudographs and exhibits similar properties as OEM [AQM$^+$97]. Usually no schema for this data is given. Instead, ontologies specify the meaning of data and are represented in RDF, using a special vocabulary whose interpretation defines the semantics. This vocabulary, called RDF Schema [DR00], specifies primitives that are similar to primitives known for representing structural aspects in object-oriented data models [1]. Thus, the Semantic Web presents a novel situation, where object-orientation fuses with semi-structured data, i.e. data that exposes non-strict typing and inheritance semantics at the same time.

The aim of the Semantic Web is to create the largest integrated information system ever built. View mechanisms are a sine qua non on the way to reach this objective, as tasks that need views like simplification of large data sets, restructuring data to comply

[1] Unlike object-orientation, RDF Schema features a property-centric approach.

to other ontologies[2], short hands for queries or means for data hiding and security will regularly occur.

Views are an established technology for both relational and object-oriented databases. They are mainly used to provide data customization, viz. the adaptation of content to meet the demands of specific applications and users. Hence, they present the key technology for integrating heterogeneous and distributed systems, facilitating interoperability by hiding the foibles of each information component and gluing individual components together to form an integrated application system. The first aspect is tackled by wrappers, that lift selected content of individual information sources to a common, usually semi-structured, data model. The latter part is done by mediators [Wie93] that provide the glue. In a sense, both mediators and wrappers form views on the data found in one or more sources. However, data does not exist at the mediator, but one may query the mediator as if it were stored data. From an information integration perspective, the Semantic Web makes wrappers obsolete and ontologies simplify the job of mediators by defining an integrated semantic model that gives an explicit representation of the semantics of information components.

This paper picks up the unique situation constituted by the RDF data model and presents a novel approach to specify a view mechanism tailored for the Semantic Web. It is structured as follows. Section 2 details the data model employed in the Semantic Web. Section 3 elaborates the preconditions for a view mechanism by evaluating the approaches taken for classical object-oriented and relational data models with respect to the specific characteristics of the Semantic Web. The query language underlying our approach is introduced in Section 4. Following that, section 5 provides a detailed description of the main features of our view mechanism and outlines how this approach meets the requirements as introduced in Section 3. Remarks on classification and consistency can be found in Section 6. In Section 7, we give a short description of the implementation of our approach. We conclude recapitulating our contribution and provide an outlook to ongoing and future work.

2 The Semantic Web

In the Semantic Web architecture two layers play a fundamental role with respect to capturing semantics: the Resource Description Framework (RDF) [LS99] and ontologies [Gru93]. RDF is used for representing data in the Semantic Web. In essence, RDF is a semi-structured data model that resembles directed labelled pseudographs. In the context of the Semantic Web, XML just plays the role of a syntax carrier that is used to encode and exchange RDF models.

Since RDF describes information by using graphs and has built-in means for linking information coming from different web sources, It inherently meets the ”Web” requirement, i.e. the linking of data should be possible.

RDF does not provide any semantic schema information. Here, ontologies come into play that define the semantics of the represented data by means of classes and properties. Both are embedded into an inheritance hierarchy. Notably, ontologies do not specify a type system for the associated RDF data.

[2] As there will be more than a single ontology.

2.1 RDF - graph-like description of data

Initially, the Resource Description Framework (RDF) [LS99] was intended to enable encoding, exchange and reuse of structured (meta)data describing web accessible resources. Data is encoded using so-called resource-property-value triples, which are also called statements.

In RDF, individual information objects are represented by using sets of statements describing the same resource. Object identity is given via the uniform resource identifier (URI) that labels the resource [3]. This object identifier is globally unique.

A set of statements constitutes a partially labelled directed pseudograph[4] and is commonly called an RDF model. The fact that properties can have multiple values, e.g. 'x:email'[5] 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.

