

# The Yin/Yang Web: XML Syntax and RDF Semantics

Peter Patel-Schneider  
Bell Laboratories

pfps@research.bell-labs.com

Jérôme Siméon  
Bell Laboratories

simeon@research.bell-labs.com

## Abstract

XML is the W3C standard document format for *writing* and *exchanging* information on the Web. RDF is the W3C standard model for *describing the semantics* and *reasoning about* information on the Web. Unfortunately, RDF and XML—although very close to each other—are based on two different paradigms. We argue that in order to lead the Semantic Web to its full potential, the syntax and the semantics of information needs to work together. To this end, we develop a model theory for the XML XQuery 1.0 and XPath 2.0 Data Model, which provides a unified model for both XML and RDF. This unified model can serve as the basis for Web applications that deal with both data and semantics. We illustrate the use of this model on a concrete information integration scenario. Our approach enables each side of the fence to benefit from the other, notably, we show how the RDF world can take advantage of XML query languages, and how the XML world can take advantage of the reasoning capabilities available for RDF.

**Keywords:** Semantic Web, Web Languages, Data Models, Model Theory, XML, RDF.

**Approximate Word Count:** 7800 words

## 1 Introduction

As the W3C standard document format for *writing* and *exchanging* information on the Web XML [6] is mostly concerned about syntax. However, syntax does not make sense without semantics, and many recent activities aim at adding more semantic capabilities to XML. Most notably, the XML Infoset [10] offers an abstract information model for XML; XML Schema [29] allows users to describe XML vocabularies, structures, and relationships; and XQuery [8] allows users to ask questions, manipulate, or reason about XML documents.

As the W3C standard model for *describing the semantics* and *reasoning about* information on the Web RDF [18] RDF is mostly concerned about semantics. However, semantics is not very useful in a computer system without a syntax, and many recent activities aim at providing a syntactic grounding for RDF. Most notably, RDF uses an XML serialization; and several query languages for RDF [17, 24] have already been proposed as well.

Many XML application scenarios require the use of semantic tools. For instance, data integration relies on the ability to build a common ontology between multiple sources [20, 3, 1]. Development of a domain's XML dialect (e.g., ebXML or VoiceXML [26]) is greatly simplified by the use of modeling methodologies based on rich semantic descriptions [7, 21]. Many RDF application scenarios require the access to existing information sources that are providing XML interfaces. For instance, semantic descriptions for Web services cannot be made without taking into account the format in which messages will be exchanged between these services.

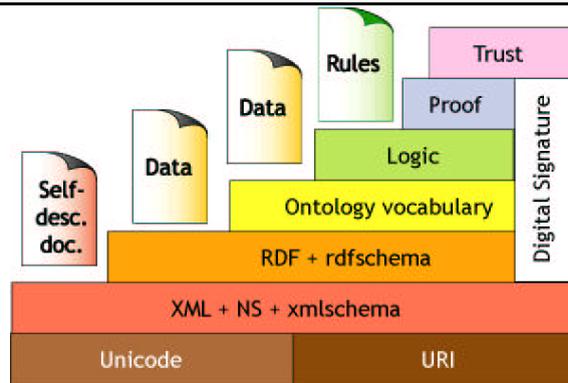


Figure 1: The Semantic Web Layer Cake

Indeed, the coming semantic Web is usually envisioned as a layer cake, like the one shown on Figure 1<sup>1</sup>, in which the semantic layer is not independent from, but *is relying on* the syntactic layer. Unfortunately, XML and RDF, which are respectively supposed to form the ground for the syntactic (or data), and semantics (or meaning) layer of the Web, are currently based on different models, and are developed within separate activities<sup>2</sup>. As a result, very few tools can actually be used jointly between XML and RDF.

We argue that syntax and semantics are the Yin and the Yang of the Web, and should be complementary to each other rather than independent—or worse, incompatible—from one another. Users facing the above applications need to deal with syntax and semantics in a unified way. We argue that rather than two Webs: one Syntactic Web and one Semantic Web, these users need one Web that encompasses both syntax and semantics: the Yin/Yang Web.

In this paper, we propose an architecture for a unified Web based on a common model theory for XML and RDF. Although RDF and XML have been two distinct activities, we will see that there is enough commonality between them to design, and implement, such a common model. There are several difficulties in the way though, due to the fact that RDF and XML viewpoints are not fully compatible. Notably, XML is ordered while RDF is not, XML uses a tree model while RDF uses a graph model, RDF distinguishes between classes (e.g., a company) and properties (e.g., the name of a company) while XML does not (e.g., company and names would both be elements). Our main contribution is a model theory that encompasses XML and RDF model properties in order to be able to represent, and reason about, both uniformly. We call this model the Yin/Yang model. As soon as a common model exists, many more interesting but difficult questions arise. In particular, how XML Schema and RDFS interact, how one can query both XML and RDF, etc. A complete solution would indeed address these, but is still beyond the scope of that paper. We will show, however, how the Yin/Yang model enables some new exciting possibilities with respect to schema, typing and querying.

More precisely, we make the following technical contributions:

- We develop an integrated model for XML and RDF based on a model theory for the XQuery 1.0 and XPath 2.0 Data Model [12].
- We explain the relationship between that model theory and previous RDF/S model theories. Most of the existing semantics is captured, and we point out and explain the few existing discrepancies.

<sup>1</sup>That figure was extracted from a presentation by Eric Miller at the 5th European Conference on Research and Advanced Technology for Digital Libraries, but was probably first presented by Tim Berners-Lee.

<sup>2</sup>One could also facetiously note that they are represented in two distinct WWW'2002 tracks!

- We show how one can perform semantic reasoning in that integrated model theory. Note that the semantics of RDFS only supports limited reasoning capabilities. Although we restrict ourselves to RDFS in that paper, moving toward more expressive models, e.g., DAML+OIL, would certainly make the Yin/Yang approach and reasoning features even more useful in practice.
- We show several new possibilities offered through that integrated model theory. Notably, we give some ideas at how to capture some aspects of the XML Schema in the model theory for reasoning purposes. We also show how one can apply XQuery on a mix of RDF and XML descriptions, and we explain how XML querying can benefit from an RDF reasoner.
- Finally we describe our implementation of that model theory on top of the XQuery data model. Because our model relies on the XQuery data model, building a Yin/Yang implementation with a reasoner can be done in a much simpler fashion than previous RDF implementation approaches.

**Ins and Outs** The Yin/Yang model provides access to both data structures, through the XQuery data model, and their corresponding semantics, through the model theory.

On the syntactic side, applications have full access to the XQuery data model, hence there is no loss of information for data-oriented applications. Applications can even take advantage of *some* of the semantic-based features of the model by treating XQuery data model constructs in accordance with the meaning provided by the model theory. As syntactic processing is done entirely within the XML framework, the RDF parsetype extension is not handled. Also, the RDF shorthand that is inconsistent with XML is treated in the XML fashion, not in RDF fashion.

On the semantic side, the model theory integrates the two different world-views of XML and RDF with minimal loss of information. The model theory allows for both the ordered view of documents from XML and the unordered view of information from RDF. It does not require the RDF distinction between classes and properties, allowing arbitrary XML, but the distinction between classes and properties, if present, can be recovered from the model theory. It includes a complete treatment of RDF typing, where type links are treated the same as other links, even when there is no distinction between classes and properties and incorporates XML names into RDF types. It allows for the identification of nodes, turning the tree view of XML into the graph model of RDF.

The model theory does, however, eliminate as irrelevant XML comments and process instructions as well as the lexical form of typed text nodes, and does not distinguish between XML elements and attributes. Further, it does not handle most of the XML Schema structural information, at least for now.

