The Semantic Web - on the respective Roles of XML and RDF

Stefan Decker[1], Frank van Harmelen[3,4], Jeen Broekstra[4]
Michael Erdmann[5], Dieter Fensel[3], Ian Horrocks [2], Michel Klein[3], Sergey Melnik [1]

[1]Department of Computer Science
Stanford University, Stanford, USA
{stefan,melnik}@db.stanford.edu

[2]Department of Computer Science
University of Manchester, UK
horrocks@cs.man.ac.uk

[3]Vrije Universiteit Amsterdam, Holland
{dieter, frankh, mcaklein}@cs.vu.nl

[4]AIdministrator Nederland B.V.
jeen.broekstra@aidministrator.nl
frank.van.harmelen@aidministrator.nl

[5]Institut AIFB, University of Karlsruhe
erdmann@aifb.uni-karlsruhe.de

Abstract

The next generation of the Web is often characterized as the “Semantic Web”: information will no longer only be intended for human readers, but also for processing by machines, enabling intelligent information services, personalized Web-sites, and semantically empowered search-engines. The Semantic Web requires interoperability on the semantic level. Semantic interoperability requires standards not only for the syntactic form of documents, but also for the semantic content. Proposals aiming at semantic interoperability are the results of recent W3C standardization efforts, notably XML/XML Schema and RDF/RDF Schema. In this paper, we make the following claims:

- A further representation and inference layer is needed on top of the currently available layers of the WWW.
- To establish such a layer, we propose a general method for encoding arbitrary ontology representation languages into RDF/RDF Schema.
- We illustrate the extension method by applying it to a particular ontology representation and inference language (OIL).

1 Introduction

The World Wide Web has been made possible through a set of widely established standards which guarantee interoperability at various levels: the TCP/IP protocol has ensured that nobody has to worry about transporting bits over the wire anymore. Similarly, HTTP and HTML have provided a standard way of retrieving and presenting Hypertext text documents. Applications have been able to use the common infrastructure, which has lead to the WWW as we know it now.

The current Web can be characterized as the second Web generation: the first generation Web started with handwritten HTML pages; the second generation made the step to machine generated and often active HTML pages. These generations of the Web were meant for direct human processing (reading, browsing, form-filling). The third generation Web, which one could call the "Semantic Web", aims at machine processable information. This coincides with the vision that Tim Berners-Lee describes in his recent book "Weaving the Web" [Berners-Lee, 1999]. The Semantic Web will enable intelligent services such as information brokers, search agents, information filters etc. Such intelligent services on the Knowledgeable Web should surpass the currently available versions of these

services, which are limited in their functionality, and (most importantly), only work as stand-alone services that do not interoperate.

The Semantic Web, with machine processable information contents, will only be possible when further levels of interoperability are established. Standards must be defined not only for the syntactic form of documents, but also for their semantic content. Such semantic interoperability is facilitated by recent W3C standardization efforts, notably XML/XML Schema and RDF/RDF Schema. These efforts are summarized in Figure 1.

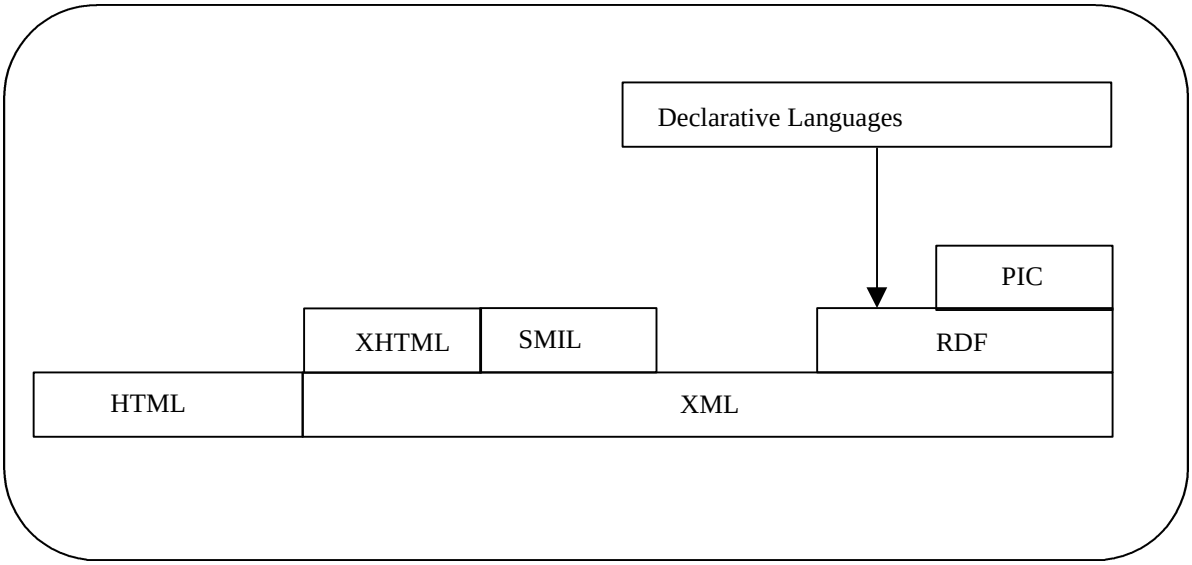


Figure 1 Language Layers on the Web

In this paper, we make the following three claims:

- A further representation and inference layer is needed on top of the currently available layers of the WWW.
- To establish such a layer, we propose a general method for encoding arbitrary ontology representation languages into RDF/RDF Schema

We illustrate the extension method by applying it to a particular ontology representation and inference language (OIL) [Horrocks et al, 2000] by placing it on top of the RDF/RDF-Schema layer.

The rest of the article is organized as follows: In section 2 we explain the role of ontologies in the architecture of the Semantic Web. In sections 3 and 4 we briefly summarize the currently available standards for semantic interoperability, namely XML and RDF. In section 5 we argue why the hope to exploit XML for semantic interoperability is ill-fated in the long run, and why RDF is a better candidate for this purpose.[1] In sections 6 and 7 we demonstrate that our claims can indeed be realized by showing how a particular ontology language OIL can be Web-enabled by making it RDF-compliant through a generic two-step translation procedure.
Finally, section 8 summarizes and concludes the article.

2 Ontologies

In the following we investigate one possibility for achieving a more semantics-based Web.

Ontologies (often also referred as Domain Model) can play a crucial role in enabling the processing and sharing of knowledge between programs on the Web. Ontologies are generally defined as a "representation of a shared

[1] However, we don't propose solutions for the interoperability problem — we just argue that the problem is easier to solve with RDF than with pure XML, and point to the literature.

conceptualization of a particular domain". They provide a shared and common understanding of a domain that can be communicated across people and application systems. They have been developed in Artificial Intelligence to facilitate knowledge sharing and reuse. Recent articles covering various aspects of ontologies can be found in [Uschold & Gruninger, 1996], [van Heijst et al., 1997], [Gomez Perez & Benjamins, 1999].

An example of the use of ontologies on the Knowledgeable Web is in e-commerce sites, where ontologies are needed (a) to enable machine-based communication between buyer and seller, (b) to enable vertical integration of markets (e.g. www.verticalnet.com), and (c) to leverage reusable descriptions between different marketplaces.

A second example of the use of ontologies can be found in search engines. By using ontologies the search engines can escape from the current keyword-based approach, and can find pages that contain syntactically different, but semantically similar words (e.g. www.hotbot.com).