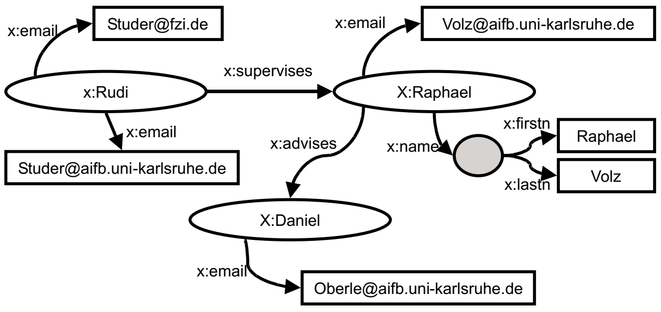

Fig. 1. A simple, exemplary RDF model

2.2 Ontologies

Ontologies provide a formal and shared conceptualization of a particular domain of interest [Gru93]. In the Semantic Web, ontologies are defined using a specialized RDF *vocabulary*[6]. From an implementation perspective this is a nice feature since the same

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

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

[5] For the sake of brevity, we write "x" to abbreviate a uniform resource identifier (URI) using XML namespaces.

[6] and predefined statements, whose interpretation give an ontology definition, for example (uri, rdf:type, rdfs:Class) expresses that the resource uri is a class.

data structures can be used for data and their associated conceptual model, i.e. ontologies. Several languages for representing ontologies in the Semantic Web are currently proposed. They mainly differ in the expressivity of the language constructs. In this paper we use the officially recommended RDF Schema [DR00] vocabulary for ontologies. Unlike OEM where semantic information is supposed to be included in the labels, which are supposed to be self-describing for humans, the semantics of data are explicitly given here. The reader may note, that ontologies do not enforce structure on RDF data.

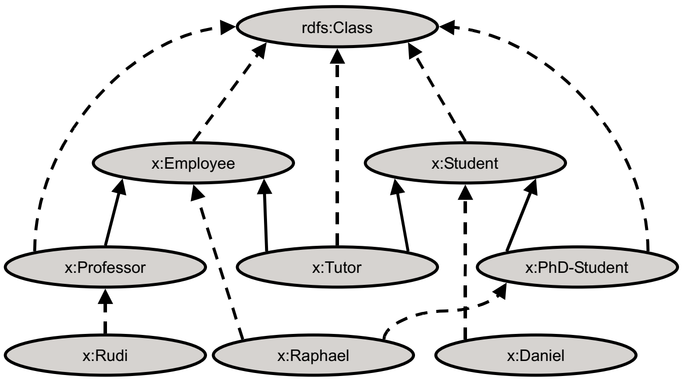

Fig. 2. Class hierarchy in RDFS for a simple ontology.

RDF Schema RDFS incorporates a unique notion of object orientation. It introduces classes and a subsumption hierarchy on classes (compare figure 2, where dashed lines denote instantiation and solid lines denote subsumption). 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.

Unlike commonly-known object-oriented data models, 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. The specification of a property defines the context, i.e. the resource-value-pairs, in which a property may be validly used within an RDF statement. The definition of a property may include the specification of (multiple) domains and ranges. In figure 3 the property 'x:advises' has two domain classes: 'x:Professor' and 'x:PhD-Student', thus 'x:advises' is correctly applied in the statement ('x:Raphael','x:advises','x:Daniel') of figure 1.

If the domain or range of a property is not defined, the instantiation of its domain or range may occur to any resource-value pair, e.g. this applies to 'x:responsible_for' in figure 3.

Properties may be placed into a subsumption hierarchy as well. In figure 3, the property 'x:advises' is a specialization of the property 'x:responsible_for'. Property subsumption establishes a partial-order and features set-inclusion semantics, too. A query for the extension of the property 'x:responsible_for' in the model of figure 1 would therefore yield the two tuples (('x:Rudi','x:Raphael'), ('x:Raphael','x:Daniel')).