**Related work** We should point out that this discussion about the relationship between syntax and semantics is not new. Several attempts have been made to provide a unified view of XML and RDF. Tim Berners-Lee [4] was one of the first to point out the reasons for the differences between XML and RDF. Also we acknowledge the fact that RDF and XML serve different purposes, we believe this difference must not prevent syntactic and semantics interoperability, which is an important user need. Sergey Melnik [23] created a version of RDF that can handle arbitrary XML, but uses information on how to handle the parts XML constructs that don't fit well into the RDF model. Harold Boley [5] has a data model that can be used for both XML and RDF. However, his approach requires changes to XML to unify it with RDF. He also stops at the data model and does not proceed on to a model theory. Fundulaki et al [14] acknowledge the need for integrating syntax and semantics, but require the development of user-defined rules in order to cope with the discrepancy between XML and RDF. Robie et al [28] also addresses the need for applications to query semantics, but maps the syntax into the semantics, hence requiring the need to hard-code some of the semantic aspects in functions. To the best of

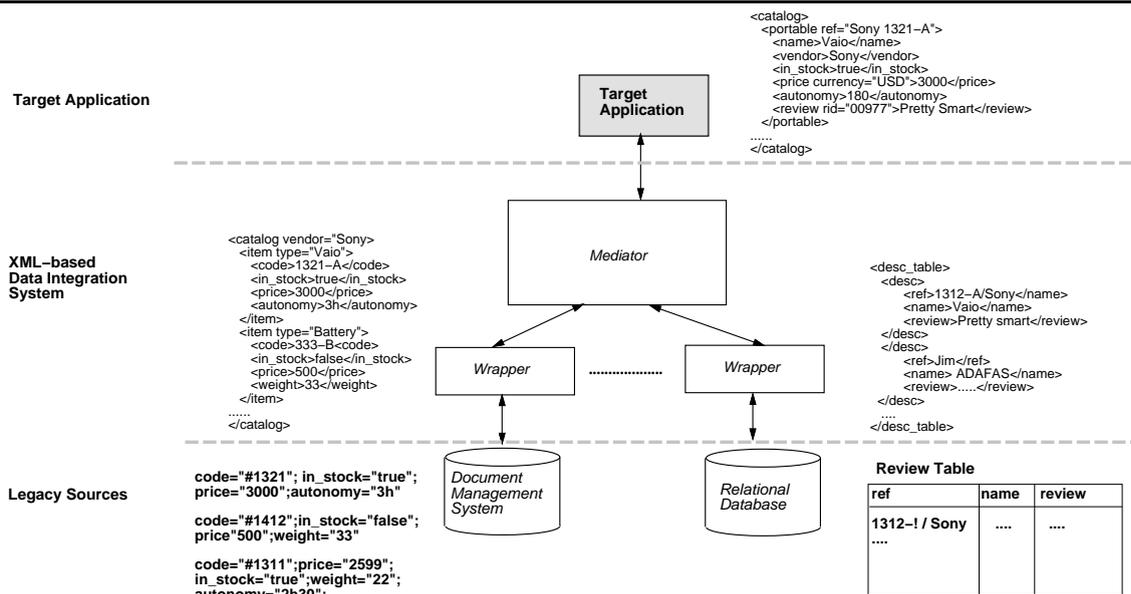


Figure 2: XML-based data integration

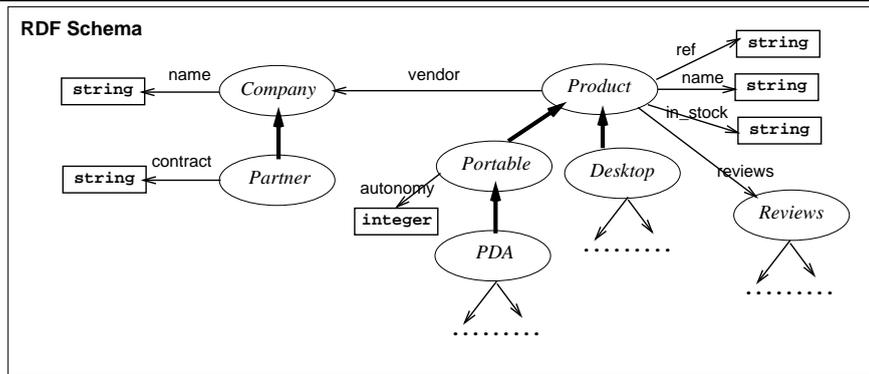
our knowledge, our approach is the first one that allows XML data access and RDF semantic reasoning in a common framework.

## 2 Using syntax and semantics on the Web

We start by giving an application scenario that illustrates the need for tight integration between syntax and semantics on the Web. In this scenario, a computer retailer wants to build its store's information system from multiple hardware and software vendors catalogs (e.g., Sony, IBM), and a product's review database maintained by a third party (e.g., <http://www.epinions.com/>).

Information integration is one of the most important use of XML. Recently, a number of companies [11, 25] and research projects [27, 13, 9, 2, 22] have been working on building XML-based data integration systems. These systems rely on the ability to represent any kind of legacy information in XML, and on XML high-level languages, such as XQuery [8], to merge their information under a common schema. Figure 2 shows the architecture of a typical data integration system, where *wrappers* are used to map legacy information into XML, and a *mediator* is used to perform the integration. We refer the reader to the related work at how to use XML query languages to specify such data integration [27, 13, 9, 2, 22].

Still, information integration cannot be fully solved without addressing semantic issues. For instance, one needs to define a global ontology for all information involved in the sources, and also to understand how similar information is represented in different ways on each source [20, 3, 1]. In our scenario, the retailer might want to organize his data according to a product hierarchy where `Product` would be the root of the hierarchy, `Portable` and `Desktop` would be categories, `PDA` and `Laptop` be sub-categories of `Portable`, etc. Each products would have a name and a reference number, while portables would have an autonomy. In order to work on the web, modern semantic integration platforms [14] are relying on RDF/S, to describe such an ontology. However, each vendor's catalog provides a different set of information for their products, for instance the Sony catalog indicates the autonomy of each laptop, while the IBM indicates a battery reference. Also the classifications within the catalogs differ, for instance Sony has separate categories for laptop and palmtop, while



```

<catalog>
  <portable ref="Sony 1321-A">
    <vendor co="Sony"/>
    <name>Vaio</name>
    <in_stock>true</in_stock>
    <price currency="USD">3000</price>
    <autonomy>180</autonomy>
    <review rid="00977">Pretty Smart</review>
    <accessories>
      <battery ref="IBM X111"/>
      <docking_station ref="IBM X112"/>
      <battery ref="Sony 333-B"/>
    </accessories>
  </portable>
  <PDA ref="Compaq 4XDF">
    <name>iPAQ</name>
    <vendor co="IBM"/>
    <in_stock>inorder</in_stock>
    <price currency="USD">500</price>
    <weight>33</weight>
  </PDA>
  ....
</catalog>
<companies>
  <company co="Sony">
    <name>Sony</name/>
    <tel>555-13-13</tel>
  </company>
  <partner>
    <name>IBM</name/>
    <tel>555-13-13</tel>
    <contract>013</contract>
  </partner>
  ...
</companies>

```

Figure 3: RDF Schema vs. XML Data

IBM has a single category for portable computers, etc. This often implies that the resulting global ontology will be a fairly complex hierarchy with many different classes, and properties for these classes. Understanding the relationship between these classes then becomes essential.

As sources export their data in XML, integration into that common ontology is more easily specified using an XML language like XQuery. This results in a mismatch between the data produced in XML and the ontology description in RDF/S. Figure 3 shows an example of an RDFS graph describing the target semantics, and of an integrated XML document generated from the sources. A first use of a unified XML and RDF model is to be able to relate the integrated XML information to its intended meaning in RDF. In section 4 we will see how our model theory can be used to understand the relationship between the XML document and its intended semantics in RDF.

It is important to note that the XML document resulting from integration reflects some of the semantic hierarchy of the ontology. Assuming the target application wants to access all the computers that have certain characteristics, one should be able to write an XML query to do that. However at the XML level, the connection between the element names (e.g., *desktop*, *laptop* and *computers*) would not be available to an XML query processor without knowledge of the semantics layer. In section 5 we will see how one can use our model theory to perform such semantics querying for XML documents.