Typically, an ontology contains a hierarchical description of important concepts in a domain (is-a hierarchy), and describes crucial properties of each concept through an attribute-value mechanism. Additionally, further relations between concepts may be described through additional logical sentences. Finally, individuals in the domain of interest are assigned to one or more concepts in order to give them their proper type.

Figure 2 shows a small example of an ontology (expressed in the ontology language OIL [Horrocks et al, 2000]). OIL will serve as an example throughout the paper, so we explain part of OIL in more detail.

```
class-def animal                    % animals are a class
class-def plant                     % plants are a class
    subclass-of NOT animal          % that is disjoint from animals
class-def tree
    subclass-of plant               % trees are a type of plants
class-def branch
    slot-constraint is-part-of      % branches are parts of trees
        has-value tree
class-def leaf
    slot-constraint is-part-of      % leafs are parts of branches
        has-value branch
class-def defined carnivore         % carnivores are animals
    subclass-of animal
    slot-constraint eats            % that eat only other animals
        value-type animal
class-def defined herbivore         % herbivores are animals
    subclass-of animal
    slot-constraint eats            % that eat only plants or parts of plants
        value-type plant OR (slot-constraint is-part-of has-value plant)
class-def giraffe                   % giraffes are animals
    subclass-of animal
    slot-constraint eats            % and they eat leafs
        value-type leaf
class-def lion
    subclass-of animal              % lions are also animals
    slot-constraint eats            % but they eat herbivores
        value-type herbivore
class-def tasty-plant               % tasty plants are plants that are eaten by
    subclass-of plant               % both herbivores and carnivores
    slot-constraint eaten-by
        has-value herbivore, carnivore
```

Figure 2 Ontology Example

An ontology consists of a list of class definitions ("class-def") and slot definitions ("slot-def") (slot definitions are omitted in the example). A class definition associates a class name with a class description, and consists of the following components (any of which may be omitted):

- The **type** of definition — this can be either primitive or defined; primitive classes give necessary but not sufficient conditions for class membership.
- **subclass-of** — an list of one or more class-expressions (see below). The class being defined in the class-definition must be a sub-class of each of the class expressions in the list.
- **slot-constraint** — a list of zero or more slot-constraints (see below). The class being defined in the class-definition must be a sub-class of each of the slot-constraints in the list (note that a slot-constraint defines a class).

A **class-expression** can be either a class name, a slot-constraint, or an arbitrarily complex boolean combination of class expressions.

A slot-constraint is a list of one or more constraints (restrictions) applied to a slot. A slot is a binary relation (i.e., its instances are pairs of individuals), but a slot-constraint is actually a class definition—its instances are those individuals that satisfy the constraint(s). For example, if the pair (Leo, Willie) is an instance of the slot eats, Leo is an instance of the class lion and Willie is an instance of the class wildebeest, then Leo is also an instance of the has-value constraint wildebeest applied to the slot eats. A slot-constraint consists of the following main components:

- **name** — a slot name (a string).
- **has-value** — a list of one or more class-expressions. Every instance of the class defined by the slot constraint must be related via the slot relation to an instance of each class-expression in the list. For example, the has-value constraint: `slot-constraint eats has-value zebra, wildebeest` defines the class each instance of which eats some instance of the class zebra and some instance of the class wildebeest. Note that this does not mean that instances of the slot-constraint eat only zebra and wildebeest: they may also be partial to a little gazelle when they can get it.
- **value-type** — a list of one or more class-expressions. If an instance of the class defined by the slot-constraint is related via the slot relation to some individual x, then x must be an instance of each class-expression in the list.

3 XML

By now, XML [Bray et al, 1998] is widely known in the WWW community and is the basis for a rapidly growing number of software development activities. We will discuss XML only briefly, in order to keep this article self-contained.

XML is intended as a markup-language for arbitrary document *structure*, as opposed to HTML, which is a markup-language for a specific kind of hypertext documents. An XML document consists of a properly nested set of open and close tags, where each tag can have a number of attribute-value pairs. Crucial to XML is that the vocabulary of the tags and their allowed combinations is not fixed, but can be defined per application of XML. An example depicted in Figure 3 serializes a part of the ontology given above.

```
<class-def>
    <class name="plant"/>
    <subclass-of>
        <NOT><class name="animal"/></NOT>
    </subclass-of>
</class-def>
<class-def>
<class name="tree"/>
    <subclass-of>
        <class name="plant"/>
     </subclass-of>
</class-def>
<class-def>
      <class name="branch"/>
      <slot-constraint>
            <slot name="is-part-of"/>
            <has-value>
               <class name="tree"/>
            </has-value>
      </slot-constraint>
</class def>
```

Figure 3 Partial XML-Serialization of the Ontology

From the indentation of the above example, it is easy to see that the basic data-model of XML is a labeled tree, where each tag corresponds to a labeled node in the data-model, and each nested sub-tag is a child in the tree[2]. It is important to point out that the above is only one possible XML syntax for the ontology given above (in fact, it is the one that is defined in [Horrocks et al, 2000]. We could just have easily have defined many other XML versions of the same semantic information. Figure 4, for example, contains essentially the same information as the final class definition in Figure 3, is also correct XML, but has an entirely different syntactic form. The possibility for multiple serializations steams from the fact, that XML is foremost a means to define grammars. Different grammars can be used to define the same syntactic content.

[2] XML elements can be given a unique ID-attribute. Using this ID, any node in the tree can refer to any other node. As a result, arbitrary graphs can be encoded in this manner. However, we emphasize that this is an *encoding* of a general graph in a tree, and that the basic data-model is still a tree.

Any XML document whose nested tags form a balanced tree is a well-formed XML document. Furthermore it is possible to enforce constraints on which tags should be used, and which nesting of these tags is allowed. In XML 1.0 this is done in a Document Type Definition (DTD). Such a DTD specifies the allowed combinations and nesting of

```
<class-def>
    <name>branch</name>
    <slot-constraint>
        <name>is-part-of</name>
        <has-value>tree</has-value>
    </slot-constraint>
</class-def>
```

Figure 4 Another XML-Serialization

tag-names, attribute-names, etc. using a grammar formalism. Developments are well underway at W3C to replace DTD's with XML Schema definitions [Thompson et al, 2000][Biron & Malhotra, 2000]. Although XML Schema's offer a number of advantages of DTD's, their role is essentially the same: they define a grammar for XML documents.

In practice, XML is being used for a number of rather different purposes:

- as a serialization syntax for other markup languages. For example, the SMIL multi-media markup language [Hoschka, 1998] uses the XML format for its markup syntax. Pure syntactically speaking, SMIL is simply a particular XML DTD. The real value of SMIL lies in the fact that there exists a common understanding of the intended meaning of the elements of that particular DTD (for example the fact that the "<PAR>" element indicates media-objects that must be played in parallel). However, it is important to realize that the DTD only specifies the syntactic conventions, and that any such intended semantics remains outside the realm of the XML specification.
- as semantic markup of Web-pages. The example in Figure 4 an be used as markup in a Web-page. An XSL style-sheet [Clark, 1999] can then use this markup to render the different elements of the page in appropriate manners.
- as a uniform data-exchange format. The above example can also be a data object transferred between two applications. Again we stress that in both cases (document markup or data format), only the syntactic structure of XML is enforced; the intended meaning of the different elements is entirely implicit in the XML document.