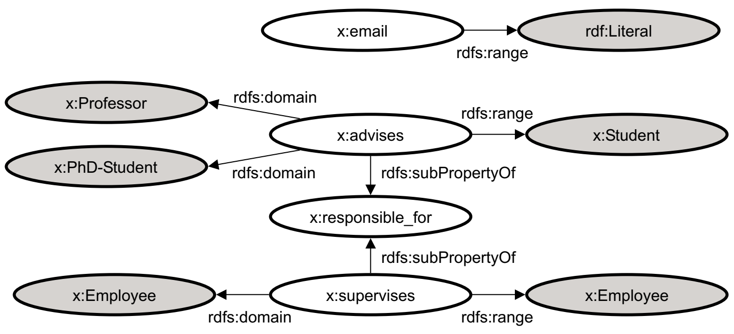

Fig. 3. Properties in RDFS for a simple ontology

RDF Schema (Fixed Architecture) Unfortunately, the RDF Schema specification does not provide formal semantics. This issue lead to a series of alternative formal specifications such as [Hay01] or [PH01]. The latter proposed a minor modification of RDF and RDFS introducing a stratified four-layer meta-model architecture such as employed by UML [Gro01]. This cleans up some imprecise definitions of RDF and RDFS[7]. Therefore, our following considerations will be based on RDFS(FA). The reader may note, that this does not restrict the generality of our approach besides disallowing infinite layers of classes.

Implementation issues are tremendously simplified by utilizing the fixed-layered meta-modelling architecture. It is a proven methodology for defining the structure of complex models that need to be reliably stored, shared, manipulated and exchanged [Kob99].

3 Design of a view mechanism

This section provides our design goals for an ontology-based view mechanism and studies the alternatives that are given for each design dimension by careful analysis of the situation constituted by the data model and the novel situation of giving semantics without specifying a type system. Along this way, we report on the related work in object-oriented views.

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.

[7] Such as rdfs:class being an instance of itself which is close to Russell's paradox.

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 semi-structured data models. The views of [ZGM98] consist of object collections only, associations between objects - which are fundamental to RDF - are not considered. [AGM$^+$97] do consider such edges, but do not meet the first and second 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 subclass and subproperty relation.

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. It also allows that views can be composed of each other. The user can choose arbitrary (previously unused) identifiers for his views[8].

3.1 Design goals

Besides the constraints constituted by the particular underlying data model, it is mainly the goals of a particular view mechanism that motivate a particular design. We want to support the following:

1. The most important goal with respect to our approach is that a view should again be based on an ontology. In particular, the operations manipulating the underlying data should be based on ontological primitives. Also, the semantics of each transformation should be properly reflected with respect to the semantics of ontological primitives.
2. Collection of views should be possible in order to form external schemata as proposed by the three-level ANSI SPARC database architecture. As we do not provide schemas we speak about external ontologies instead.
3. Our approach should be functional, that is it should be allowed to create views based on other view(s). This is already fulfilled (see above).
4. Additionally it should be Web-aware allowing to create views on a set of source RDF models.

3.2 Design considerations

Several approaches have been proposed for object-oriented views. The approaches differ not only in data models and the used query languages but also in the set of functionalities supported by the view mechanism. We follow the dimensions established in [GBCGM97]:

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

Support for external schemata in the sense of the ANSI three-level architecture for databases, where programs have to be allowed to access the database through subschemas. As this is one of our design goals, we support this feature in the sense of [GBCGM97,Run92,SLT90,CSDSA94]. We refer to class and property views. The classes and properties coming from the underlying RDF model are called base classes and base properties respectively.

Placement of view elements in the hierarchy The next consideration we have to decide upon on is where to place views in their respective hierarchies. This can happen in two ways. On the one hand, it must be done by 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. As the semantics of an operation that defines a view should be properly represented, this relation should be established automatically. Along with consistency, those issues are discussed in section 6.

View population A further decision we have take is how to populate the views. In principle three alternatives exist for object-oriented views:

1. *object preservation*, where view classes are populated with instances of base classes
2. *object generation*, where views are populated with new instances
3. *set tuples*, where views return sets of tuples akin to the relational world.