## 2.1 Other Scenarios for the Yin/Yang Web

We took the data integration example for ease of exposure and the striking need for interaction between the semantic and the data worlds. But there is no shortage of important applications for the Yin/Yang Web.

**XML Dialects** The development of domain-specific dialects is an important activity area around XML. Witness of that activity, the Oasis consortium [26] hosts several dozens of dialects that describe information from almost all possible domains of human knowledge (e.g., music, theology), industry segments (e.g., car manufacturing, voice interface), or specific transversal activities (e.g., Web presentation with XHTML, calendars).

These dialects make possible for communities to share information in a common syntax. Yet, this common syntax is only a means to share information with an agreed upon semantics. It is therefore essential to develop that dialect based on a mutually shared understanding. Semantic modeling tools [7, 21] provide services to define ontologies for a given application or domain. After the modeling phase, this usually results in a concrete XML Schema, in which part of the semantics is either lost (e.g., the distinction between an entity and a relationship) or deprecated (e.g., typed references to object can be preserved only as integrity constraints). In the Yin/Yang Web, the applications can be given full access to the data (XML), structure (XML Schema), but also its complete intended semantics (RDF/S).

**Web services** High-level service descriptions, like DAML-S, are generally written in a semantic language, such as RDFS (or DAML+OIL). Lower-level activities, including passing messages between services, instead use XML, possibly including XML Schema validation. In the Yin/Yang Web, these two levels can be firmly joined and clearly related to each other.

## 3 The Yin/Yang Approach and Processing Model

### 3.1 Processing information on the Web

When an application gathers some information from the Web, this information is usually accessed as a file, most often in XML syntax. This file goes through several stages before the actual meaning of the information is accessible to the application. According to current W3C architectures, initial stages fall into the category of producing an abstract syntax tree for a document, resulting in one of the various data models, such as the XQuery data model [12] for data or documents, or the RDF model [18], for semantic information. Each stage in these processes produce an abstraction of the original document, and can both remove information that they deem irrelevant (such as non significant white space) and add implied information (such as typing information). Although they perform very similar tasks, there are significant differences between the initial processing stages of XML documents and of RDF documents.

- After parsing, XML documents are usually validated against a DTD or XML Schema. As well as checking that all and only the indicated information is present, this process adds several important information to the documents, such as default values, datatypes, some derivation information, etc. The resulting so-called post-schema validation infoset can then loaded into the XQuery data model for querying. XQuery data model structures provide a tree representation of the XML document and can be accessed by applications via a functional interface.
- Parsing RDF documents results in an RDF graph structure, similar to the XQuery data model, but a graph instead of a tree. This graph structure is given semantics by means of a model theory [16]. Many RDFS constructs, such as its subclass property, result in constraints in the model theory, such as requiring

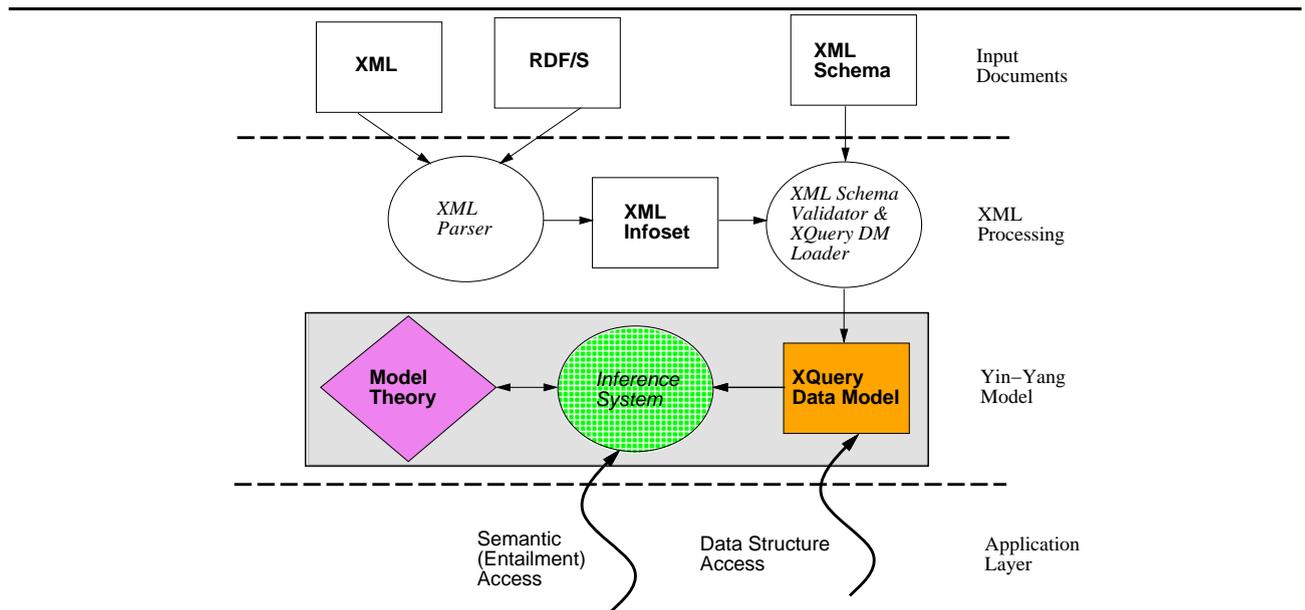


Figure 4: Yin/Yang model

that certain kinds of relationships are transitive. The model theory does not specify how these semantic constraints are to be implemented. The model theory also does not provide a data structure that can be accessed. On the other hand, access to the semantic information in a document can be performed via software that implements logical operations on the the model theory, such as entailment.

The existence of these two distinct models, along with distinct processing stages for XML and for RDF is the main reason that prevents applications from dealing with information both at a data level and at a semantic level. In addition, due to the similarity of processing in both cases, there is important duplication of work. Our approach is to reuse as much XML processing as possible before adding semantic layers. In a nutshell, we first build an instance of the XQuery data model for XML, RDF and even RDFS documents. This already supports structural manipulation for data-oriented applications over XML, RDF or RDF schema information. Then, we use that data model in a model theory in order to support semantic reasoning. The stages in the Yin/Yang model are described on Figure 4.

This processing architecture has the following advantages. First, it builds on existing XML processors as much as possible, which reduces the work required to develop a semantic processor. It provides tight coupling between the data layers and the semantic layers. It allows data applications to benefit from semantic reasoning. It allows semantic-based applications to access data consistently. Before we explain more about the Yin/Yang model itself, we briefly give some background information on the XQuery data model and on model theory.

### 3.2 From Syntax to Data Model

For our purposes a data model is a collection of data types that can be used to construct an abstract view of a web document (or collection of documents), along with functions that allow access to the information contained in values belonging to these types. Data models generally also have construction functions, but we will not be talking much about the construction of the data model and so will mostly ignore them.

The XQuery 1.0 and XPath 2.0 Data Model (henceforth “data model”) represents an XML document as a collection of nodes of different kinds, arranged in a tree. For example, an element node in the data model, which

corresponds to an XML element information item has accessors for its name, parent, namespaces, attributes, and children, as one would expect. The data model includes other types of nodes, such as attribute nodes, namespace nodes, comment nodes, and text nodes.

One reason that we are using this data model is that it contains support for DTD and XML Schema validation. To access information gathered from validation, the data model provides accessors for the XML Schema type associated with many kinds of nodes, and, if the type is a simple XML Schema datatype, to the sequence of typed values resulting from XML Schema processing of the text within the node (for element nodes) or the value of the node (for attribute nodes).

So the first phase of the processing of an XML document or collection of documents, so far as we are concerned, results in the a tree or forest of nodes in the data model. This processing includes not only the parsing of the document, but also DTD and XML Schema validation and decoration.