4 RDF

RDF [Lassila & Swick, 1999] is a recent W3C recommendation, and as a result less widely known than XML. Its motivation is to provide a standard for meta-data, for descriptions about resources on the web. However, the distinction between data and meta-data is sophisticated: RDF as framework for describing data about web-resources is also capable to represent data.

The basic construction in RDF is an object-attribute-value triple: an object O has an attribute A with value V. Such a triple corresponds to the relation that is commonly written as A(O,V), for example: hasPrice('http://www.books.org/ISBN0012515866', "$62"). A third way to think about such a basic RDF triple is as a labeled edge between two nodes: '[O] -A-> [V]'. This last notation is particularly useful, since RDF allows objects and values (1st and 3rd elements of the basic RDF triples) to be mixed: any object can play the role of a value. In terms of the graph representation, this amounts to chaining two labeled edges:
For example:

```
hasName('http://www.w3.org/employee/id1321',"Jim Lerners").
authorOf('http://www.w3.org/employee/id1321','http://www.books.org/ISBN0012515866').
hasPrice('http://www.books.org/ISBN0062515861, "$62").
```

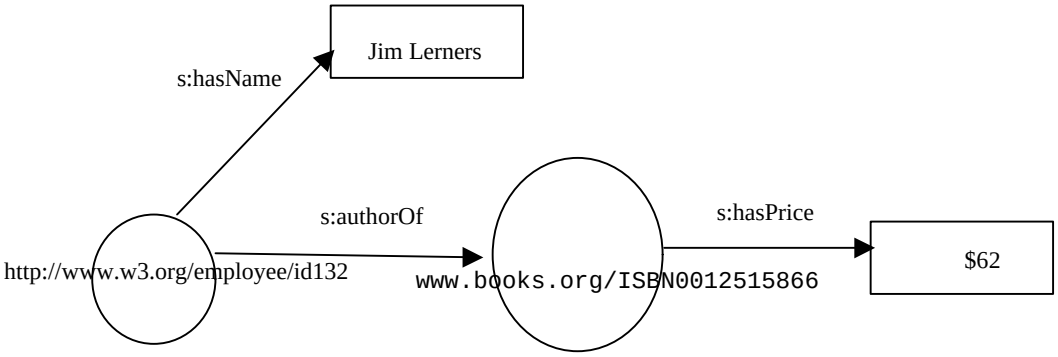


Figure 5 RDF Graph

has the labeled graph given in Figure 5.

Furthermore, RDF allows a form of *reification*[3], allowing any RDF statement to be the object or value of a triple allowing the graphs not only to be chained, but also to be nested. Finally, it is possible to indicate that a given object is of a certain type, such as stating that 'ISBN03547X' is of type *book*:

```
<rdf:Description about="www.books.org/ISBN0012515866">
        <rdf:type resource="http://description.org/schema/book">
</rdf:Description>
```

It is important to note that the intended role of RDF is to provide a basic object-attribute-value (OAV) data-model for meta-data. Besides this intended Object-Attribute-Value-semantics (which is itself only informally described in the RDF standard), no further data modeling commitments are made. In particular, no particular reserved terms for any further data modeling are defined. Like the XML data model, the RDF data model provides no mechanisms for declaring the property names that are to be used.
Just as XML Schema provides a vocabulary definition facility for XML, RDF Schema [Brickley & Guha, 2000] provides a similar facility for RDF. RDF Schema allows the definition of a particular vocabulary that should be used for RDF attributes (e.g. *authorOf*), and it allows the specification of the kinds of object to which these attributes may be applied. In other words, the RDF Schema mechanism provides a basic type system for use in RDF models.
This type system itself uses some predefined terminology. These are terms such as *Class* and *subClassOf* that are used in specifying application-specific schemas. RDF Schema expressions are themselves also valid RDF expressions (just as XML Schema expressions are valid XML).

Two crucial RDF Schema constructions are *subClassOf* and *subPropertyOf*. RDF objects may be instances of one or more classes; this is indicated using the *type* property. The subClassOf property form RDF Schema allows the specification of the hierarchical organization of such classes. subPropertyOf does the same for properties. Furthermore, constraints on properties can be specified using *domain* and *range* constructs.

Using these constructions, RDF Schema can be used to extend both the vocabulary and the intended interpretation of RDF expressions. This is the mechanism that we use in section 7 to translate an ontology representation language to RDF.

[3] Latin: *res*=thing, *facere*=to make; re-ify=to turn into a thing

5 Representing Knowledge on the Web with XML or RDF?

The Web is a universal medium for exchanging data and knowledge: for the first time in history we have a widely exploited many-to-many medium for data interchange. This medium poses new requirements for any format used for exchanging data on the web:

1 *Universal expressive power*: Since it is not possible to anticipate all potential uses, a data format must have enough expressive power to express any form of data.
2 *Support for Syntactic Interoperability*: By syntactic interoperability we mean how easy it is to read the data and get a representation that can be exploited by applications. For example, software components like parsers or query APIs should be as reusable as possible among different applications. Syntactic interoperability is high when the parsers and APIs needed to manipulate the data are readily available.
3 *Support for Semantic Interoperability*: With semantic interoperability we mean the difficulty of understanding the data. Please note the difference from syntactic interoperability: syntactic interoperability talks about parsing the data, while semantic interoperability means to define mappings between unknown terms and known terms in the data. We regard this requirement as one of the most important, since the cost of establishing semantic interoperability is usually higher then for establishing syntactic interoperability, due to the need for content analysis.

In the next two subsections we compare XML and RDF regarding the above sketched requirements.

5.1 Using XML

XML is just formalism for defining a grammar, so anything can be encoded in XML if a grammar can be defined for it. This means that the first requirement is fulfilled by XML.

