Exploring Semantic Web Modeling Approaches for Web Application Design

Fernanda Lima Daniel Schwabe

ferlima@inf.puc-rio.br schwabe@inf.puc-rio.br

Abstract

In this paper we argue the need for flexible models capable of representing semi-structured data processed in Web applications. We also give the requirements for query languages that allow the specification of navigational aspects of Web applications according to the OOHDM approach, and show how these can be met by RQL. We give additional uses of this query language, as applied to such models, in terms of its applicability to other aspects, such as application design frameworks and adaptable web applications.

1 Introduction

Web design methods such as HDM [Garzotto+93], RMM [Isakowitz+95], OOHDM [Rossi+99, Schwabe+98], WebML [Ceri+00], OO-H Method[Gomez+00], UWE [Henniker+00] capture web application essences through software engineering models, preferably ones that are well known by the web engineering community. For example, OOHDM utilizes the well known UML Class Diagram [OMG99] to express the Conceptual Model related to a specific domain. This means that the domain is represented using object-oriented concepts [Rumbaugh+91]. Other methods also use OO concepts or even Entity-Relationship concepts for this same purpose.

Nevertheless, web application domains might not be accurately represented using only these paradigms, since data found on the web cannot be considered structured, as these notations presume. Nowadays researchers refer to web data as semi-structured data [Abiteboul+00].

Recently, XML has been largely used to represent web data, since it can be used to describe document structure in an organized manner. However, XML alone does not take us very far; in order to express anything with some semantic meaning, it is necessary to add other layers on top of XML.

According to [Berners-Lee00], the architecture of the Semantic Web is composed of several layers, where both XML and RDF play major roles. The idea is to reach "a Web of data that can be processed directly or indirectly by machines".

In this work, we make use of the following layers: XML-Schema [W3C01a], RDF [W3C98], RDF-Schema [W3C00b], Ontology and Logic to model distributed web applications, query their data and infer knowledge.

The purpose of this paper is to propose models for hypermedia (and Web) applications based on semantic web data models, using the OOHDM approach. More specifically, we present ways to map each OOHDM model to XML-like structures and show the main benefit of seamlessly querying both metadata and data on the web.

This paper is organized as follows. In Section 2, we briefly review the main concepts of OOHDM method and, through an example, show the mapping of the Conceptual Model to an RDF representation. In Section 3, we present the Navigational Model, which entails requirements for an RDF query language that can query both schema and instance, and show some query examples. In Section 4 we describe some uses of the present work. In Section 5 we conclude and make brief comments about our future work.

2 Conceptual Model

In OOHDM, the Conceptual Model shows classes and their relationships specifically related to a domain. Classes are described as in object-oriented UML models, with two distinguished details on attributes: they can be multi-typed, representing different perspectives of the same real-world entity, and they are described with multiplicity, implying the number of times the attribute will occur in instances.

Figure 1 shows our motivating example that will be detailed throughout this paper. This CD Store Conceptual Model describes the attributes of a CD and its relationships with both the Order that a Client can make and the Songs that Artists can compose with the Versions they perform.