### 3.3 From Data Model to Semantics

The next phase of our approach is to move from the data model into the semantic realm. We do this by adopting conventions from model theory, a branch of mathematics that is used to provide meaning for many logics and representation formalisms, and has recently been applied to several web-related formalisms, namely RDF [16]. and DAML+OIL [30].

One of the particularities of our approach is the choice of relying on two distinct paradigms: data model for data, and model theory for semantics. There are several fundamental differences between data models and model theory approaches that justify that choice.

**Information retention:** Data models tend to retain almost all of the information in the input document, such as comments and the exact form of typed values. In model theory, on the other hand, there is a decision made on just what information to retain, and the information is ignored. It is typical in model theories to use sets instead of sequences and thus to ignore the order in which information is presented.

**Direction of flow:** In data models there is a process of generating a data model from an input document and thus the result is constructed from the input. In model theory, on the other hand, the interpretations are simply mathematical objects that are not constructed at all. Instead there is a relationship between syntax constructs and interpretations that determines which interpretations are compatible with a document. Generally there are many interpretations that are so compatible, not just one.

**Schema vs. Data:** Data model approaches usually make a fundamental distinction between schema and data. In model theory, both schema and data are part of a model on which one can perform reasoning. As shown on Figure 4, this allows us to deal with both RDF and RDFS in a common way, while some aspects of XML Schema will remain out of the scope of the inference system.

### 3.4 Using model theory for reasoning

In a model theory, we end up with not just a single interpretation or model, but instead a collection of interpretations or models. These models can be thought of as the different ways that the world can be and still be compatible with the information in the input document.

What is generally done next in model theory is to define a relationship between input syntax called entailment, which can be read as “follows from”. A collection of sentences (or documents), called the antecedents, entails a sentence (or document), called the consequent, if every interpretation that is a model for each element of the collection is also a model for the consequent. This relationship can be read as “if the world is compatible with each each antecedent, then it is also compatible with the consequent” or “if each antecedent is true then

so is the consequent.” It is possible to think of entailment as a version of relational retrieval where the query specifies explicit values for all elements of the tuple, i.e, there is at most one possible answer. Generalizations of entailment have been used that allow open variables in the consequent, resulting in a version of entailment close to retrieval.

Note again, that the *kind of* reasoning that can be achieved depends on the expressive power of the semantics description language. Again, in that paper we are limited in functionality by the expressiveness of RDFS, but we intend to investigate more expressive formalisms in the future.

## 4 The Yin/Yang Model Theory

### 4.1 Processing of input documents

Our model theory starts with a tree structure composed of a set of nodes,  $N$ , in the XQuery 1.0 and XPath 2.0 Data Model [12]. This tree structure corresponds to an XML or RDF document (or collection of documents) that have already been through a serious amount of processing, notably parsing and schema validation. Each tree node,  $n$ , is assumed to have a mapping,  $UTS(n)$  which is the map from strings to qualified names, given the namespace declarations in scope at the node.

**Example** Consider the following pieces of an XML document (actually RDF with an XML Schema datatype extension):

```
<Laptop rdf:about="Vaio505G">
  <manufacturer rdf:resource="Sony"/>
  <price xsi:type="xsd:integer">3000</price>
</Laptop>

<rdf:Description rdf:about="Sony" home="www.sony.com">
  <rdf:type>
    <rdf:Description rdf:about="Company">
  </rdf:type>
</rdf:Description>
```

This document is parsed and then loaded into the XQuery data model. This results in the following data set, written here in a simple data structure where nodes are represented as tuples containing the relevant bits of information prefixed with a node identifier:

```
1:<Laptop,attributes=[ 2:<rdf:about , "Vaio505G">],
  elements=[ 3:<manufacturer,attributes=[ 4:<rdf:resource, "Sony">]>,
    5:<price,attributes=[ 6:<xsi:type, "xsd:integer">],
      elements=[ 7:<" 3000">]>]>
8:<rdf:Description,
  attributes=[ 9:<rdf:about, "Sony">, 10:<home, "www.sony.com">],
  elements=[ 11:<rdf:type,
    elements=" [ 12:<rdf:Description,
      attributes=[ 13:<rdf:about, "Company">]>]>]>
```

This structure can then be accessed using the XQuery data model accessor operations on nodes, can be queried using XQuery, etc.

## 4.2 Resources, Names, Data Values, and Datatypes

The Yin/Yang model theory assumes a universe of resources and data values. For simplicity, we make the assumption that QNames are suitable as RDF identifiers, but readers could read the document substituting RDF identifiers for QNames. (A full treatment of RDF identifiers is somewhat messy so a simpler treatment is used here.) The construction `rdf:name` below refers to the QName with local part *name* and URI `http://www.w3.org/1999/02/22-rdf-syntax-ns` (the rdf URI).

**Definition 4.1** We call  $L$  the lexical space of strings, and  $U$  the value space of QNames, i.e., pairs of URIs and local parts. We call  $DT$  the subset of  $U$  corresponding to XML Schema primitive datatypes, and  $DV$  the union of the value spaces of the XML Schema primitive datatypes. In RDF elements of  $DV$  are generally called literals. The function  $DTC : DT \rightarrow \mathcal{P}(DV)$ , (where  $\mathcal{P}$  is the powerset operator) maps XML Schema primitive datatypes to their value spaces and  $DTS : DT \rightarrow (L \rightarrow DV)$ , maps XML Schema primitive datatypes to their lexical to value maps. We define the union of the datatype mappings  $XTS : L \rightarrow \mathcal{P}(DV)$ , where  $v \in XTS(l)$  iff  $v = DTS(dt)(l)$  for some XML Schema datatype  $dt$ .

## 4.3 Yin/Yang Interpretations

Interpretations are the essential component of our model theory. An interpretation corresponds to one possible way the world can be, hence encoding a certain meaning for the information manipulated by an application. Interpretations give information about resources and related resources through relationships and semantic constraints. For instance, one resource may be a *Laptop*, related to another resource *Sony* through a *manufacturer* property. We define a notion of interpretation that is suitable for both XML and RDF documents, through the XQuery data model.

**Definition 4.2** An interpretation  $I$  is a six-tuple,  $\langle R, E, EXT, CEXT, O, S \rangle$ , where:

- $R$  is a set of resources,
- $E$  is a set of relationships,
- $EXT : E \rightarrow R \times (R \cup DV)$  maps relationships to the resources they relate,
- $CEXT : R \rightarrow \mathcal{P}(R \cup DV)$  maps class resources to their extensions,
- $O : R \rightarrow \mathcal{P}(E \times E)$  provides a local order on the relationships, and
- $S : U \rightarrow R$  is a partial map from QNames to resources.

An interpretation can be thought of as a multigraph with an ordering on the edges. Resources ( $R$ ) form the nodes of the graph. Edges of the graph are formed from relationships ( $E$ ) and  $EXT$ . For instance, a relationship  $e1 \in E$  with  $EXT(e1) = \langle Vaio505G, manufacturer \rangle$  indicates that the resource *Vaio505G* is related to the resource *manufacturer*, and a relationship  $e2 \in E$  with  $EXT(e2) = \langle manufacturer, Sony \rangle$  indicates that the resource *manufacturer* is related to the resource *Sony*. We remark that there is no distinction at this point between *Sony*, as an instance of a class, and *manufacturer*, which is a property. This allows the model theory to represent arbitrary XML documents, while we will see we can still recover the traditional RDF semantics.

$S$  provides a mapping between syntax (QNames) and their denotation (resources).  $S$  gives a means to identify these entities using QNames. There is no requirement that all resources have corresponding QNames, nor is there a requirement that QNames are all mapped to different resources.

$CEXT$  provides typing information for resources. For instance, if *Sony* is in  $CEXT(Company)$  then the resource *Sony* is of type *Company*. Loosely speaking, in RDF terms  $CEXT$  serves for both property and class extensions. Or, considered another way, a property is presented as a type whose values and related tuples identify arcs in the traditional RDF graph structure.