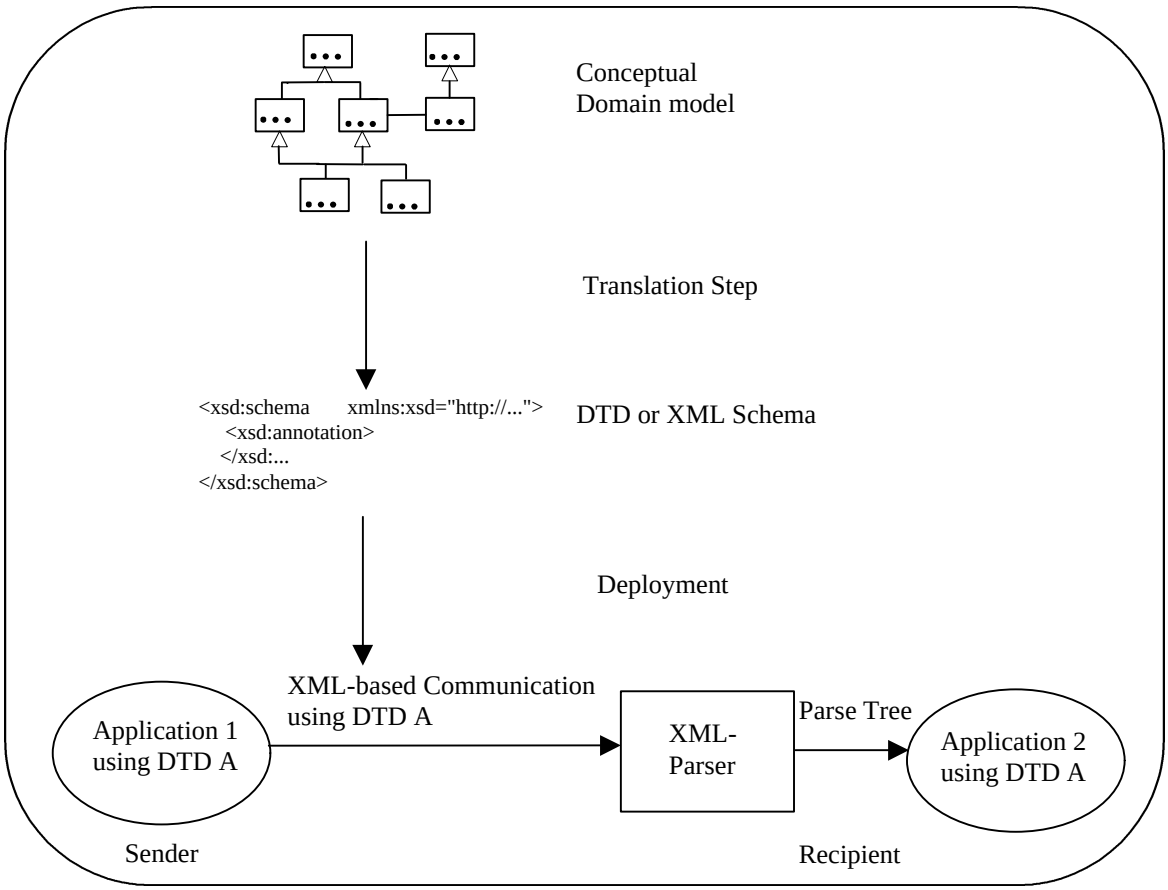


Figure 6 DTD Development and Point-to-Point communication with XML

Since an XML parser can parse any XML data, and is usually a reusable component (XML parsers are already part of standard software libraries e.g. [Davidson, 2000]), the second requirement is also fulfilled by XML.

However, when it comes to semantic interoperability, XML has disadvantages. XML is aiming at the structure of documents and does not impose any common interpretation of the data contained in the document. This is the major limitation of XML: since XML just describes grammars there is no way of recognizing a semantic unit from a particular domain of interest.

Although this limitation is at the heart of the ``schema-wars'' which are currently raging at forums such as www.biztalk.org, www.oasis-open.org, www.rosettanet.org, www.schema.net, etc., this fundamental aspect of XML is not yet widely recognized.

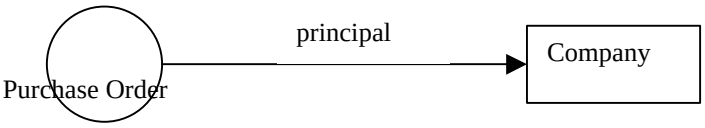
We illustrate this point by two scenarios. In the first scenario (depicted in Figure 6), two applications try to communicate to each other. Both applications agree on the use and intended meaning of the document structure given by DTD A.

However, before data exchange can be established between the applications, a model of the domain of interest has to be built, since it is necessary to clarify what kind data is sent from the first application to the second.

This model is usually described in terms of objects and relations (e.g. like in UML (e.g. [Page-Jones & Constantine, 1999]) or Entity Relationship Modeling (e.g. [Barker, 1990])). From the domain model a DTD or an XML Schema is constructed. Since a DTD just describes a grammar, and there exists multiple possibilities to encode a given

domain model into a DTD ([Erdmann & Studer, 1999]. So the direct connection from the DTD to the domain model is lost and it cannot be easily reconstructed.

a)



b)

Encoding DTD	Example XML Instance Data
<!ELEMENT PurchaseOrder (principal)> <!ATTLIST PurchaseOrder id ID #REQUIRED> <!ELEMENT principal (Company)> <!ATTLIST Company id ID #IMPLIED>	<PurchaseOrder id="X"> <principal> <Company id="Y"/> </principal> <PurchaseOrder>
<!ELEMENT principal (PurchaseOrder, Company)> <!ELEMENT PurchaseOrder (#CDATA)> <!ELEMENT Company (#CDATA)>	<principal> <PurchaseOrder>X</PurchaseOrder> <Company>Y</Company> </principal>
<!ELEMENT PurchaseOrder (id, principal)> <!ELEMENT id (#CDATA)> <!ELEMENT principal (Company)> <!ELEMENT Company (id)>	<PurchaseOrder> <id>X</id> <principal> <Company> <id>Y</id> </Company> </principal> </PurchaseOrder>
<!ELEMENT rel EMPTY> <!ATTLIST rel src CDATA #REQUIRED type CDATA #REQUIRED dest CDATA #REQUIRED>	<rel src="X" type="principal" dest="Y"/>
<!ELEMENT PurchaseOrderInfo (Company)> <!ATTLIST PurchaseOrderInfo orderID ID #REQUIRED> <!ELEMENT Company (#CDATA)>	<PurchaseOrderInfo orderID="X"> <Company>Y</Company> </PurchaseOrderInfo>

Figure 7 show a binary relationship “principal” between two concepts “Purchase Order” and “Company” and several encoding possibilities in XML. Just from the DTDs it is impossible to determine, what the relation is and what are the concepts. So the reengineering of the domain model is difficult from just the DTD. Please note, that this example is a simple case, a single binary relationship. Significantly more encoding options exists in case of multiple, ordered etc. relationships. The relationship depicted in Figure 7 represent a valid RDF model.

The advantage of using XML in this case is limited to the reusability of the parsing software components. Certainly, this is a situation that is relevant and useful. But this is a one-on-one communication with parties that have reached agreement in advance. It neglects properties on the Web, where one often has to cope with multiple and frequently changing communication partners. In fact, this scenario amounts to the “implicit and procedural” representation of semantics that was discussed in the first paragraph of section 2.

The second scenario (sketched in Figure 8) occurs frequently on the Web: new business partners have to be added to an existing relationship, new information sources become available etc. Since this is a very frequent operation, it is important to reduce the costs of adding new communication partners as much as possible. However, using XML and DTDs (or XML Schemas) requires much more effort than necessary (see Figure 9): it is not sufficient to map one domain model to the other one, since the domain models are encoded in the DTDs, and have to be reengineered first. A direct mapping based on the different DTDs is not possible, since the task is *not to map one grammar* to another grammar, but to *map objects and relations* from one domain of interest to another one. So we have to define the mappings between the different domain models, not between the different DTDs. Defining the mapping between DTDs is an additional step, after the correct domain mappings are identified. So the following tasks have to be

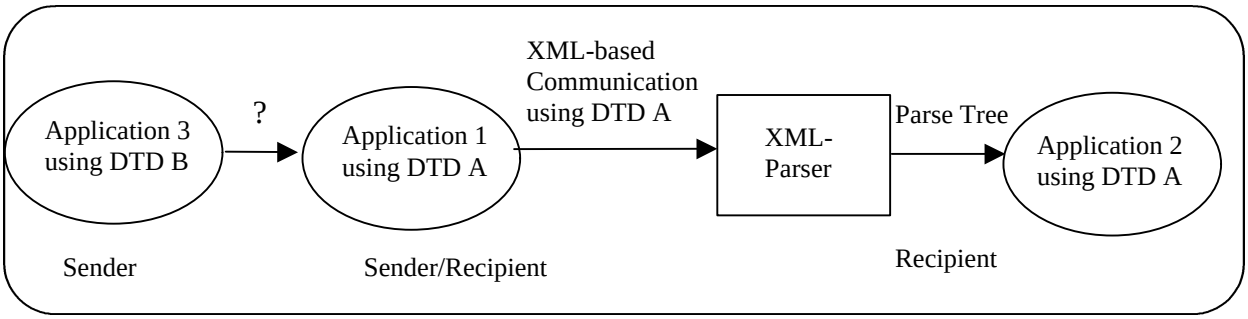


Figure 8 Adding a new Communication Partner

performed:

1. *Reengineering of the original Domain Model from the DTD or XML Schema* (see Figure 9, step 1). Since it is necessary to map semantic entities (and not grammars) it is necessary to reconstruct the original domain models that was used to construct the DTDs.
2. *Establishing mappings between the entities in the domain model*: the concepts and relationships from the domain models have to be mapped to each other. Here techniques developed in the Knowledge Engineering and Database area are often helpful (e.g. [Jannink et al, 1999],[Noy & Musen, 1999], [McGuinness et al, 2000]).
3. *Defining translation procedures for XML Documents*: since XML documents are exchanged, the mappings established in the task 2 above have to be translated in mapping procedures for XML documents (e.g. XSLT definitions for grammars. This is again a high-effort task, since it depends on the particular encoding chosen to construct the initial DTDs.

Although step (2) is already non-trivial, additional effort has to be spent in the particular encoding. The additional effort consists of: the translation of the original domain model into an XML-DTD, the reengineering of the domain model, and the generation of mapping procedures for XML documents based on the established domain mappings (see Figure 9**).** Using a more suitable formalism for the data transfer than pure XML can save much of the additional effort, since then the translations are not necessary anymore.

In a nutshell, this means that XML is very suitable for data interchange between applications that both know about what the data is, but not in situations where the addition of new communication partners occurs frequently.

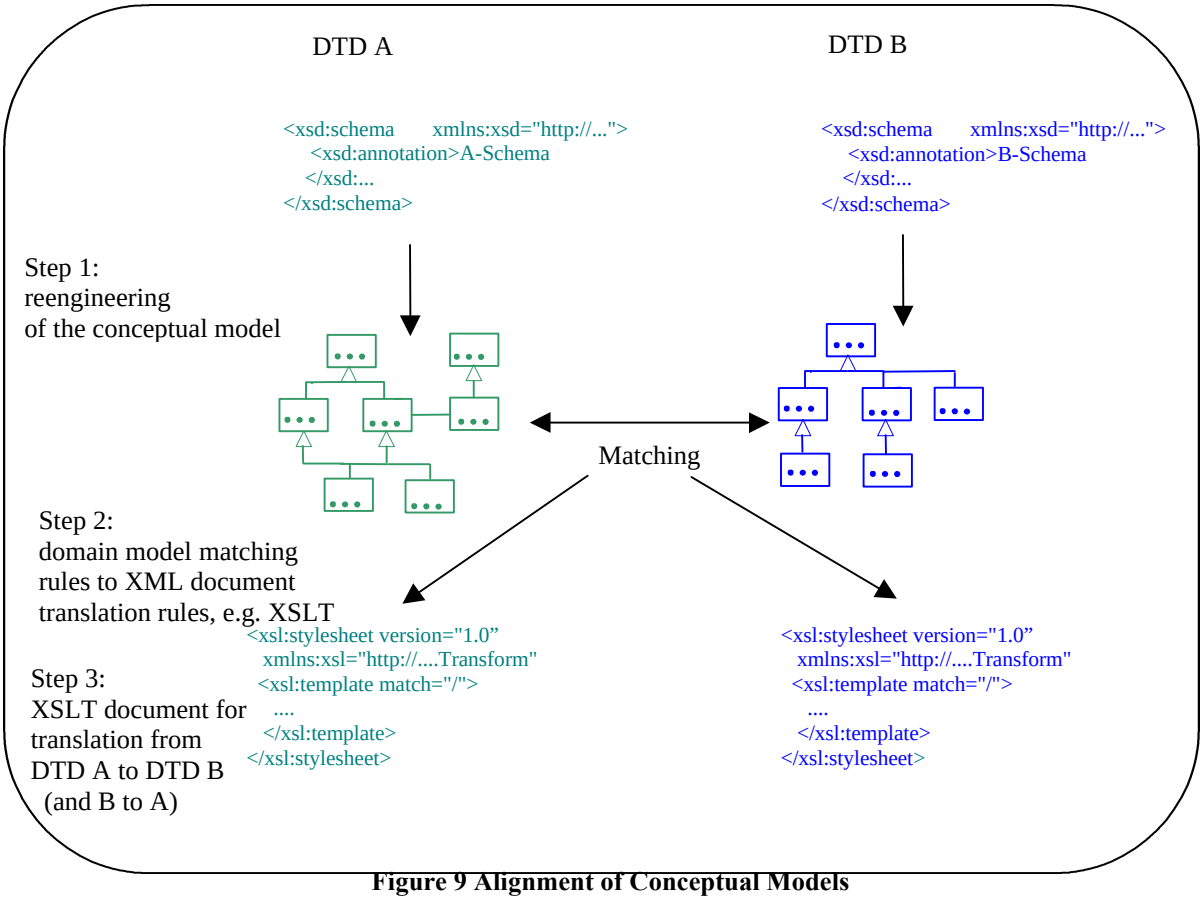


Figure 9 Alignment of Conceptual Models

5.2 Using RDF

RDF defines a nested object-attribute-value structure. The universal expressive power (our first requirement) holds for RDF as well, but is not easy to see. We will demonstrate extension techniques for RDF in the next section. Different RDF parsers are available[4], and are application independent, so requirement 2 is also fulfilled.

When it comes to *Semantic Interoperability,* RDF has significant advantages over XML: semantic units are given naturally through is object-attribute structure: all objects are independent entities. A domain model, defining objects and relationships of a domain of interest, can be represented naturally in RDF, so translation steps, as required when using XML, are not necessary. To find mappings between two RDF descriptions, techniques from Knowledge Representation are directly applicable. *Of course this does not solve the general interoperability problem, that is, finding semantic-preserving mappings between objects. However, the usage of RDF for data interchange raises the level of potential reuse much beyond parser reuse, which is all that one would obtain from using plain XML.*

Furthermore, since RDF describes a layer independent of XML, the RDF model (and software using the RDF model) can still be used, even if the current XML syntax is changed or disappears.

Another way of phrasing the advantages of RDF is as follows. Ideally, we would like a universally shared Knowledge Representation language to support the Semantic Web. For a variety of pragmatic and technological reasons, this ideal is not achievable in practice, and we will have to live with a multitude of meta-data representations. RDF contains as much Knowledge Representation technology as can be shared between widely varying meta-data languages. Furthermore, as we will show in the next two sections, the RDF Schema language is powerful enough to define richer languages on top of the limited primitives of RDF.

[4] e.g. http://WWW-DB.Stanford.EDU/~melnik/rdf/api.html or http://www.ics.forth.gr/proj/isst/RDF/

Since we are concerned with exploiting the semantics of Web-page contents, we should consider the two main approaches in Computer Science to modeling semantics: declarative and procedural semantics.