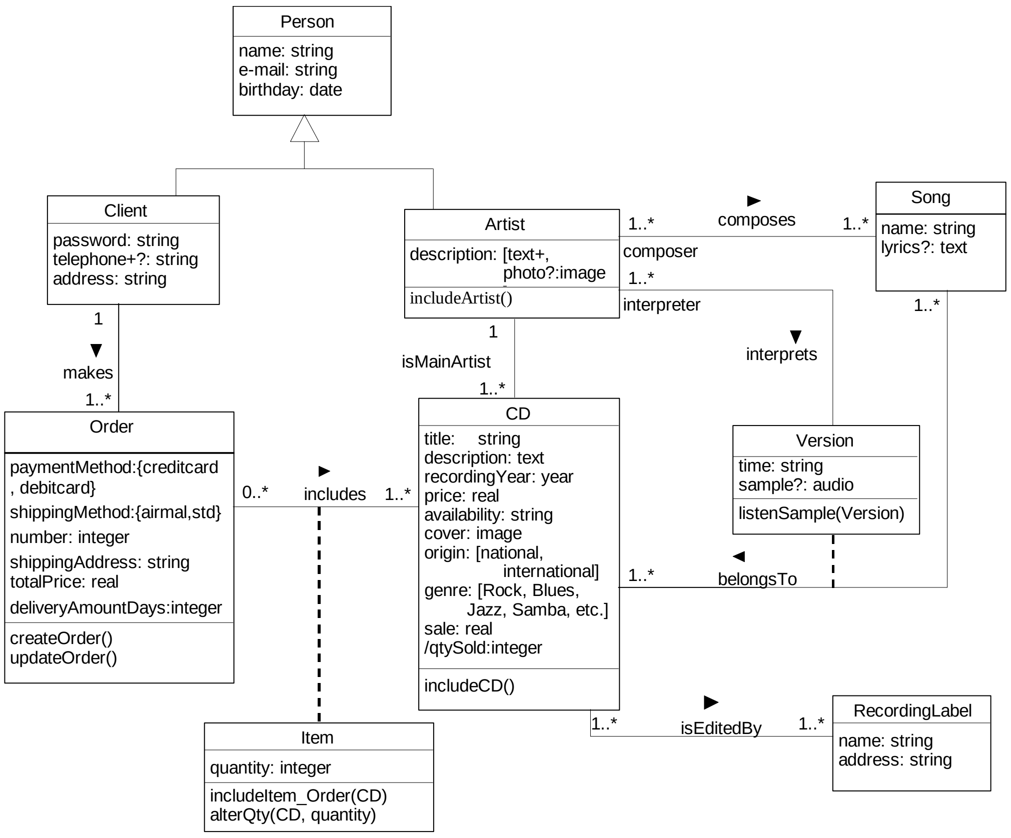

Figure 1 - CD Store Conceptual Model

To facilitate describing semi-structure data, we propose to represent the Web Application Conceptual Models using a notation from the UML Specification Language seldom used, attribute multiplicity.

As stated in [OMG99], attribute multiplicity is the possible number of data values for the attribute that may be held by an instance. The cardinality of the set of values is an implicit part of the attribute. In the common case in which the multiplicity is 1..1, then the attribute is a scalar (i.e., it holds exactly one value).

Attribute multiplicity is particularly useful to present semi-structured data as semi-structured classes, since it makes possible the representation of attributes that occur in one instance and do not exist in another. This facilitates the mapping between a class diagram and an XML (eXtensible Markup Language)[W3C00a] representation of the model. Web-based and non-Web based applications can be described by a number of schema specification mechanisms and most of the static information represented in UML class model can be represented in RDF-Schema [W3C98].

According to [W3C02], the Resource Description Framework (RDF) is a general-purpose language for representing information in the World Wide Web. It is particularly intended for representing metadata about Web resources. However, by generalizing the concept of "Web resource", RDF can be used to represent information about anything that can be identified on the Web, such as information about products available in online shopping-centers.

Based on the previous OOHDM Conceptual Model, we can obtain an XML description that allows us to express information about Web resources.

We chose to use not only RDF-Schema, but also DAML+OIL [Harmelen+01] ontology language to express more advanced features such as constraints (restrictions), enumeration and datatypes according to XML Schema [W3C01b]. DAML+OIL is a semantic markup language for Web resources. It builds on the W3C standards RDF and RDF-Schema, and extends these languages with richer modelling primitives, commonly found in frame-based languages. We use the most recent DAML+OIL [Harmelen+01] specification, which utilizes XML Schema datatypes.