Finally,  $O$  provides ordering information between the relationships that are related to a common resource. This information is not usually part of RDF model theories [16], but it is important to capture document order in XML documents. We add one special attribute to RDF,  $\text{rdf:order}$ , to indicate that a node should not have its outgoing relationships ordered. This is not an ideal solution. We have considered adding a flag to the semantics to indicate ordering, but decided to use this simple method for now.

**Definition 4.3** In order to be an *RDF interpretation*, the above six-tuple must also satisfy the following additional conditions:

1.  $O(r)$  is a strict partial order.
2. If  $\langle x, y \rangle \in O(r)$  then  $EXT(x)$  and  $EXT(y)$  have  $r$  as their first element.
3.  $CEXT(S(\text{rdf:Description})) = R$
4.  $CEXT(S(\text{rdf:Property})) \subseteq R$
5.  $S(\text{rdf:type}) \in CEXT(S(\text{rdf:Property}))$
6. If  $\langle x, y \rangle \in EXT$ ,  $y \in CEXT(S(\text{rdf:type}))$ , and  $\langle y, z \rangle \in EXT$  then  $x \in CEXT(z)$ .
7. If  $x \in CEXT(z)$  and  $x \in R$   
then  $\exists y \in R : \langle x, y \rangle \in EXT \wedge y \in CEXT(S(\text{rdf:type})) \wedge \langle y, z \rangle \in EXT$ .
8. If  $d \in DT$  then  $CEXT(S(d)) = DTC(d)$ , provided that  $S$  is defined on  $d$ .

The first and second conditions say that  $O(r)$  is a strict ordering over the relationships emanating from  $r$ . The third, fourth, and fifth conditions provide part of the meaning for some of the built-in RDF vocabulary.

The sixth and seventh conditions relate the two ways of providing typing for resources, one via  $CEXT$  and one via  $\text{rdf:type}$  links. This is needed because  $\text{rdf:type}$  is part of both the theory and the metatheory of RDF. As part of the theory of RDF,  $\text{rdf:type}$  is given a denotation, and relationships impinge on it. As part of the metatheory of RDF, these relationships impose conditions on the extension ( $CEXT$ ) of RDF classes. The sixth condition goes from  $\text{rdf:type}$  links to  $CEXT$  for all resources and data values whereas the seventh condition goes the other way, but not for data values.

The eighth conditions provides meaning for XML Schema primitive datatypes. It ensures that the extension of a resource that corresponds to an XML Schema primitive datatype is the value space of that datatype.

**Example** In Figure 5 we present an RDF graph (the original method for providing meaning to RDF documents) and part of an interpretation. The interpretation is presented in the form of a graph. Most nodes are resources, i.e., elements of  $R$ ; the node with 3000 next to it is the integer 3000 and the node with “www.sony.com” next to it is a string, both data values. (The labels on resource nodes are just so that they can be referred to in the text below.) The mapping  $S$  is given by the dashed arrows from QNames to the nodes. Relationships in the interpretation ( $E$  and  $EXT$ ) are shown as links between the resources. Finally  $CEXT$  contains  $CEXT(l) = \{v\}$ ,  $CEXT(c) = \{s\}$ ,  $CEXT(p) = \{pv\}$ ,  $CEXT(m) = \{mv\}$ ,  $CEXT(h) = \{hs\}$ ,  $CEXT(t) \supset \{tv, ts, tmv, tpv, ths, tt\}$  This is only a partial representation of an interpretation as it does not incorporate  $\text{rdf:Description}$  and  $\text{rdf:Property}$ , relationships to them, and  $CEXT$  for them. Nor does the graph specify the ordering relationships that come from the document order.

As an abbreviation, we will use a triple notation, saying that  $\langle s, p, o \rangle$  is in  $I$  iff there is some resource  $r \in R$  such that  $\langle s, r \rangle \in EXT$ ,  $\langle r, o \rangle \in EXT$ , and  $r \in CEXT(p)$ . For example to indicate that *Vaio505G* is related to *Sony* through a *manufacturer* resource we will use  $\langle Vaio505G, manufacturer, Sony \rangle$ .

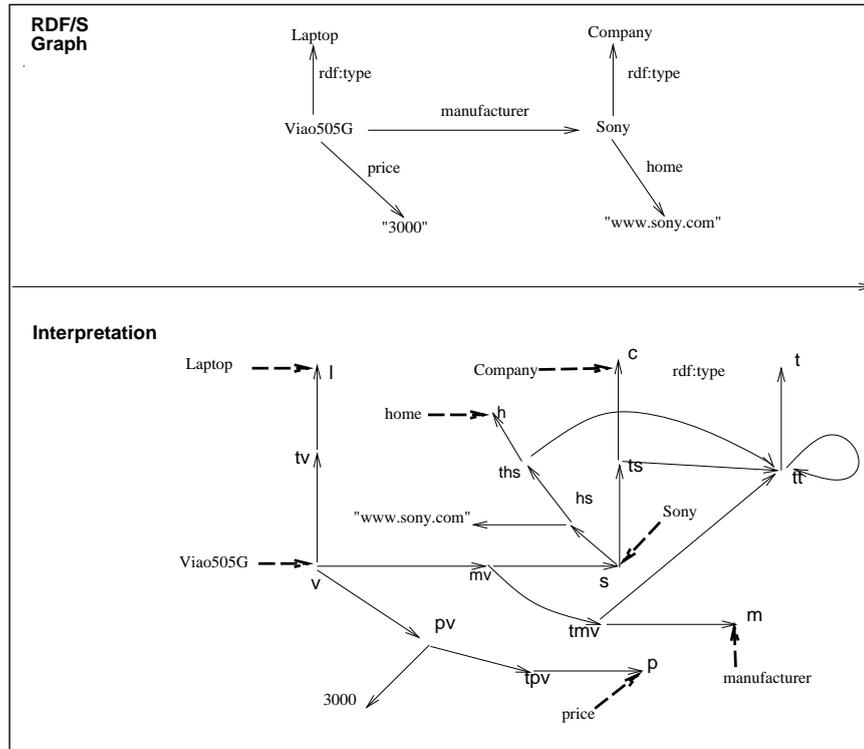


Figure 5: RDF Graph and a corresponding interpretation

**Recovering RDF meaning** As one can see in Figure 5, RDF graphs make a clear distinction between classes and properties, while this distinction is not present in our model theory in order to deal with XML documents. Still, RDF graphs correspond to a precise subset of our interpretations. In a nutshell, RDF graphs corresponds to interpretations in which a proper alternation between classes and properties exists.

This statement is made more precise by the following definition.

**Definition 4.4** Given an interpretation  $I = (R, E, EXT, CEXT, O, S)$  let

- $P = \{x : \exists y x \in CEXT(y) \wedge y \in CEXT(\text{rdf} : \text{Property})\}$
- $E' = E - \{e : \exists y EXT(e) = \langle y, S(\text{rdf} : \text{type}) \rangle\}$   
 $- \{e : \exists x, y, e' EXT(e) = \langle x, y \rangle \wedge EXT(e') = \langle y, S(\text{rdf} : \text{type}) \rangle\}$

If  $P$  makes  $E'$  bipartite, i.e., all  $EXT(e')$  for  $e' \in E'$  either originate or terminate, but not both, in  $P$ , and also for each  $x \in P$  there is exactly one  $e' \in E'$  with  $EXT(e')$  of the form  $(x, y)$  for some  $y$  and exactly one  $e' \in E'$  with  $EXT(e')$  of the form  $(y, x)$  for some  $y$ , then  $I$  is a *bipartite interpretation*.

A bipartite interpretation can be turned into an interpretation in the RDF model theory [16] by taking each pair of relationships  $\langle e', e'' \rangle \in E'$  with  $EXT(e') = (x, p)$  and  $EXT(e'') = (p, z)$  where  $p$  is in  $P$  and replacing it with  $(x, z)$  in  $IEXT(r)$  for each  $r$  such that  $p \in CEXT(r)$  then adding  $(x, c) \in IEXT(S(\text{rdf} : \text{type}))$  for each  $x$  in  $CEXT(c)$  for  $x$  not in  $P$ .