In a declarative semantics, the meaning of an expression E is given by a mapping to another, well-understood formalism, or by stating the conclusions or properties that follow from E. The meaning of expression E can be understood without reference to any specific computational procedure, which is why this approach to semantics is dubbed "declarative". Using procedural semantics, the meaning of an expression E is given by referring to the behavior that some real or virtual procedure (or program, or machine) will exhibit on E. Often the only way to obtain the meaning of an expression E under such a procedural semantics is to simply execute the procedure on E, and observe its behavior.

This difference between a declarative and a procedural semantics loosely coincides with the difference between the XML and RDF approaches to semantics of Web-pages. As we've argued, an XML expression has no inherent semantics, and its semantics is only determined by the actions that one or more programs undertake on the XML expression (e.g. is tag-nesting interpreted as part-of, or subtype-of, or something else again?). An RDF expression on the other hand has a specific declarative semantics (e.g. the intended meaning of “subClassOf”), and this is specified independent of any processor for RDF expressions (or stated otherwise: any RDF processor must conform to this intended semantics).

Together with the W3C, we stand in the line of along tradition in Computer Science and AI, which argues that the declarative approach to semantics leads to more shareable and extendible information and knowledge sources. We would like to note that the above arguments about XML vs. RDF do not change substantially when regarding XML Schema instead of XML DTD’s as the language in which to specify the structure of XML documents. Readers might be tempted to compare the “type extension” mechanism of XML Schema with the “subclassOf” mechanism of RDF Schema. However, the similarity between these two is only superficial. In fact, the “type extension” mechanism of XML Schema cannot at all be used to model ontological subtypes: in XML Schema, if type T’ is derived from type T, then elements of the derived type T’ are not necessarily members of the original type T. This is in contrast with the “subClassOf” relationship from RDF Schema, where any member of a subclass is by definition also a member of the original super-class. As a result, “subClassOf” can be used to model ontological subtyping, whereas XLM Schema’s type extension cannot.

We have argued that RDF is more suitable for representing formal data on the web. However, the data model of RDF is quite simple, so one might think that it is not universally expressive (like XML is, because XML defines grammars). However, in the following section we show how to express complex data in RDF, and how to exploit the common object structure for interoperability purposes.

6 Enriching RDF

In this section we show how to define sophisticated data models on top of RDF. We do so by enriching RDF with modeling primitives from OIL. However, the techniques are universally applicable. Before we do so, we recall Brachmans distinction of three layers in a knowledge representation system [Brachman, 1979]:

- *Implementation level*: The implementation level consists of, e.g., data structures of a particular implementation of a Knowledge Representation System.
- *Logic level*: this level defines, in an abstract way, the inferences that are performed by the Knowledge Representation System.
- *Epistemological level*: adequate representation primitives are defined at this level. These are usually the primitives used by a knowledge engineer.

Usually the epistemological level is defined by a *grammar*, defining the *language* of interest. However, the representation primitives can also be regarded as an (representation) *ontology,* and so as *objects* of a particular domain. Using this view, domain-modeling techniques are again suitable for defining a knowledge representation

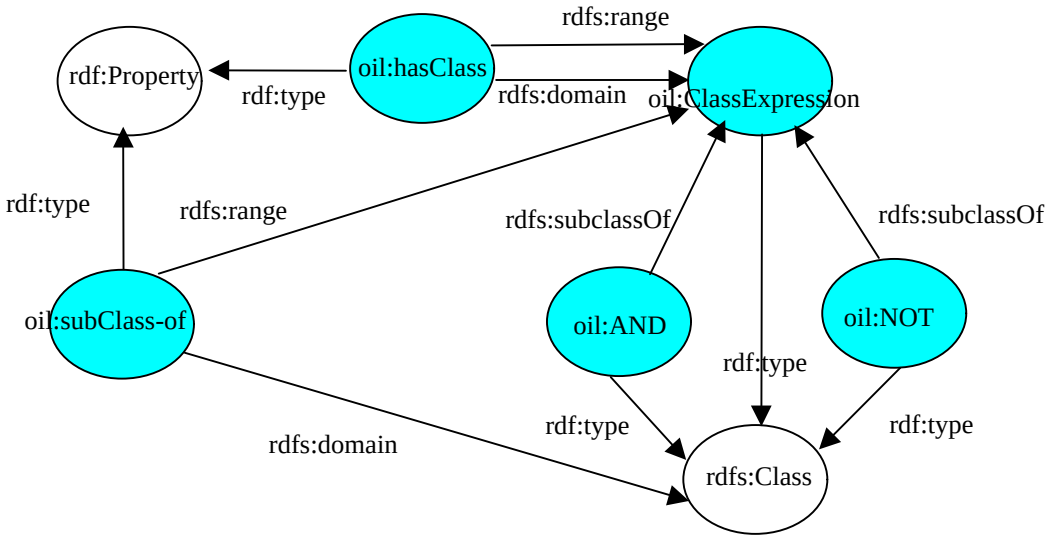


Figure 10 RDF-Schema description for (a part of) OIL

language. As a consequence, the epistemological level of a knowledge representation language is itself given by an *ontology*, which defines the terms of the representation language. Defining an ontology in RDF means defining an RDF Schema, which defines all the terms and relationships[5] of the particular language. Figure 10 shows how the RDF Schema mechanism can be used to define (a part of) OIL. Since OIL extends RDF Schema, the shaded ellipses show which elements must be added to the existing definition of RDF Schema in order to obtain a schema for OIL.

We illustrate the principle with an example from the example ontology (Figure 2). The definition of "herbivore" uses the "subclass-of" modeling primitive. The subclass-of primitive is a relation. This identifies the two arguments of "subclass-of" as objects: the class (*herbivore*) and a (possibly sophisticated) class-expression (*animal AND NOT carnivore*). The view of the class-expression as a object is justified by the fact that the expression "*animal AND NOT carnivore"* indeed defines a new (unnamed) class. Defining the language primitives as an Ontology results in the RDF graph depicted in Figure 10: three classes and two properties are defined. The property "oil:subclass-of"[6] is just the described subclass-of relationship between simple classes and class expressions. The class "oil:ClassExpression" is needed for definitional purposes (e.g. to define domain and range of properties for ClassExpressions). "oil:AND" and "oil:NOT" define class-expressions. The property "oil:hasClass" is an auxiliary property that is needed to connect different class expressions.

Using the above depicted graph, the expression:

[5] One example for a language definition as an Ontology is RDF Schema itself: RDF Schema is defined in terms of RDF Schema (see [Brickley & Guha, 2000]).

[6] Please note that rdfs:subClassOf and oil:subclass-of are different modeling primitives with different semantics. Please see section 7 for a discussion about the differences.

```
class-def defined herbivore subclass-of animal, NOT carnivore
```
can be described in RDF as depicted in Figure 11.

