On 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

May 7, 2002

Abstract

The Semantic Web is an extension of the current Web in which information on the Web 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 over 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 and 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, Web pages are supplemented with semi-structured metadata which are supposed to provide the 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], defines 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. leading to 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 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. Thus, they present the key technology for integrating heterogeneous and distributed systems, i.e. data warehouse 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 of the data found in one ore more sources, as 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. Section 4 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. Section 5 gives a short description of the implementation of our approach. We conclude recapitulating our contribution and giving 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.

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

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

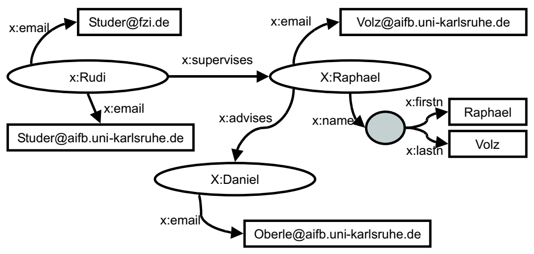

Figure 1: A simple RDF model

Since RDF describes information by means of graphs and has built-in means for linking information coming from different Web sources, RDF inherently meets the ”Web” requirement, where data should be linked.

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 that are embedded into an inheritance hierarchy. Notably, ontologies do not specify a type system for the associated RDF data.

2.1 RDF - graph-like description of data

Initially, the Resource Description Framework[LS99] was intended to enable encoding, exchange and reuse of structured metadata describing Web-accessible resources. This metadata is encoded using a simple resource-property-value mechanism. Each resource-property-value triple, in [LS99] also referred to as (subject, predicate, object), is called a statement. Sets of statements build partially labelled directed graphs and are commonly called RDF models. Graph-theoretically, these graphs are pseudographs, due to the possibility to have multiple edges between (possibly identical) nodes. In this very general data model, two types of nodes exist: resources and literals. Resources can be connected with arbitrary other nodes, whereas literals can only be terminal nodes in edges. In figure 1 ’x:Raphael’ is a resource, whereas ’Volz@aifb.uni-karlsruhe.de’ is a literal [3]. A resource can either be unlabelled - so-called anonymous nodes - or labelled with URIs [BLFIM98]. The resource pointed to by ’x:name’ in figure 1 is an example for an unlabelled node. Literals always have to be labelled with arbitrary strings. All edges in the graph have to be labelled with URIs.

From an object-oriented perspective individual objects are represented in RDF using a set of statements describing the same resource. These objects get an identity if the resource is labelled with an URI. The given object identity is then globally unique: if data is distributed in several RDF models, the sets of statements are simply merged to form a global RDF model. This is a truly monotonic extension as properties can have multiple values, e.g. ’x:email’ for the resource ’x:Rudi’ in figure 1.

[3]For the sake of brevity, we used x to abbreviate the URIs using XML namespaces. In a real world scenario, this could be any URI for disambiguating the resource-labels

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*[4]. From an implementation perspective this is a nice feature since the same 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, here the semantics of data are explicitly given. The reader may note, that ontologies do not enforce structure on RDF data.

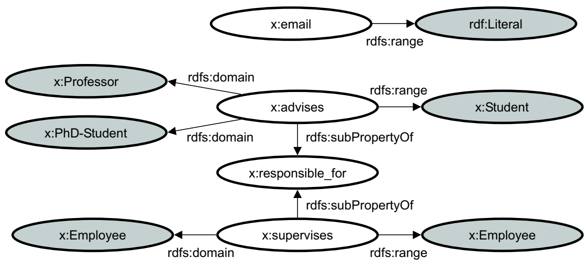

Figure 2: Properties in RDFS for a simple ontology

RDF Schema RDF Schema presents an unique notion of object-orientation. It introduces classes and a subsumption hierarchy on classes (compare figure 3). In RDF Schema subsumption allows for multiple inheritance and has set-inclusion semantics. Unlike commonly-known OO-models class specifications do not define attributes or associations to other classes. Instead such class properties are defined as first-class primitives, so-called properties, which exist on their own. (Possibly multiple) resources, namely those defined in the domain of a property, can share this property, which points to another resource defined in its range. Due to the set-inclusion semantics of inheritance and instantiation, properties are also valid for all members of a class, if a class was used as a properties' domain or range[5]. In figure 2 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 property can validly be applied to all RDF nodes, for example 'x:responsible_for' in figure 2. A subsumption hierarchy on properties is also possible, e.g in figure 2 the property 'x:advises' is a specialization of the property 'x:responsible_for'. Another peculiarity of RDF Schema