## 4.4 Models of XML and RDF documents

Now that we have defined our notion of interpretation, we need to explain how instances of the XQuery data model correspond to these interpretations, in other words, how  $EI$  corresponds to data set  $ED$  above. Intuitively, each node in the XQuery data model is mapped to a resource in the interpretation, and  $EXT$  relationships are built according to the original tree structure of the XQuery data model instance. On top of that the specific XML Schema and RDF attributes `xsi:type`, `rdf:ID`, `rdf:about`, `rdf:type`, `rdf:resource`, and `rdf:order` are treated specially in order to build a theory of RDF documents that reflect their intended meaning.

**Definition 4.5** An interpretation  $I = \langle R, E, EXT, CEXT, O, S \rangle$  is a *model* for a data set  $N$  if  $S$  is defined on all names in  $N$ , and there are mappings  $M : N \rightarrow R \cup DV$  and  $MA : N' \rightarrow DV$ , where  $N'$  consists of the attribute nodes in  $N$ . Further, the interpretation and mapping have to satisfy the following conditions. Although there are a lot of conditions here, they all really boil down to doing the obvious thing.

1. For each  $n \in N$  an element node,
  - (a)  $M(n) \in R$  and  $M(n) \in CEXT(S(name(n)))$
  - (b) If  $n$  has an attribute with name `rdf:ID` and string-value  $u$  then  $M(n) = S(UTS(n)(u))$ .
  - (c) If  $n$  has an attribute with name `rdf:about` and string-value  $u$  then  $M(n) = S(UTS(n)(u))$ . (This treats `rdf:ID` and `rdf:about` as exactly the same which is not quite correct but a full treatment would have to address the messy differences between QNames and RDF identifiers.)
  - (d) If  $n$  has an attribute with name `rdf:resource` and string-value  $u$  then there is an  $e \in E$  with  $EXT(e) = \langle M(n), S(UTS(n)(u)) \rangle$ .
  - (e) If  $n$  has an attribute with name `rdf:type` and string-value  $u$  then  $\langle M(n), S(UTS(n)(u)) \rangle \in CEXT$ .
  - (f) For each attribute node,  $n'$ , of  $n$ , except for attributes with any of the specific names above, then there is an  $e \in E$  with  $EXT(e) = \langle M(n), M(n') \rangle$ .
  - (g) If  $n$  has a simple type,  $dt$ , then
    - for each of the  $k$  typed-values,  $v_i$ , of  $n$  there is an  $e_i \in E$  with  $EXT(e_i) = \langle M(n), DTS(d)(string - value(v_i)) \rangle$ .
    - if  $n$  has no attribute with name `rdf:order` and value "false" then  $\langle e_i, e_j \rangle \in O$  for  $1 \leq i < j \leq k$ .
  - (h) If  $n$  does not have a simple type, then
    - for each of the  $k$  element or text children nodes,  $n'$ , of  $n$ , in document order there is an  $e_i \in E$  with  $EXT(e_i) = \langle M(n), M(n') \rangle$ .
    - if  $n$  has no attribute with name `rdf:order` and value "false" then  $\langle e_i, e_j \rangle \in O$  for  $1 \leq i < j \leq k$ .
2. For each  $n \in N$  a text node  $M(n) \in DV$  and  $M(n) \in XTS(string - value(n))$
3. For each  $n \in N$  an attribute node, except for attributes with any of the specific names above,
  - (a)  $M(n) \in R$  and  $M(n) \in CEXT(S(name(n)))$
  - (b)  $MA(n) \in DV$  and  $MA(n) \in XTS(string - value(n))$
  - (c) There is some  $e \in E$  with  $EXT(e) = \langle M(n), MA(n) \rangle$ .

- (d) If  $n$  has a simple type,  $dt$ , then  $MA(n) = DTS(dt)(string - value(n))$
4. For each  $n \in N$  a reference node,  $M(n) = M(deference(n))$ .

**Definition 4.6** An *RDF model*  $I$  for  $N$  is an RDF interpretation  $I$  that is a model for  $N$ .

**Example** Now the interpretation in Figure 5 is a model for the document above under the mapping  $M(1) = v$ ,  $M(3) = mv$ ,  $M(5) = pv$ ,  $M(7) = 3000$ ,  $M(8) = s$ ,  $M(11) = ts$ ,  $M(12) = c$ ,  $M(10) = hs$ , and  $MA(10) = "www.sony.com"$ . The other nodes are “structural nodes” and thus do not have a mapping. As XML Schema datatypes only show up in the “structural” nodes, they don’t need to be present.

## 4.5 Entailment

Finally, we are now ready to define a notion of entailment for XML and RDF data sets.

**Definition 4.7** A data set  $N$  *entails* another data set  $N'$  iff every RDF model of  $N$  is also an RDF model of  $N'$ . A collection of data sets entails another data set  $N'$  iff every RDF model of every element of the collection is also an RDF model of  $N'$ .

As we will see in Section 5, entailment is the main reasoning tool that we will use at the application level. Entailment captures valid reasoning, in that if a data set  $N'$  is entailed by another  $N$ , the information in  $N'$  is implicitly present in  $N$ .

## 4.6 Dealing with RDF Schema

RDF Schema (RDFS) is an extension of RDF that has a vocabulary for stating relationships between classes and properties as well as some built-in meaning for this vocabulary.

We handle the RDFS vocabulary by requiring that an RDFS interpretation include the following triples. (Actually there is more RDFS vocabulary than given below, but it is not important.)

```

<S(rdfs:Description), S(rdf:type), S(rdfs:Class)>
<S(rdfs:Description), S(rdfs:subClassOf), S(rdfs:Resource)>
<S(rdfs:Resource), S(rdfs:subClassOf), S(rdf:Description)>
<S(rdfs:Resource), S(rdf:type), S(rdfs:Class)>
<S(rdf:Property), S(rdf:type), S(rdfs:Class)>
<S(rdfs:Class), S(rdf:type), S(rdfs:Class)>
<S(rdfs:Literal), S(rdf:type), S(rdfs:Class)>
<S(rdf:type), S(rdf:type), S(rdf:Property)>
<S(rdfs:subClassOf), S(rdf:type), S(rdf:Property)>
<S(rdfs:subPropertyOf), S(rdf:type), S(rdf:Property)>
<S(rdfs:Class), S(rdfs:subClassOf), S(rdfs:Resource)>
<S(rdf:type), S(rdfs:range), S(rdfs:Class)>
<S(rdfs:subClassOf), S(rdfs:domain), S(rdfs:Class)>
<S(rdfs:subClassOf), S(rdfs:range), S(rdfs:Class)>
<S(rdfs:subPropertyOf), S(rdfs:domain), S(rdf:Property)>
<S(rdfs:subPropertyOf), S(rdfs:range), S(rdf:Property)>
<S(rdfs:range), S(rdfs:domain), S(rdf:Property)>
<S(rdfs:range), S(rdfs:range), S(rdfs:Class)>
<S(rdfs:domain), S(rdfs:domain), S(rdf:Property)>
<S(rdfs:domain), S(rdfs:range), S(rdfs:Class)>

```

RDFS interpretations also must meet the following additional constraints:

1.  $CEXT(S(\text{rdfs} : \text{Resource})) = R$
2.  $CEXT(S(\text{rdfs} : \text{Literal})) = DV$
3. If  $x \in CEXT(y)$  and  $\langle y, S(\text{rdfs} : \text{subClassOf}), z \rangle \in I$  then  $x \in CEXT(z)$ .
4. If  $\langle x, S(\text{rdfs} : \text{subClassOf}), y \rangle \in I$  and  $\langle y, S(\text{rdfs} : \text{subClassOf}), z \rangle \in I$  then  $\langle x, S(\text{rdfs} : \text{subClassOf}), z \rangle \in I$ .
5. If  $\langle x, r, y \rangle \in I$  and  $\langle r, S(\text{rdfs} : \text{subPropertyOf}), s \rangle \in I$  then  $\langle x, s, y \rangle \in I$ .
6. If  $\langle x, S(\text{rdfs} : \text{subPropertyOf}), y \rangle \in I$  and  $\langle y, S(\text{rdfs} : \text{subPropertyOf}), z \rangle \in I$  then  $\langle x, S(\text{rdfs} : \text{subPropertyOf}), z \rangle \in I$ .
7. If  $\langle x, p, y \rangle \in I$  and  $\langle p, S(\text{rdfs} : \text{range}), c \rangle \in I$  then  $y \in CEXT(c)$ .
8. If  $\langle x, p, y \rangle \in I$  and  $\langle p, S(\text{rdfs} : \text{domain}), c \rangle \in I$  then  $x \in CEXT(c)$ .