A look at the related work on object-oriented views reveals that many approaches support more than one alternative. For instance, [AB91] introduces virtual classes that show instances of base classes, thus allowing object-preserving views (like the majority of OO view approaches). Additionally, imaginary classes are allowed and are populated by tuples defined by query results for which new object ids are computed. [Ber92] suggest to allow object-generating views motivated by the goal to support schema evolution via a view mechanism, which also requires the computation of object identifiers.

As object-generation is a useful feature, e.g. views defined via join operation require object-generation and the RDF data model allows for anonymous resources obviating the need of oid generation, we support this kind of views. We also allow for object-preserving views. Of course, object identifiers are only preserved if they were given in the source model. Notably, for object-preserving views the interface resolution conflict mentioned in [CSDSA94] does not arise as RDF supports multiple instantiation.

Updates on views Whether or not data can be updated is another important criterion for a view mechanism. In fact, view update is a commonly-known problem for relational databases, as views introduce a potential ambiguity in this context, i.e. if they are defined via operations like a relational join. For object-oriented view mechanisms this problem is somewhat simpler due to the fact that ambiguity is avoided with OIDs. Thus, our view mechanism is principally updatable for object-preserving views, if object identifiers were present in the source database. Our implementation (cf. chapter 7) cannot handle updates yet, as dynamic reclassification of objects is required if objects can change state. The fact that ontology and data are encoded in the same model complicates this task tremendously as we would have to check whether the view definitions themselves are still valid. This problem is similar to the problem of maintaining view definitions if database schemas change. We plan to consider updatable views as a next step.

4 Query Language

For our view definition language we rely on a subset of RQL [ACK$^+$00], which is a declarative language for querying RDF. It is a typed language and follows 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. RQL is able to switch between schema and data querying while exploiting class and property hierarchies.

Our choice of RQL is motivated by the fact that it 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 [BCF$^+$02]) fail to meet our goals. Due to lack of space, we can only give a short introduction to RQL in this section. The interested reader may refer to [ACK$^+$00] for a more in-depth description.

The following query gives a gentle introduction to RQL and returns the collection of all resource-value pairs which are related via the property email:

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

RQL queries follow the basic select-from-where construct as known from SQL. The construct {X}x:email{Y} is called a basic data path expression, the atom of all path expressions. In the above example the variables X and Y are bound to the resources and values of 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. 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 $=$, $<$, $>$ for pattern matching can be used for selection in where-clauses.

5 View Definition Language

This sections talks about our view definition language, i.e. the means to define class views, property views and external ontologies. It resembles SQL syntax and takes RQL as a query language.

5.1 Views on classes

The definition of views on classes involves two components. First, users must define arbitrary RQL queries which return unary tuples of resources. These tuples constitute the instances that are in the extent. Second, the view must be properly classified in the class hierarchy. This enables the understandability of the view wrt. to the set-inclusion semantics of the subclass relationship. The basic syntax for the definition of class views (cf. Figure 4) therefore involves these two components.

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

Fig. 4. Definition framework for views on classes

For example, one could characterize the class of all ”Wissenschaftlicher 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
    )
```

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

Selection views One convenience operation provided by the system is selection, where a subset of the members of a certain class is selected.

```
CREATE CLASS VIEW <URI> ON <BASE_URI>
    USE
    SELECT INSTANCE
    FROM ...
    [ WHERE ... ]
```

The user is still able to pose arbitray RQL queries. The system ensures that the result of the query is a selection on the base class (cf. chapter 6). The following definition creates a new class ”x:Supervisors”, which is populated with 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 is created 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)
```

Conforming to the classification which will be discussed in section 6, the view is made a subclass of x:Employee. Additionally, the above mentioned type casting expression was appended to the where clause.

Difference views We provide the following convenience syntax:

```
CREATE CLASS VIEW <VIEW_URI>
    ON <URI> MINUS <URI>
```

The following definition would create a class view ”x:Unemployed-Students” which is populated with all students that have no job.