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

[5]Thus, for all statements (subject, predicate, object) of a RDF model, "subject" must be in the extension of one of domains of the property "predicate" and "object" must be part of the its extension.

is multiple instantiation thus instances can belong to several classes as depicted in figure 3.

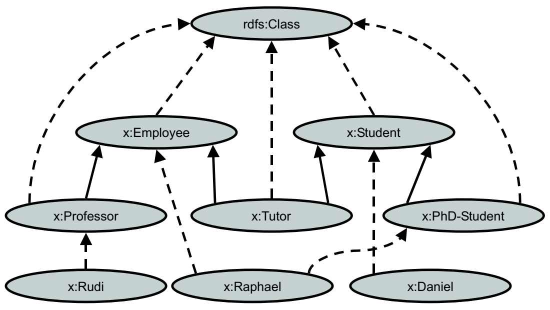

Figure 3: Class hierarchy in RDFS for a simple ontology, dashed lines denote instantiation, solid lines denote subsumption.

Unfortunately, the RDF Schema specification does not provide formal semantics. Therefore, the interpretation of the specification may lead to different semantic interpretations hindering interoperability. The recently proposed RDF Model theory [Hay01] is a first step towards the requirement to have formally specified semantics, but does not yet cover the complete specification.

Another problematic issue with RDF Schema is its unique meta-model. In order to simplify the vocabulary no fixed layers were introduced, thus applications cannot distinguish between the meta-model, the ontology and the instance elements. For example, classes are also instances of themselves leading to Russell's paradoxon known from set theory. This situation also complicates implementation issues tremendously as one cannot rely on set-inclusion semantics to resolve the role of an individual uri.

RDF Schema (Fixed Architecture) Formal semantics are a *sine qua non* to achieve the Semantic Web's major aim of machine-understandable data, hence we rely on RDFS(FA) as proposed by [PH01]. The reader may note, that this does not restrict the generality of our approach besides disallowing infinite layers of classes.

Another reason for choosing RDFS(FA) is that implementation issues are tremendously simplified by utilizing a fixed-layered meta-modelling architecture and the fact to have the same kind of four layer meta-modelling architecture as presented by UML [Gro01], which is commonly known in the object-oriented world, might even promote the tangibility of this approach. The four-layer metamodel of UML architecture is a proven methodology for defining the structure of complex model that need to be reliably stored, shared, manipulated and exchanged [Kob99].

Thus RDFS primitives are disambiguated and separated into layers:

1. The *Metalanguage Layer* is the basis of the metamodeling architecture. It is responsible for the definition of the language for specifying a metamodel. Meta-objects like MetaClass, etc. are situated in this layer.

2. The *Language layer* instantiates a Metalanguages and it is used for defining a language for specifying models. Entities here are e.g. Class and Property.

3. An Ontology is specified at the *Ontology Layer* and an instance of a Metalanguage. Here, a specific domain is described by using instances of metamodel-entities. A class ”Employee” or a property ”advises” would be situated within this layer.

4. The *Instances Layer* defines instances of an ontology. The responsibility of this layer is to describe a specific information domain. Examples for this layer would be instances of the class ”x:Employee” like ”x:Raphael”.

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.

It may seem to be natural to consider views as queries, such as in the relational world. But this does not seem apt here, as we can base the view definition on the ontology. For the same reason we can not rely on previous work about views for semi-structured data models. The views of [ZGM98] consist of object collections only, which makes it unsuitable for the Semantic Web where edges between objects are fundamental. [AGM$^+$97] do consider edges, but do not base their approach onto an ontological basis. This leads to an approach that is similar to the approaches taken for object-oriented databases.

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, particularly the operations manipulating the underlying data should be based on ontological primitives and 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).