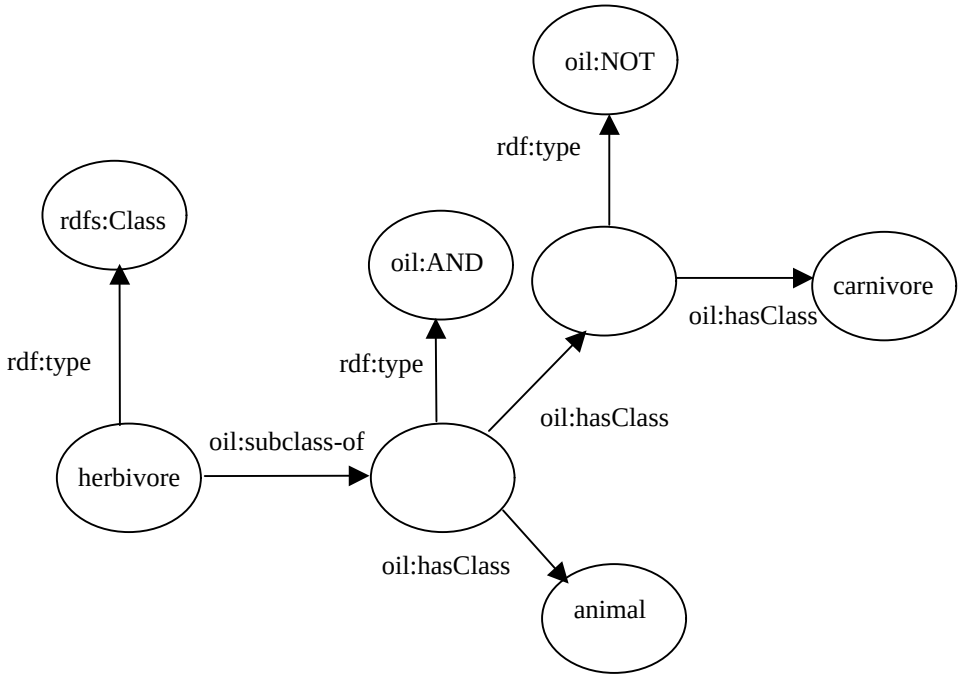


Figure 11 RDF-Graph for the RDFS-OIL definitions

Since every ontology (RDF Schema) uses its own *namespace* [Bray et al, 1999], terms from different ontologies can be mixed in one RDF document without confusion. Since RDF defines a clear object structure, it is possible to make assertions with one language about an object defined in terms of another language. Note that this is not possible in XML: Since the meaning of a particular tag (Object, Attribute, Value etc.) is not defined, nothing can be assumed about the object structure. For example, it would be possible to mix the ontology with behavior statements about different animal-classes using a finite state-automaton language (see ([Melnik et at 2000] for such an approach of mixing different languages).

In the previous paragraphs we have discussed how to establish an (representation) ontology (e.g. about OIL), and how to write a domain ontology (e.g. about animals). However, we have not discussed how to use the domain ontology. The domain ontology itself defines a vocabulary: properties and classes. These properties and classes can be used to write instance information in RDF (e.g. a particular carnivore etc.).

Our proposal can be summarized as follows. We propose a mechanism for encroaching the very simple data-model of RDF with modeling primitives from any ontology language of choice. This mechanism works as follows. Let us use the letter L for the ontology language whose primitives we want to use to describe a particular domain. Then:

- Step 1: Use RDF Schema to describe the modeling primitives from language L (effectively writing the meta-ontology of L in RDF Schema).
- Step 2: Use the resulting RDF Schema document (containing the meta-ontology of L) to describe a specific ontology in L (such as the ontology from Figure 11 above).
- Step 3: Use the RDF Schema documents from steps 1 and 2 to describe instances of the specific L ontology modeled in step 2.

The resulting situation is depicted in Figure 12 below. Notice that our approach suggests additional requirements for an ontology language. Since RDF Schema is already an ontology definition language, compatibility of L with RDF Schema is desirable, because compatibility would enable existing RDF Schema processors to make the maximum use of ontologies defined in L. Also, the same kind of tools are applicable in all steps, so this leads to a very flexible design of customized languages.

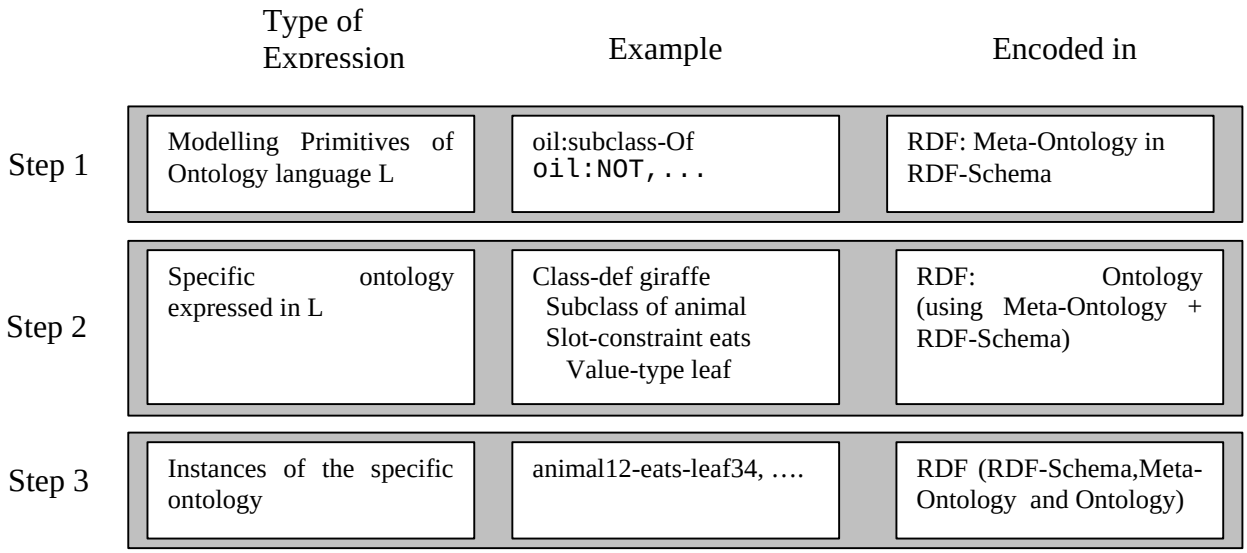


Figure 12 Summary of different Steps

7 Merging an Ontology Language with RDF Schema

This section discusses some of the issues that need to be coped with when defining an extension for RDF-Schema. First, the language needs it's own namespace, since RDF relies on Namespaces and Namespace prefixes (we have chosen the prefix "oil").

Defining an ontology language as an extension of RDF-Schema means that every RDF-Schema ontology is a valid ontology in the new language (i.e., an OIL processor will also understand RDF Schema). However, the other direction is also desirable: defining the new language as close as possible to RDF Schema allows maximal reuse of existing RDF Schema-based applications and tools. However, since the ontology language usually contains new aspects (and therefore new vocabulary, which an RDF Schema processor does not know), 100% compatibility is not possible. In the case of OIL it was necessary to introduce an additional sub-class constructor, since the original sub-class constructor in RDF-Schema only allows primitive classes as subclasses. However, to maintain maximum compatibility with existing applications, it is recommended to use the RDF-Schema vocabulary wherever possible (in the case of OIL this means whenever both arguments are primitive classes). Since RDF follows an Object-Attribute-Value model, a basic design decision for the OIL vocabulary was that every OIL expression is an object. To allow subexpressions, auxiliary attributes are introduced, which do not correspond to any original OIL vocabulary (“oil:hasClass” and “oil:hasProperty”). Another difference is the handling of slot expressions: since slots are separate entities in RDF, it was necessary to treat slot expressions as subclass expressions, as in the underlying description logic framework.