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

The translation of the convenience syntax to the standard view definition involves the generation of the subclassof statement which corresponds to the minuend. Additionally, the appropriate RQL query must be generated[9].

Union view We define the following syntax for creation of views via unions of class extents:

```
CREATE CLASS VIEW <VIEW_URI>
    ON <URI> UNION <URI> { UNION <URI> }
```

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

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

```
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 subclassof statement is omitted in the translation. Consequently only two properties, namely x:email and x:responsible_for, can be validly applied to x:Scientist wrt. to updates. Since walking generalized path expressions in RQL involves implicit joins, all other properties defined for the instances in the extent of the view remain visible when the view participates in queries. The generation of the RQL query which is used in the translated definition is straightforward[10].

Intersection view For the intersection operator the following convenience syntax is defined:

```
CREATE CLASS VIEW <VIEW_URI>
    ON <URI> INTERSECT <URI>
```

The following example defines a class view ”workingstudents” as being those students who do have a job:

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

Now the translation of the convenience syntax to a standard definition involves the generation of at least two subclassof statements as well as an RQL query.

5.2 Views on Properties

Property views can be defined using arbitrary query expressions. As mentioned before, these queries must return binary tuples[11]. Besides the query itself, several additional information is required to define a view on properties. As depicted in figure 5 this involves

1. the definition the domains and ranges of the view
2. sorting the view into the property hierarchy
3. forcing the query to return binary tuples

There are also convenience definitions for property views like there were for class views. Those comprise selection, difference, intersection and union. Besides, one can use convenience definitions to rename and refine properties. Due to the lack of space and the similarity to class views, we won’t go into more detail here.

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

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

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

Fig. 5. Definition framework for views on properties

5.3 External ontology

The basic principles mentioned above ensure that views can fulfill their traditional role of providing a new abstraction level allowing the customization of information and (possibly) access management for the Semantic Web. While this is sufficient within one application, several additional goals must be met from a web perspective to ensure interoperability of content. Here, content is not understood as a basic object but rather as a set of basic objects which are published together in one document.

The formation of new sets of objects is therefore central to making content in the Semantic Web interoperable. Our view definition language additionally supports such document formations.

Thus, collections of view definitions, imported classes and imported properties from the base RDF model form a new *external ontology*. 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 allows a customized and transformed serialization of an RDF model.

In this serialization the extents of all classes and properties are materialized. The serialization involves modification of the ontology to meet access restriction requirements. Here, only those parts that are required for the understanding of the external ontology are made visible in the serialization[12]. This involves the reduction and adoption of both class and property hierarchies (cf. section 6).

Since content is often not situated in one RDF model but distributed in several RDF models it is allowed to aggregate several RDF models in one external ontology and do a transformation of this data by means of views in one (virtual) model. This is an important feature for community portals that want to expose their data to other content consumers such as characterized in the introduction[13].

Figure 6 presents the basic notation for the definition of an external ontology. It is allowed to refer to data given in several source databases. The user can import classes and properties from those source databases. This import makes the direct extents of

[12] One might argue that the materialization of the extents is sufficient but very often the conceptualization provided by ontologies presents an important value itself. For example, many companies have internal thesauri which should not be published freely.

[13] For example, the OntoWeb community portal (http://www.ontoweb.org/) collects distributed information from its community members.

```
CREATE EXTERNAL ONTOLOGY <URI>
 DATABASE <MODEL_URI> {, <MODEL_URI>}
IMPORT CLASS <URI> {, <URI> }
IMPORT PROPERTY <URI> {, <URI> }
{ Additional view definitions }
```

Fig. 6. BNF-like notation for the definition of an external ontology

those classes and properties visible in the given external ontology. This does not refer to the complete extents (which would include instances and resource-value pairs of subclasses and subproperties) to enable a fine-grained control of the uncovering of extents. Naturally import operations have object-preserving semantics, thus statements are updateable if the data is managed within the system. Updates can be propagated easily to the underlying source databases. The reader may note that the import operation cannot distinguish between base classes and properties and their view siblings which might have been created in the source databases.

To allow customization of this data, additional views can be defined within this external ontology. The queries within these views do have unrestricted access to data. This means that not only the imported data is visible but the complete set of data contained in all included source databases to those queries. Truly virtual databases are established by such external ontologies.

The following example provides an external ontology that captures an administration perspective for an scientific department. This includes all information about people who actually receive payments. Such information could be extracted from the institute's web site[14] and is intended to be republished to the human resource department of the university:

```
CREATE EXTERNAL ONTOLOGY x:HumanResources
 DATABASE x:AIFBNet
IMPORT CLASS x:Employee, x:Scientists, x:Working_Students
IMPORT PROPERTY x:email, x:supervises
```

6 Classification and Consistency

6.1 Automatic hierarchy classification

As users can combine arbitrary algebraic operations in the view [15], the semantic characterization of the view cannot be given automatically because this problem is undecidable [Run92,Bee89]. We therefore introduced additional possibilities to conveniently define views on classes where the classification can automatically be determined from query semantics (cf. section 5.1).

Figure 7 provides an overview of the classification which is semantically correct for each algebraic operation. Since class views can only be defined via unary queries

[14] for example: www.aifb.uni-karlsruhe.de runs on Semantic Web technology
[15] except for the fixed projection

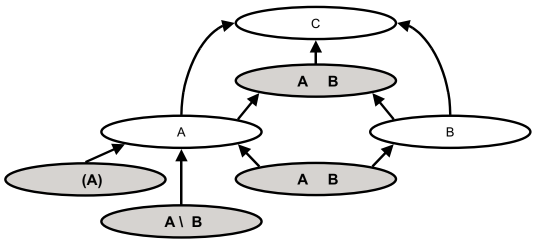

Fig. 7. Placement of views in the hierarchy

operations 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 queries. 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.

The semantics of the depicted operations are motivated by the set-inclusion semantics of inheritance in RDF: Selection always reduces the initial set and creates subsets. If a query is defined via a selection, the basis of the query 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 extensions of the unified classes are subsets of the view. Therefore, the view subsumes the unified classes. The view itself is subsumed by the least common superclass of all unified classes. For the intersection operator the opposite fact holds. The intersection is a subset of the extents of all intersected classes. Hence, every intersected class subsumes a view based on the intersection operation.

Class views that are created via these algebraic operators are updatable. All updates can be propagated to the underlying instances since instances are identified by their URI. Insertions of new resources in the class views are always propagated to insertions of the respective superclasses of the view. Thus adding a new instance to a view based on intersection physically adds the new object to all of its classes that are subject of intersection. This is also true for adding a new instance to view based on selection or difference operators. No new objects can be added to union-based views since the adding of the instance to the superclass of the view would not meet the query condition and would become invisible to the view.

Regarding our stance of classification, we consider Description Logics [FB90] as related work here. One advantage of Description Logics is the hierarchical classification of concepts which resembles the subclass-relation in our work. This so-called subsumption is automatically computed by regarding the subset-relation among concepts' extensions. Views on classes however, are implicitly classified by the ON-clause or by manually assigning a super-class in our approach.

6.2 Ensuring the consistency

Class views (selection) The system ensures that the result of the query is a selection on the base class. This is done by appending the following expressions to the where-clause in figure 4 :

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

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.

Since the query can involve arbitrary joins and aggregation, the updateability of such a general property cannot be automatically ensured. Additionally, the consistency of the view definition must be ensured at compile time.

Consistency of a property's domains and ranges First, it must be ensured that only tuples are returned that meet the constraints specified by the domain and range definition. This is a kind of type cast which implemented by ensuring that the variables DOMAIN and RANGE can take values that are in the extent of the specified domains and ranges of the view (cf. figure 5). Technically, this is done by appending the following expressions to the where-clause of the query:

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

This is done for all specified domains. The adoption for ranges is straightforward. For ranges, an exception is made for properties whose range is specified to be basic literals. This automatism motivates why the variable names which are allowed to occur in the projection part of the query are fixed.