While UML diagrams are good for communicating between software engineers, RDF is about making machine-processable statements. RDF statements are divided into three parts: subject, predicate and object. The subject identifies what the statement is about, the predicate identifies the property or characteristic of the subject, and the object is the value of that property. More details about RDF are beyond the scope of this paper and can be found in W3C site http://www.w3.org/RDF/. The reader interested in a comparison between XML-Schema, RDF-Schema and DAML+OIL should refer to [Gil+01].

The CD Store example shown in Figure 1 can be mapped to a RDF-Schema using the concepts of class and subclass to build hierarchies and rdf:properties to represent UML attributes and relationships.

In Figure 2 we show some parts of the our CD Store Conceptual Model in RDF-Schema using the RDF/XML serialization format. Notice the use of DAML+OIL to specify restrictions and classes.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--namespaces declarations-->
    <!--Class Person-->
    <daml:Class ID="Person">
        <rdfs:label>Person</rdfs:label>
    </daml:Class>
    <!--Person Properties-->
    <daml:DatatypeProperty rdf:ID="Person.name">
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#String"/>
        <daml:domain rdf:resource="#Person"/>
    </daml:DatatypeProperty>
    <daml:DatatypeProperty rdf:ID="Person.birthday">
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#date"/><!--datatype date-->
        <daml:domain rdf:resource="#Person"/>
    </daml:DatatypeProperty>

    <!--Class Client *** Simple Inheritance ***-->
    <daml:Class rdf:ID="Client">
```

```
        <rdfs:label>Client</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Person"/>
    </daml:Class>
    <!--Client Properties-->
    <daml:DatatypeProperty rdf:ID="Client.telephone">
        <daml:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
        <daml:domain rdf:resource="#Client"/>
    </daml:DatatypeProperty>
    <daml:Restriction>
        <daml:onProperty rdf:resource="# Client.telephone"/>
        <daml:minCardinality>0</daml:minCardinality><!--cardinality expressing optional attribute-->
    </daml:Restriction>

    <!--Class Artist *** Simple Inheritance ***-->
    <daml:Class rdf:ID="Artist">
        <rdfs:label>Artist</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Person"/>
    </daml:Class>

    <!--Class Order-->
    <daml:Class rdf:ID="Order">
        <rdfs:label>Order</rdfs:label>
    </daml:Class>
    <!--Order Properties-->
    <rdf:ObjectProperty rdf:ID="Order.shippingMethod">
        <daml:range rdf:resource="#ShippingMethod"/>
        <daml:domain rdf:resource="#Order"/>
    </rdf:ObjectProperty>
    <!--Order Enumeration Properties-->
    <daml:Class rdf:about="ShippingMethod">
        <daml:oneOf parseType="daml:collection">
                <cd:DeliveryMethod rdf:about="StandardGround"/>
                <cd:DeliveryMethod rdf:about="2ndDayAir"/>
                <cd:DeliveryMethod rdf:about="Express"/>
        </daml:oneOf>
    </daml:Class>

    <!--Class CD-->
    <daml:Class rdf:ID="CD">
        <rdfs:label>CD</rdfs:label>
    </daml:Class>
    <!--CD Properties-->
    <daml:DatatypeProperty rdf:ID="CD.title">
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#String"/>
        <daml:domain rdf:resource="#CD"/>
    </daml:DatatypeProperty>
    <daml:DatatypeProperty rdf:ID="CD.genre">
        <daml:range rdf:resource="#Genre"/>
        <daml:domain rdf:resource="#CD"/>
    </daml:DatatypeProperty>
    <daml:DatatypeProperty rdf:ID="CD.sale">
        <daml:range rdf:resource="Number"/>
        <daml:domain rdf:resource="#CD"/>
    </daml:DatatypeProperty>
    <!--CD Enumeration Properties-->
    <daml:Class rdf:ID="Genre ">
        <daml:oneOf parseType="daml:collection">
                <cd:Genre rdf:ID="Rock"/>
                <cd:Genre rdf:ID="Blues"/>
                <cd:Genre rdf:ID="Samba"/>
                <cd:Genre rdf:ID="Jazz"/>
                <cd:Genre rdf:ID="Instrumental"/>
        </daml:oneOf>
    </daml:Class>