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, w. 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]:

3.2.1 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]. Thus collections of view definitions form a new external ontology. This implies that view definitions behave like ontology definitions even though they are virtual. We follow the established terminology and refer to virtual classes and properties with regard to views. The classes and properties coming from the underlying RDF model are called base classes and base properties respectively.

3.2.2 Placement of view elements in the hierarchy

The next consideration we have to decide upon on is where to place the virtual entities in their respective hierarchies. We follow [AB91, Run92, SLT90] by automatically positioning the virtual entities at the appropriate location of their respective hierarchies. As we don't have to worry about correct typing, this means that we only have to keep the subsumption hierarchy well-formed. The type of conflict arising in object-oriented databases when a view is defined via a projection and selection at the time, leading to supertypes (projection operator) and subclasses (selection operator) does not arise here.

3.2.3 View population

The next 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 as well as object-preserving views, which is the standard case for RDF as the following section will show. 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.

3.2.4 Updates on views

Whether or not data can be updated is another important criterion for an 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. 7) can not 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 follows a functional approach (like [ASL89]) and supports generalized path expressions featuring variables on RDF nodes and properties. RQL is able switch between schema and data querying while exploiting class and property hierarchies.

The following query gives a gentle introduction to RQL and returns all resources which have a property email,. The construct {X}x:email{Y} is called a basic data path expression.

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

For graph-based data models generalized path expressions are of great use as queries are simplified tremendously. Additionally RQL provides short hand notations for accessing class extensions. This is illustrated in the following query, which returns the email addresses of all students advised by PhD-Students:

```
SELECT Z
FROM PhD-Student{X}.
x:advises{Y}. x:email{Z}
```

Here PhD-Student$\{$X$\}$ determines the extent of class PhD-Student. The "." used to concatenate the path expressions is just a syntactic shortcut for an implicit join condition between the extension of the class "x:PhDStudent" and the domain values of the property "x:advises". This construction is taken, as RDF classes do not define types on which attribute extractor operators like "." can be defined. Thus, if no property is given for any element of the respective extends, the path expression can not be evaluated and returns the empty set. Due to lack of space, we refer the interested reader to [ACK$^+$00] for a detailed description of RQL.

5 View Definition Language

In order to support the notion of properties being first-class citizens in RDF, we differentiate between views on classes (virtual classes) and views on properties (virtual properties). Views on classes basically refine the extension of a class using several view operations.

Views on properties alter the definition of base properties and therefore provide projection like functionality. Notably, projections on classes do not make sense within RDF due to it's graph like nature.

The view definition language introduced here implements the goals mentioned in section 3, thus a collection of views is defined within an external ontology (cf. figure 4). As views cannot be distinguished from base classes and base properties one can nest external ontologies as well as view definitions. To support this, every external ontology must be bound to a certain URI, which is required to identify it. In order to support the goal of Web-awareness, external ontologies can be created on a set of source RDF models. XML namespaces can be bound to aliases to support short hand notations for view definitions.

```
{ BIND XML-NS TO URI }

CREATE EXTERNAL ONTOLOGY <URI>

BASE <MODEL_URI>

{ BASE <MODEL_URI> }
```

Figure 4: BNF-like notation of header defining an external ontology

5.1 Import operations

The view administrator can import base classes and properties into the external ontology. It is possible to import classes as well as properties, to this extend two operations are defined:

```
IMPORT CLASS <URI>
IMPORT PROPERTY <URI>
```

Notably, an import statement only imports the exact class or property identified by a given <URI>. It does not import all existing subclasses respectively subproperties to provide a fine granular control over importing data. Nevertheless, these operations make all instances of classes available [6]. Accordingly all statements that use an imported property are available within the external ontology.

Naturally import operations have object-preserving semantics, thus updates can be propagated easily to the underlying base models.

5.2 Virtual properties

Virtual properties can be defined using several operations, we basically consider renaming properties, refining and defining the extension of properties.