In the case of oil the major integration points of RDF/RDFS and OIL are defined by the abstract OIL class ClassExpression. Furthermore, OIL-slots are realized as instances of rdf:Property or of subproperties of the original rdf:Property. The subslot relationship is also expressed by original RDF-means, namely the rdfs:subPropertyOf relationship. rdf:Property is enriched by a number of properties that specify inverse and transitive roles and cardinality constraints, not originally possible in RDF/RDFS (for details, see [Horrocks et al., 2000]).

Primitive class definitions are inherited from the original rdfs:Resource from the RDF-Schema specification. Primitive classes (as well as rdfs:Class) can be related with arbitrary class expressions via subclass or equivalence relationships. By doing this, already existing classes from RDFS-vocabularies can be accessed and refined in OIL descriptions.
To define subclass or equivalence relations between arbitrary class-expressions, we introduced ClassConstraint, which encapsulates the constraints. The definition of a proprietary subClassOf-property is necessary to allow for cyclical subClassOf-definitions, which are prohibited by the RDFS-specification. The following table contains some of the OIL mappings. A complete mapping is given in [Horrocks et al., 2000].

OIL	**RDF VOCABULARY**
Class-def	rdfs:Class
Subclass-of	rdfs:subClassOf (single parent class) oil:subclass-of (parent is class expression)
Slot constraint	Slot constraints are subclass expressions in RDF-Oil
AND, ","	oil:AND
NOT	oil:NOT
Has-value	oil:has-value

8 Conclusions

In this paper we have argued that semantic interoperability will be a *sine qua non* for the Semantic Web, and that such semantic interoperability must be achieved by exploiting the current RDF proposals, and *not* by exploiting XML labeling. The RDF data-model is sound and makes approaches from AI and Knowledge Engineering for establishing semantic interoperability directly applicable. Furthermore, it is universally extensible. We have shown the feasibility of our proposal by applying it to a particular ontological modeling language, and we have argued that a similar strategy should apply to any knowledge modeling language.

The arguments in this paper are an important message to at least two different communities.

The Web community is currently regarding XML as the most important step towards semantic integration. We have argued why this cannot be true in the long run, and why RDF is a much better platform for this.

The AI community is currently very much interested in applying many of its techniques to the Web. We have shown a generic method for Web-enabling arbitrary Knowledge Representation languages. This is an important step towards the realization of the dream of the Semantic Web.

Bibliography

[**Barker, 1990**] R. Barker: Entity Relationship Modelling Addison-Wesley Pub Co; ISBN: 0201416964, 1990.

[**Benjamins et al, 1999**] V R. Benjamins, D. Fensel, S. Decker, and A. Gomez-Perez: (KA)2: Building Ontologies for the Internet: a Mid Term Report. *International Journal of Human-Computer Studies (IJHCS),*51, 687--612, 1999.

[**Berners-Lee, 1999**] T. Berners-Lee: Weaving the Web. Harper, San Francisco, 1999.

[**Biron & Malhotra, 2000**] P. V. Biron, A. Malhotra (eds.): XML Schema Part 2: Datatypes, W3C Working Draft 07 April 2000, http://www.w3.org/TR/2000/WD-xmlschema-2-20000407/

[**Brachman, 1979**] R.J. Brachman: On the Epistemological Status of Semantic Networks, in Associative Networks: Representations and Use of Knowledge by Computers, Findler, N.V: Academic Press, 1979, pp. 3-50.

[**Bray et al, 1998]** T. Bray, J. Paoli, and C.M. Sperberg-McQueen (eds.): Extensible Markup Language (XML) 1.0, W3C Recommendation 10-February-1998, http://www.w3.org/TR/REC-xml

[**Bray et al, 1999]** T. Bray, D. Hollander, and Andrew Layman (eds.): Namespaces in XML, World Wide Web Consortium Recommendation, 14-January-1999, http://www.w3.org/TR/REC-xml-names/

[**Brickley & Guha, 2000**] D. Brickley, R. Guha (eds.): Resource Description Framework (RDF) Schema Specification, W3C Candidate Recommendation 27 March 2000, http://www.w3.org/TR/2000/CR-rdf-schema-20000327

[**Clark, 1999**] James Clark (ed): XSL Transformations (XSLT), W3C Recommendation 16 November 1999, http://www.w3.org/TR/xslt

[**Davidson, 2000**] James Davidson: Java API for XML Parsing, Version 1.0 Final Release, http://java.sun.com/aboutJava/communityprocess/final/jsr005/index.html

[**Erdmann & Studer, 1999**] M. Erdmann, R. Studer: Ontologies as Conceptual Models for XML Documents. In: *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, Banff, Canada, October 1999.

[**Gomez Perez & Benjamins, 1999**] A. Gomez Perez, V. R. Benjamins: Applications of Ontologies and Problem-Solving Methods. In AI-Magazine 20(1):119-122, Spring 1999.

[**van Heijst et al, 1997**] van Heijst, G., Schreiber, A.T., and Wielinga, B.J. (1997). Using Explicit Ontologies in KBS Development, in Int. Journal of Human-Computer Studies, Vol.46, 1997, pp.293-310.

[**Horrocks et al, 2000**] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. Van Harmelen, M. Klein, S. Staab, and R. Studer: The Ontology Interchange Language OIL. Technical Report, Free University of Amsterdam, 2000. http://www.ontoknowledge.org/oil/

[**Hoschka, 1998**] P. Hoschka (ed.): Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, W3C Recommendation 15-June-1998, http://www.w3.org/TR/REC-smil

[**Jannink et al, 1999**] J. Jannink, P. Mitra, E. Neuhold, S. Pichai, R Studer, G. Wiederhold: An Algebra for Semantic Interoperation of Semistructured Data; in 1999 IEEE Knowledge and Data Engineering Exchange Workshop (KDEX'99), Chicago, Nov. 1999.

[**Lassila & Swick, 1999**] O. Lassila, Ralph Swick (eds).: Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, http://www.w3.org/TR/REC-rdf-syntax/

[Melnik et al, 2000] S. Melnik, H. Garcia-Molina and A. Paepcke: A Mediation Infrastructure for Digital Library Services, Proc. of the ACM Digital Libraries Conf., June 2000

[**McGuinness et al, 2000**] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder: The Chimaera Ontology Environment.Proceedings of the The Seventeenth National Conference on Artificial Intelligence (AAAI 2000), Austin, Texas, July 30 - August 3, 2000.

[**Noy & Musen, 2000**] N. Fridman Noy, M. Musen: An Algorithm for Merging and Aligning Ontologies: Automation and Tool Support. In the Proceedings of the Workshop on Ontology Management at Sixteenth National Conference on Artificial Intelligence (AAAI-99), Orlando, FL.

[**Page-Jones & Constantine, 1999**] M. Page-Jones, L. L. Constantine: Fundamentals of Object-Oriented Design in UML, Addison-Wesley, ISBN: 020169946X, 1999.

[**Staab el al, 2000**] St. Staab, M. Erdmann, Alexander Maedche, Stefan Decker: An extensible approach for Modeling Ontologies in RDF(S). Technical Report 401, AIFB, University of Karlsruhe, March 2000.

[**Thompson et al, 2000**] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn (eds.): XML Schema Part 1: Structures W3C Working Draft 7 April 2000, http://www.w3.org/TR/2000/WD-xmlschema-1-20000407/

[**Uschold & Gruninger, 1996**] M. Uschold & M. Gruninger: Ontologies: Principles, Methods and Applications", The Knowledge Engineering Review, Vol 11:2, 93-136, 1996.