    <!--Relationships-->
    <daml:ObjectProperty rdf:ID="makes">
```

```
          <daml:domain rdf:resource="#Client"/>
          <daml:range rdf:resource="#Order"/>
          <daml:minCardinality>1</daml:minCardinality>
     </daml:ObjectProperty>
     <daml:ObjectProperty rdf:ID="composes">
          <daml:domain rdf:resource="#Artist"/>
          <daml:range rdf:resource="#Song"/>
     </daml:ObjectProperty>
     <daml:ObjectProperty rdf:ID="interprets">
          <daml:domain rdf:resource="#Artist"/>
          <daml:range rdf:resource="#Version"/>
     </daml:ObjectProperty>
     <daml:ObjectProperty rdf:ID="isMainArtist">
          <daml:domain rdf:resource="#Artist"/>
          <daml:range rdf:resource="#CD"/>
     </daml:ObjectProperty>
</rdf:RDF>
```

Figure 2 - CD Store RDF-Schema in RDF/XML serialization format

Parts of one instance of the CD Store represented in RDF is shown in Figure 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--namespace declarations--  ...   -->
     <cd:CD rdf:ID="cd001">
          <rdfs:label>cd001</rdfs:label>
          <cd:CD.name>Heitor Villa-Lobos-Obra Integral para Violao Solo</cd:CD.name>
          <cd:CD.availability><xsd:nonNegativeInteger rdf:value="3"/></cd:CD.availability>
          <cd:CD.genre>Instrumental</cd:CD.genre>-
          <cd:CD.sale>10.0</cd:CD.sale>
     </cd:CD>
     <cd:Artist  rdf:resource="art01">
          <rdfs:label>art01</rdfs:label>
          <cd:Person.name>Heitor Villa-Lobos</cd:Person.name>
     </cd:Artist>
     <cd:Artist  rdf:resource="art02">
          <rdfs:label>art02</rdfs:label>
          <cd:Person.name>Paulo Pedrassoli</cd:Person.name>
          <cd:Person.eMail>paulop@mus.cepuerj.br</cd:Person.eMail>
          <cd:Person.birthday><xsd:date rdf:value="1919-10-26"/>"</cd:Person.birthday>
     </cd:Artist>
     <cd:Song rdf:resource="son03">
          <cd:Song.name>Preludio n.1</cd:Song.name>
     </cd:Song>
     <cd:Version rdf:resource="ver03">
          <cd:Version.time>4:21</cd:Version.time>
     </cd:Version>
     <cd:RecordingLabel rdf:ID="rec05">
          <cd:RecordingLabel.name>CEPUERJ</cd:RecordingLabel.name>
     </cd:RecordingLabel>
     <cd:Client rdf:ID="cli01" xmlns="urn:myrdf-person-schema">
          <cd:Person.name>Joao da Silva></cd:Person.name>
...
     </cd:Client>
     <cd:Order rdf:ID="ord01">
          <rdfs:label>ord01</rdfs:label>
          <cd:Order.paymentMethod>creditcard</cd:Order.paymentMethod>
          <cd:Order.shippingMethod>2ndDayAir</cd:Order.shippingMethod>
     </cd:Order>

     <!--Relationships-->
     <cd:makes>
          <cd:Client rdf:resource="#cli01"/>
          <cd:Order rdf:resource="#ord01"/>
     </cd:makes>
     <cd:includes>
          <cd:Order rdf:resource="ord01"/>
          <cd:CD rdf:resource="cd01"/>
```

```
    </cd:includes>
    <cd:composes>
        <cd:Artist rdf:resource="#art01"/>
        <cd:Song  rdf:resource="#son03"/>
    </cd:composes>
    <cd:interprets>
        <cd:Artist rdf:resource="#art02"/>
        <cd:Version rdf:resource="#ver03"/>
    </cd:interprets>
    <cd:isEditedBy>
        <cd:CD rdf:resource="#cd001"/>
        <cd:RecordingLabel rdf:resource="#rec05"/>
    </cd:isEditedBy>
    <cd:has>
        <cd:CD rdf:resource="#cd001"/>
        <cd:Song rdf:resource="#son003"/>
    </cd:has>
</rdf:RDF>
```