5.2.1 Renaming properties

The following statement creates a new virtual property that is populated with the extension of the property <BASE_URI>:

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

Semantically the property is a subproperty of the property <BASE_URI>. Thus, if <BASE_URI> or any of its superproperties are visible in the external ontology, the semantic fact that <VIEW_URI> subsumes these properties must be properly expressed in the external ontology, viz. virtual entities must be properly classified with respect to each other. This issue is common to all virtual properties and virtual classes and described in the next section. Renaming is object-preserving and therefore principally updateable.

5.2.2 Refining properties

The following operation refines a base property <BASE_URI> by altering its domain and ranges:

```
CREATE VIRTUAL PROPERTY <VIEW_URI>
ON <BASE_URI>
    { SET DOMAIN <DOMAIN_URI> }
    [ SET RANGE <RANGE_URI> ]
```

[6]The reader may note, that this does not make all *members* of a class available, which refers to the instances of possible subclasses

A refinement can only further constraint the existing domains and ranges of <BASE_URI>. Thus, <DOMAIN_URI> must be a valid subclass of some given domain. Notably, if no domains or ranges were given for <BASE_URI> all classes are valid. The consistency of the refinement is checked at compile time. The following definition refines the property ”x:email” from 1

```
CREATE VIRTUAL PROPERTY
x:stud-mail
ON x:email
SET DOMAIN x:Student
```

Thus only email addresses given for members of the student class are visible within the external ontology via the property x:stud-mail. The reader may notice that it is principally possible to create a virtual property that uses the URI of the base property (thus renaming is avoided), generally every URI can be used if it is not used elsewhere. Refining properties is a object-preserving operation, it is therefore principally updateable.

5.2.3 Defining the extension of properties

The following operation defines the extension of virtual property via an arbitrary RQL query that returns binary relations with variables called subject and object.

```
CREATE VIRTUAL PROPERTY <VIEW_URI>
    { SET DOMAIN <DOMAIN_URI> }
    [ SET RANGE <RANGE_URI> ]
    USE <query>
```

The fixed names are required to facilitate user understanding of the semantics of this operation. This operation is implemented as follows: First the query is evaluated, then each pair of RDF nodes in the result is augmented with the predicate <VIEW_URI>, if the subject variable is in the extension of the domain and the object variable is a valid range for the property. For example the following definition creates a new virtual property which relates all PHD-Students with the email addresses of the advised students.
1

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

It can easily be seen that this operation is object-generating and therefore not updateable. The created property is also not part of the property hierarchy, as no semantically correct classification for this virtual property can be given.

5.3 Virtual classes

More operations are possible for views on classes. The operations provided here always define the extension of classes. Notably users can only see those properties of instances that visible within the external ontology via either import or one of the operations for defining virtual properties.

We introduce special operations for each of the operations known from relational algebra in order to be able to provide a correct classification of virtual classes with regard to the class hierarchy. As RDF supports multiple instantiation instances of classes do not have to be reclassified, they simply remain instances of the base classes and additionally gain instantiation of the newly defined virtual classes.

5.3.1 Selection views

The first operation that we consider for views on classes is selection, where a subset of the members of a certain class is selected.

```
CREATE VIRTUAL CLASS <VIEW_URI>
ON <URI>
SELECTION <query>
```

Again, <query> has to be an appropriate RQL query. In this case the query is only allowed to return unary relations. The operation is implemented as follows: First the query is evaluated, second the result is intersected with the extension of the underlying class[7], thus it is ensured that the query actually defines a selection.

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

```
CREATE VIRTUAL CLASS x:Supervisors
ON x:Employee
SELECTION
SELECT X
{X}x:supervises
```

Naturally selection is a object-preserving operation and thus updateable. As a subset of the original extension of the respective base class is selected, selection views semantically subsume their respective base class. Thus, if either the base class or one of its super classes is visible within the external ontology, the semantic fact that this virtual class subsumes those classes must be properly captured. As mentioned before we detail this feature in the following section.

5.3.2 Difference view

The difference view class is a subclass of the minuend and contains all resources that are not member of the subtrahend. Thus all properties defined for the minuend can validly be applied to the view class.