Now the RDFS analogues of RDF models and RDF entailment are defined in the obvious way.

**Definition 4.8** An RDFS model for a data set  $N$  is an RDFS interpretation that is a model for  $N$ . A data set  $N$  RDFS-entails another data set  $N'$  iff every RDFS model of  $N$  is also a RDFS model of  $N'$ . A collection of data sets RDFS-entails another data set  $N'$  iff every RDFS model of every element of the collection is also a RDFS model of  $N'$ .

## 4.7 Caveats

This is quite a bit of machinery. Some of it is required to handle the two ways, XML and RDF, of looking at the world, and some of it is required to handle the vocabulary of RDF. We feel that this model theory captures the important parts of XML and RDF, but there are some aspects of both XML and RDF that it is missing.

The model theory explicitly discards as irrelevant most of the formatting aspects of the initial document. Similarly it also discards as irrelevant the lexical form of elements that have XML Schema datatypes. There is no way of recovering the exact character sequence of the initial document from the model theory.

Just about the only significant aspect of XML documents that are not handled in the model theory is the distinction between attributes and elements. This is discarded to obtain consistency with RDF.

On the RDF side our scheme does not handle certain RDF constructs. The meaning of reification in RDF and RDF containers have not yet been determined, so the model theory does not address them. One RDF shorthand form is counter to XML practice and thus is not handled in the model theory. As syntax processing is completely handled by XML, our scheme cannot handle the RDF `parsetype` extension.

## 5 Applications of the Yin-Yang model

We can now go back to our catalog example from Figure 2. Remember that the data integration is done in two steps. On the semantic side, the integration designer and builds a global ontology for the information he wants to manipulate in his target application. On the data side, he accesses data from various information providers using XML, resulting in a mismatch between data and semantics, as illustrated on Figure 3.

## 5.1 Semantic Consequences

The first result of our integration is that the semantic consequences of the RDF Schema information are applied to the XML data. For example, the Compaq 4XDF is a portable because that information is entailed from the fact that it is a PDA and the class relationships in the RDF Schema information. This entailed information is available to applications that access the data model view.

We are still exploring the various capabilities that a powerful inference system can support in the XML world. Still, there are two important remarks to be made. First, it is always possible to import more information for reasoning. For instance, it is quite common that various terminologies or vocabularies will be used between the data sources, and in the ontology. Following Fundulaki [1], RDFS, hence our model theory, is quite expressive enough to capture typical thesaurus and incorporate them in the reasoning, on our case on both RDF and XML. For instance, if one would use `manufacturer` instead of `vendor`, the system would still be able to figure our semantics relationships and inconsistencies, assuming these are declared as homonyms in the thesaurus.

## 5.2 Adding querying and restructuring to RDF

A second important application of the Yin/Yang model is related to querying RDF with XQuery. Robie et al [28] propose one approach to query RDF and topic maps documents using XQuery. One of the benefits they gain is the ability to query both XML and RDF information in an uniform framework. Still, a difficulty of their approach relies in lack of understanding of the semantic features of RDF in the XQuery data model. To solve that problem, [28] uses special-purpose functions in XQuery that perform some part of the reasoning.

In the Yin/Yang model, one now has access to both the data model representation and the semantics of the RDF information. Hence, one can use XQuery to access the data structure of the RDF document, while using entailment to access its semantics. For instance, the following function from [28] performs a recursive access on the class hierarchy in order to figure out whether an entity is an instance of a given class.

```
define function rdf:instance-of-class(ListOfDescription $t, charstring $base-name)
  RETURNS ListOfDescription
{
  $t[rdf:type = $base-name]
  ,
  for $i in $t[rdfs:subClassOf = $base-name]
  return rdf:instance-of-class($t, string($i/@rdf:about))
}
```

This function can be defined directly through entailment in the following way:

```
define function rdf:instance-of-class(ListOfDescription $t, charstring $base-name)
  RETURNS ListOfDescription
{
  for $d in $t
  where entails($yin:yang,
    <rdf:Description rdf:about={$d}>
      <rdf:type>
        <rdf:Description rdf:about={$base-name}>
      </rdf:type>
    </rdf:Description>)
  return $x
}
```

This function returns only those descriptions `$d` in `$t` such that the current semantics (represented as a global variable `$yin:yang`) entails the statement (written here in rdf syntax) that `$d` is of class `$base-name`.

Indeed, model theory and entailment provide a precise formal foundation for the techniques presented in [28]. Entailment also allows to perform more complex reasoning, that cannot be captured by a finite set of functions. For instance, one can ask whether all companies with a contract are partners:

```
entails( ($yin:yang,
  <rdf:Description rdf:about={$d}>
    <rdf:type>
      <rdf:Description rdf:about="Company">
        <contract>{$c}</contract>
      </rdf:type>
    </rdf:Description>),
  <rdf:Description rdf:about={$d}>
    <rdf:type>
      <rdf:Description rdf:about="Partner">
        </rdf:type>
      </rdf:Description>)
```

A somewhat interesting remark is that our work would also allow a converse approach to the one of Robie et al, by using an RDF query language such as RQL [17] on the model theory representation of XML documents, hence allowing to query both XML and RDF with and RDF query language. Note again that we are just starting to explore the possibilities provided by such semantics reasoning in RDF as well as in XML. We cannot go into more details due to space limitation, but we believe these simple examples are giving some ideas of the many interesting possibilities open by our approach.

### 5.3 Semantic Integrity

Another interesting question to ask as a user is whether the data built through the integration system is semantically consistent with the target ontology. If there are no models for both the RDF Schema information and the XML data then there is some semantic inconsistency between them.

Unfortunately, RDF Schema is too weak to provide this sort of reasoning. For example, if a mistake in the input causes a resource to be both a Company and a Product, this is not a semantic error, as Company and Product are not necessarily disjoint. This disjointness cannot be enforced in RDF Schema. Adding disjointness reasoning to our approach awaits the integration of more-powerful semantic formalisms, such as the DAML+OIL ontology language [31].

There are also other forms of reasoning we are not able to do due to the expressive power limitations of RDF Schema. We cannot perform any recognition on the information, such as determining that an IBM computer that has a portable property belongs to the same category as a SONY laptop. Again such inference is available in newer proposals for the semantic web, such as DAML+OIL.

## 6 Implementation

The approach we have outlined has several implementation advantages. First and foremost, as the initial stages of processing for both XML and RDF documents are the same, there is no need to have a syntax processor for RDF. Second, the approach integrates XML Schema datatypes, eliminating the need for a separate datatype implementation for RDF.

As a result, an implementation of the entire Yin/Yang model can be quickly and easily written, by exploiting tools for the XQuery data model. In fact, we have written an initial reference implementation of the scheme in OCaml [19]. This implementation builds on the Galax XQuery engine [15] and data model implementation, and our implementation on top of it is only about 300 lines of code.

Although our implementation is not complete, it provides the core of what is needed for a complete system based on our scheme: the mapping from the XQuery data model and the entailment algorithm. This implementation is available at <http://www-db.research.bell-labs.com/user/pfps/xmlrdf-semantics.ml><sup>3</sup>.