Property hierarchy If we establish a specialization of the property we must ensure that the values are also allowed for its super properties. Thus, the domain and range definitions of the property 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. [OV02]). Since RDF Literals are used to represent all kinds of datatypes, the semantic consistency with respect to datatypes can not be guaranteed. For example, numbers might fill the range of the property whereas strings might be used in super properties.

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 as no domain constraints have been specified for the super property x:email. The range is the same therefore no conflicts arise from that. The query is modified to:

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

This is due to the required casting mentioned above. Here the exception for casting the range applies since range is a literal. The domain is ensured to be in the extent of x:PhD-Student by the appended where clause. The reader may note that the created property cannot be automatically classified into the property hierarchy, as a semantically correct classification cannot be given automatically.

7 Implementation

We give a brief survey of our implementation effort. The view mechanism is currently implemented as part of KAON Server, a multi-user capable, transactional RDF repository. KAON Server is part of the open-source KAON tool suite[16].

In order to provide a query language for the view mechanism, we implemented a large subset of RQL by translating queries into appropriate logic programs, whereas the original RQL definition is compiled into SQL3 to provide an implementation of the query language. Our logic programs were operationalized using the SiLRI [DBSA98] inference engine. Notably, this approach for querying RDF is quite common (cf. Triple [SD01], Metalog [MS98]). The usual approach taken here is the translation of RDF models into simple ternary predicates[17] and to query RDF using logic programming approaches of different expressivity. Logic programming allowed us to provide a clean implementation of the aforementioned consistency rules utilizing the inference engine to operationalize the complete view mechanism. The view definitions themselves do not rely on the full expressivity provided by SiLRI, rather the consistency rules require a highly expressive language.

Specifications for external ontologies are stored like normal RDF data. This is due to the fact, that specifications for external ontologies can easily be mapped into RDF using a specialized vocabulary that defines the required primitives.

Besides querying, consistency check and several inference processes, the question remains how the view mechanism is finally implemented. Clients are able to use the object-oriented KAON-API built upon the W3C's RDF API to access ontologies and corresponding instances. This API finally implements the transparent access to class and property views. The extension and values defined for views are computed at runtime. As soon as an external ontology is loaded, the consistency check and inference processes are started. We transform the external ontology into logic which serves as input to SiLRI. Inferred inconsistency then would lead to a denial of service.

SiLRI is also used to implement the query engine. Therefore the agreggation functions which were specified for RQL are not implemented yet. The inference engine can

[16] http://kaon.semanticweb.org/
[17] statement(subject, predicate, object)

also be used to materialize external ontologies. Currently, they are transiently materialized as snapshots.

Our prototype does not support updates yet. Thus external ontologies and views remain read-only. This motivates the aforementioned materialization strategy for external ontologies which makes sense from a web perspective where agents access collections of objects (contained in one document) via standard protocols and not individual objects such as in queries.

8 Conclusion

In recent years the Semantic Web has evolved to an important research and development topic. Ontologies are widely seen as a crucial part of the Semantic Web by providing precisely defined semantics for the metadata that are associated with Web sources. However, developers of ontologies are not able to envision all kinds of uses in advance. Thus, a view mechanism is required that allows to build application specific views to these ontologies and associated metadata.