[7]Notably this can be virtual as well as base classes

In general, a equivalent selection expression that resembles the difference can be found if negation is permitted. The results regarding object-preservation and updateability for selection-based view classes therefore apply to difference-based view classes. We provide the following syntax

```
CREATE VIRTUAL CLASS <VIEW_URI>
ON <URI> DIFFERENCE <URI>
```

For example, the following definition would create a virtual class ”x:Unemployed-Students” which is populated with all unemployed students.

```
CREATE VIRTUAL CLASS
x:Unemployed-Students
ON x:Student DIFFERENCE x:Employee
```

5.3.3 Union view

The union view merges the extensions of the participating classes and is created via the following expression:

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

Semantically, union leads to a super class of the participating classes, thus a union view class is a superclass of the participating classes and contains all resources that are in their respective extensions. It also subsumes the smallest common superclass of the participating classes (if such a class exists). Thus all properties that can be validly applied to this superclass can be validly applied to the union view class.

To provide an example we define a class ”x:Scientists” which consist of professors as well as PhD-Students:

```
CREATE VIRTUAL CLASS x:ScientificStaff
ON x:Professor UNION x:PhDStudent
```

With respect to our ontology example 3 no common superclass of ”x:Professor” and ”x:PhDStudent” can be found. Union views are again object-preserving and therefore updateable.

5.3.4 Intersection view

The intersection view is populated with the intersection of the extensions of participating classes.

```
CREATE VIRTUAL CLASS <VIEW_URI>
ON <URI> INTERSECT <URI>
{ INTERSECT <URI> }
```

From a set theoretic perspective the intersection is a subset of participating sets, thus an intersection view is a subclass of all participating classes. Naturally all properties defined for the participating classes can validly be applied to the intersection view. The operation is again object-preserving.

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

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

5.3.5 Arbitrary virtual classes

The last operation that we consider for virtual classes are arbitrary queries, where the population of a class is determined via the results of the query.

```
CREATE VIRTUAL CLASS <VIEW_URI>
{ SUBCLASS OF <URI>}
USE <query>
```

Again, <query> has to be an appropriate RQL query. In this case the query is only allowed to return a list of URIs that are considered to be instances of that class.

Notably, it is no longer possible to automatically classify the virtual class to the appropriate location in the class hierarchy. This has to be done manually instead.

It is not possible to update such virtual classes as updates might not be validly propagated to the underlying models. This is due to the fact that properties, which are valid within the view due to manual classification might no longer be valid for instances in the base model where the information is lost that an instance belongs to this virtual class.

6 Consistency and Classification

In order to be consistent with its base ontology, a view has to fulfill certain properties. This is also true for object-oriented views [HZ88], thus a view mechanism is only complete if it ensures semantic consistency.

For this purpose, we defined a set of axioms expressing consistency. The subset of an RDF model that defines an ontology is translated into logical predicates[8]. With this mapping, we use can use a logic programming systems to check the axioms. Generally

[8] $statement(subject, predicate, object)$

speaking, the axioms are a set of Horn-rules with a binary[9] violation-predicate in the head. A set of statement-predicates, describing the external ontology, are input to an inference-engine, followed by a query that determines whether any violation occured[10]. If any violations occur, the external ontology isn't consistent. Due to lack of space we only give two example rules for consistency, we refer the interested reader to [OV02] for a complete description.

6.1 Inferring the class hierarchy

As mentioned before the correct placement of classes and properties in the respective hierarchies must be deduced. We were able to express this deduction by Horn-rules, as well. This time however, subClassOf-statements form the head of such rules. For the sake of tangibility, we have chosen to present them in an informal way (cf. [OV02] for a concise and formal presentation):

1. If a virtual class generalizes other classes via a union view,a subclass statement to all visible classes is automatically added.

2. For union views the smallest common superclass of all classes participating in the definition is deduced. It is automatically imported into the ontology, if not visible to ensure that properties are defined for the union view.

3. The required subclass statements for imported classes are copied from the base-ontology.

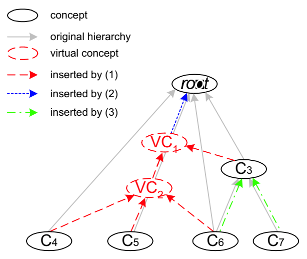

Figure 5: Hierarchy consistency example

Figure 5 illustrates the inference process. VC_1 and VC_2 are union views. VC_1 defines a union on C_3 and VC_2, whereas VC_2 unifies the extensions of C_4, C_5, C_6 respectively. Rule (1) deduces, that VC_2 and C_3 are subclasses of VC_1 and that C_4, C_5, C_6 are subclasses of VC_2.