The implementation uses the fact that a canonical model exists for our scheme. This is a traditional implementation technique for semantic reasoners. Given a collection of XML and RDF documents, the implementation builds a canonical model for them. This canonical model can then be used to determine whether another document is entailed by the original documents.

## 7 Future Work and Conclusion

Our work aims at providing both the data and semantics access required by many of the new Web applications. We have presented an original framework and model for a Web that integrates both syntax and semantics: the Yin/Yang Web. This unified Web is obtained through a tighter integration between XML and RDF, and we believe such integration is essential for the coming Semantic Web envisioned by many.

We see this work as foundational, and at this point it is raising many new interesting, but probably difficult, questions. The obvious next step is to understand precisely how much of XML Schema can and should be incorporated into the semantic framework. We only touched the problem by addressing two aspects of XML Schema that are easy to capture in our semantic framework: ID/IDREF and type names hierarchy. Still, XML Schema includes a very large number of constructs that are more difficult to handle in a semantic framework such as regular expressions and arity constraints.

Also on the querying part, we believe both XML and RDF users could benefit from the proposed framework and we intend to explore that in more depth. Notably, it is not clear to us whether it would be more suitable to query the Yin/Yang Web by extending an XML Query language such as XQuery, by extending an RDF query language, such as RQL, or by designing a unified language that would provide some features of both.

Another interesting direction is to extend this scheme to more-expressive formalisms, such as DAML+OIL. The problem here is that DAML+OIL includes constructs that violate one of the fundamental assumptions of XML and RDF. All constructs and XML and RDF are conjunctive, in that the meaning of an entire collection of documents is the conjunction of the meaning of each of the pieces. Handling non-conjunctive constructs in an upwardly-compatible fashion requires a different treatment of DAML+OIL than in its semantics. (The basic change would be to treat DAML+OIL syntax in a more XML-like fashion, i. e., as a tree, rather than its current RDF fashion.)

Other extensions to our scheme are also possible, potentially leading to a unified semantics for the entire semantic web layer cake. As our model theory is somewhat different from the standard logical model theories, some work will be required create a compatible model theory for logics like first-order logic.

**Acknowledgments.** We thank Vassilis Christophides and Jonathan Robie for provocative discussions about semantics and about the Web, and for feedback on earlier drafts. We thank the Galax team for developing a system we could use in our implementation. Finally, special thanks to Arnaud Sahuguet for the idea of the Yin and the Yang.

---

<sup>3</sup>This is a temporary placement for review purposes only.

## References

- [1] B. Amann, I. Fundulaki, and M. Scholl. Integrating ontologies and thesauri for rdf schema creation and metadata querying. *International Journal of Digital Libraries*, 2000.
- [2] C. K. Baru, V. Chu, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, and P. Velikhov. XML-based information mediation for digital libraries. In *ACM conference on Digital Libraries*, pages 214–215, Berkeley, CA, Aug. 1999.
- [3] R. J. Bayardo Jr., W. Bohrer, R. S. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. H. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Semantic integration of information in open and dynamic environments. In *Proceedings of ACM Conference on Management of Data (SIGMOD)*, pages 195–206, Tucson, Arizona, May 1997.
- [4] T. Berners-Lee. Why RDF model is different from the XML model, 1999.  
<http://www.w3.org/DesignIssues/RDF-XML.html>.
- [5] H. Boley. A web data model unifying XML and RDF.  
<http://www.dfki.uni-kl.de/~boley/xmlrdf.html>, Sept. 2001.
- [6] T. Bray, J. Paoli, C. Sperberg-McQueen, and E. Maler. Extensible markup language (XML) 1.0 (second edition). W3C recommendation., Oct. 2000.  
<http://www.w3.org/TR/1998/REC-xml>.
- [7] D. Carlson. *Modeling XML Applications with UML. Practical e-Business Applications*. Addison-Wesley, 2001.
- [8] D. Chamberlin, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. Xquery 1.0: An xml query language, June 2001.  
<http://www.w3.org/TR/xquery/>.
- [9] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! In *Proceedings of ACM Conference on Management of Data (SIGMOD)*, pages 177–188, Seattle, Washington, June 1998.
- [10] J. Cowan and R. Tobin. Xml information set. W3C Recommendation, Oct. 2001.  
<http://www.w3.org/TR/xml-infoset/>.
- [11] Enosys markets.  
<http://www.enosysmarkets.com>.
- [12] M. Fernández and J. Marsh. The XQuery 1.0 and XPath 2.0 data model. W3C Working Draft., June 2001.  
<http://www.w3.org/TR/query-datamodel/>.
- [13] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. System demonstration - Strudel: A web-site management system. In *Proceedings of ACM Conference on Management of Data (SIGMOD)*, Tucson, Arizona, May 1997. Exhibition Program.
- [14] I. Fundulaki, B. Amann, M. Scholl, C. Beerli, and A.-M. Vercoustre. Mapping xml fragments to community web ontologies. In *Fourth International Workshop on the Web and Databases (WebDB'2001)*, Santa Barbara, CA, June 2001.

- [15] Galax xquery 1.0 implementation.  
<http://db.bell-labs.com/galax/>.
- [16] P. Hayes. RDF model theory. W3C Working Draft, Sept. 2001.  
<http://www.w3.org/TR/rdf-mt/>.
- [17] G. Karvounarakis, V. Christophides, D. Plexousakis, and S. Alexaki. Querying rdf descriptions for community web portals. In *The French National Database Conference, BDA'2001*, Agadir, Maroc, Oct. 2001.  
<http://139.91.183.30:9090/RDF/RQL/>.
- [18] O. Lassila and R. Swick. Resource description framework (RDF) model and syntax specification. W3C Recommendation, Feb. 1999.  
<http://www.w3.org/TR/REC-rdf-syntax/>.
- [19] X. Leroy. *The Objective Caml system, release 3.02, Documentation and user's manual*. Institut National de Recherche en Informatique et en Automatique, Nov. 1999.  
<http://caml.inria.fr/>.
- [20] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of International Conference on Very Large Databases (VLDB)*, pages 251–262, Bombay, India, Sept. 1996.
- [21] M. Mani, D. Lee, and R. R. Muntz. Semantic data modeling using xml schemas. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER 2001)*, Yokohama, Japan, Nov. 2001.
- [22] I. Manolescu, D. Florescu, D. Kossmann, F. Xhumari, and D. Olteanu. Agora: Living with xml and relational. In *Proceedings of International Conference on Very Large Databases (VLDB)*, pages 623–626, 2000.
- [23] S. Melnik. Bridging the gap between RDF and XML, Dec. 1999.  
<http://www-db.stanford.edu/~melnik/rdf/fusion.html>.
- [24] L. Miller. Inkling: Rdf query using squishql.  
<http://swordfish.rdfweb.org/rdfquery/>.
- [25] Nimble technology.  
<http://www.nimble.com>.
- [26] Organization for the advancement of structured information standards.  
<http://www.oasis-open.org/>.
- [27] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, pages 251–260, Taipei, Taiwan, Mar. 1995.
- [28] J. Robie, L. M. Garshol, S. Newcomb, M. Biezinski, M. Fuchs, L. Miller, D. Brickley, V. Christophides, and G. Karvounarakis. The syntactic Web: Syntax and semantics on the Web. In *Extreme Markup Languages'2001*, Montreal, Canada, Aug. 2001.
- [29] H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML schema part 1: Structures. W3C Recommendation, May 2001.  
<http://www.w3.org/TR/xmlschema-1/>.

- [30] F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks. A model-theoretic semantics for DAML+OIL, Mar. 2001.  
<http://www.daml.org/2001/03/model-theoretic-semantics.html>.
- [31] F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks. Reference description of the DAML+OIL (march 2001) ontology markup language, Mar. 2001.  
<http://www.daml.org/2001/03/reference.html>.