Nowadays, RDF and RDF Schema are considered as the representational basis for metadata and ontologies in the Semantic Web. In order to meet the specific requirements of Semantic Web applications, RDF/RDF Schema exhibit characteristics that differ in important aspects from object oriented data models. Therefore, we have proposed a view mechanism that takes these specific aspects into account, notably the absence of typing, the handling of properties as first class primitives, and the specific notion of object identity that is provided by RDF in the decentralized Semantic Web context. Our solution provides a full-fledged view approach that is tailored to these requirements.

The approach as described in this paper does not yet include strategies for materialization. Such materialization strategies are required for addressing scalability issues of the Semantic Web. We are currently in the process of defining such strategies. Furthermore, our approach does not yet support view updates. However, this is a less urgent requirement since many Semantic Web applications will rather exploit a read access to views than updates.

From our point of view, a view mechanism is an important step in putting the idea of the Semantic Web into practice. In the future, our KAON Semantic Web infrastructure will be gradually extended to include these additional aspects of a view mechanism for the Semantic Web.

References

[AB91] Serge Abiteboul and Anthony Bonner. Objects and Views. In *Proc. Intl. Conf. on Management of Data*, pages 238–247. ACM SIGMOD, May 1991.

[ACK+00] 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.

[AGM+97] 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.

[AQM+97] S. Abitebou, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semistructered data. *Int. Journal on Digital Libraries*, 1997.

[BCF+02] Scott Boag, Don Chamberlin, Mary F. Fernandez, Daniela Florescu, Jonathan Robie, Jerome Simeon, and Mugur Stefanescu. Xquery 1.0: An xml query language. Technical report, W3C, April 2002.

[Bee89] C. Beeri. Formal Models for object oriented databases. In *Proc. 1st Intl. Conf. on Deductive and object-oriented databases*, pages 370–396, 1989.

[Ber92] 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.

[BL00] Tim Berners-Lee. Xml 2000 - semantic web talk. Internet: http://www.w3.org/2000/Talks/1206-xml2k-tbö/slide10-0.html, december 2000.

[CSDSA94] C. Delobel C. S. Dos Santos and S. Abiteboul. Virtual schemas and bases. In *Proc. Extending Database Technology (EDBT)*, 1994.

[DBSA98] S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for RDF. In *QL98 - Query Languages Workshop*, December 1998.

[DR00] 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.

[FB90] H.-J. Buerckert F. Baader. Terminological knowledge representation: A proposal for a terminological logic. DFKI Technical Memo TM-90-04, Deutsches Forschungszentrum fuer Kuenstliche Intelligenz, Kaiserslautern, 1990.

[GBCGM97] Giovanna Guerrini, Elisa Bertino, Barbara Catania, and Jesus Garcia-Molina. A formal model of views for object-oriented database systems. *Theory and Practice of Object Systems*, 3(3):157–183, 1997.

[Gro01] Object Management Group. Unified modeling language (uml), version 1.4. Internet: http://www.omg.org/technology/documents/formal/uml.htm, 2001.

[Gru93] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 6(2):199–221, 1993.

[Hay01] Pat Hayes. RDF Model Theory. Internet: http://www.w3.org/TR/2001/WD-rdf-mt-20010925/, September 2001.

[Kob99] Cris Kobryn. Uml 2001: A standardization odyssey. In *Communications of the ACM, Vol.42, No. 10*, October 1999.

[LS99] O. Lassila and R. Swick. Resource description framework (RDF) model and syntax specification. Internet: http://www.w3.org/TR/REC-rdf-syntax/, 1999.

[MS98] Massimo Marchiori and Janne Saarela. Query + metadata + logic = metalog. In *QL98 - Query Languages Workshop*, 1998.

[OV02] Daniel Oberle and Raphael Volz. Implementation of an view mechanism for ontology-based metadata. 523, University of Karlsruhe (TH), 2002.

[PH01] J. Pan and I. Horrocks. Metamodeling architecture of web ontology languages. In *Proceedings of the First Semantic Web Working Symposium (SWWS'01)*, pages 131–149, 2001.

[Run92] 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.

[SD01] Michael Sintek and Stefan Decker. TRIPLE - an RDF query, inference and transformation language. In *Deductive Databases and Knowledge Management (DDLP)*, 2001.

[SLT90] 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.

[Wie93] Gio Wiederhold. Intelligent integration of information. In *SIGMOD-93*, pages 434–437, 1993.

[ZGM98] Y. Zhuge and H. Garcia-Molina. Graph structured views and their incremental maintenance. In *Proc. 14th Int. Conf. on Data Engineering*, 1998.