Now rule (2) comes into action and determines that $root$ is the closest possible super class to VC_1. This is correct as all imported classes C_i are subclasses of $root$ in the base model. Currently, neither of the imported classes C_i are related as they were imported separately. Now, rule (3) puts them into relation again.

A similar set of rules establishes the property hierarchy.

[9]the first parameter serves as description, whereas the second refers to the violating resource.

[10]that is $\neg violation(x, y)$ holds for all x, y

6.2 Ensuring the correctness of property domains

Another important and interesting issue are domains and ranges of imported properties. As long as at least one original domain, i.e. a class of the base-ontology, is imported into the view, nothing has to be changed. Otherwise, the domains have to be redefined to appropriate subclasses that were imported (cf. figure 6). If no such classes exist, further axioms for consistency come into play that bind the violation predicate to the respective property disallowing the import operation.

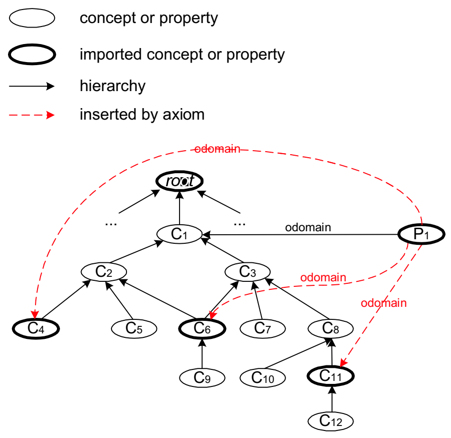

Figure 6: Domain consistency example

The figure 6 depicts an interesting example. The domain of imported property P_1 used to be class C_1. But C_1 is not part of the external ontology. The domains therefore go down to the closest imported subclasses C_4, C_6, C_{11}.

The axioms for inferring the new ranges of an imported property are similar. One difference is that there can be only one range per property. Thus, if the original range is not imported and several possible candidates from the imported subclasses exist, a violation has to be issued again.

7 Implementation

We give a brief survey about 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[11].

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 (c.f. Triple [SD01], Metalog [MS98]). The usual approach taken for here is the transla-

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

tion of RDF models into simple ternary predicates[12] and to use query RDF using logic programming approaches of different expressivity. Using 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 virtual concepts and properties. The extension and values defined for virtual resources are computed at runtime. As soon as a view is loaded, the consistency check and inference processes are started. We therefore transform the external ontology into logic which serves as input to SiLRI. Inferred inconsistency then would lead to a denial of service.

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 that other Web users or Web applications will be built by exploiting these ontologies. 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 view mechanism 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

[12] statement(subject, predicate, object)

will be gradually extended to include these additional aspects of a view mechanism for the Semantic Web.

Acknowledgements We thank our colleague Alexander Maedche for comments on early versions of the paper. Our work was partially sponsored by European Commission information society technologies (IST) programme via the WonderWeb project (IST-2001-33052)

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.

[ASL89] A. M. Alashqur, S. Y. W. Su, and H. Lam. OQL: A query language for manipulating object-oriented databases. In *VLDB Conference*, pages 433–442, 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.

[BLFIM98] Tim Berners-Lee, R. Fielding, U.C. Irvine, and L. Masinter. Uniform resource identifiers (uri): Generic syntax. Internet Engineering Task Force (IETF), Request for Comments (RFC) 2396, August 1998.

[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.

[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.

[HZ88] Sandra Heiler and Stanley Zdonik. Views, Data Abstraction, and Inheritance in the FUGUE Data Model. In K.R. Dittrich, editor, *Advances in Object-Oriented Database Systems, Proc. 2nd Intl. Workshop on Object-Oriented Database Systems*, number 334 in Lecture Notes in Computer Science, pages 225–241, FRG, September 1988. Springer.

[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.