Figure 3 - CD Store instance in RDF/XML serialization format

Figures 4 and 5 show parts of other two possible forms of representing the same CD Store instance, through a direct labeled graph (labeled both on edges and nodes) and N-triples.

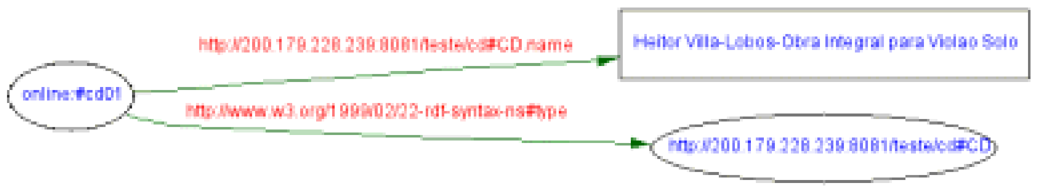
online:#cd01
http://200.179.228.239:8081/teste/cd#CD.name
Heitor Villa-Lobos-Obra Integral para Violao Solo
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://200.179.228.239:8081/teste/cd#CD

Figure 4 - Part of a CD Store instance in RDF direct labeled graph format

Subject	Predicate	Object
cd01	rdf:type	'http://200.179.228.239:8081/teste/cd#CD'
cd01	http://www.w3.org/2000/01/rdf-schema#:label	literal(cd01)
cd01	http://200.179.228.239:8081/teste/cd#:CD.name	literal(Heitor Villa-Lobos-Obra Integral para Violao Solo)
art01	rdf:type	'http://200.179.228.239:8081/teste/cd#Artist'
art01	http://www.w3.org/2000/01/rdf-schema#:label	literal(art01)
art01	http://200.179.228.239:8081/teste/cd#:Person.name	literal(Heitor Villa-Lobos)

Figure 5 - Part of a CD Store instance in N-triples format

3 Navigational Model

As stated in [Rossi+99], web applications, as hypermedia applications, should provide easy navigational access to their related resources, preventing users from getting lost and providing consistent navigation operation. In order to achieve that, the OOHDM approach defines the following navigation primitives [Schwabe+98]:

- navigation objects: views of conceptual objects;
- navigation contexts: sets of navigation objects according to rules determined by the application designer.

Navigation contexts may be further specified as groups of contexts, since it is possible to sometimes parameterize their defining property. For example, “CD by Genre” is actually a set of sets; each set is a context, determined by one value of the "genre" attribute. A similar definition may be obtained for contexts whose property is based on 1-to-n relations, such as "CD by Artist".

The OOHDM Navigation Model consists of two schemas: Navigational Class schema and Navigational Context schema. The former defines all navigable objects as views over the application domain, using a set of pre-defined classes: nodes, links and access structures. The latter defines the main structuring primitive for the navigational space: navigational contexts and links that connect them. The contexts can represent objects related to each other by some aspect (e.g., common attributes or being related to a common object) and organize these objects as sets of nodes, defining in which way they are accessed (e.g., sequentially).

These primitives allow designers using OOHDM to define the web application navigation, deciding which attributes will be shown, which objects will be navigated, their relationship with the conceptual objects, which contexts are useful for navigation, how the user can go from one context to the other, how he can navigate inside a context, and how user profiles may change the way objects are shown within contexts.

Figures 6 and 7 show the CD Store Navigational Class schema and Navigational Context schema, respectively.

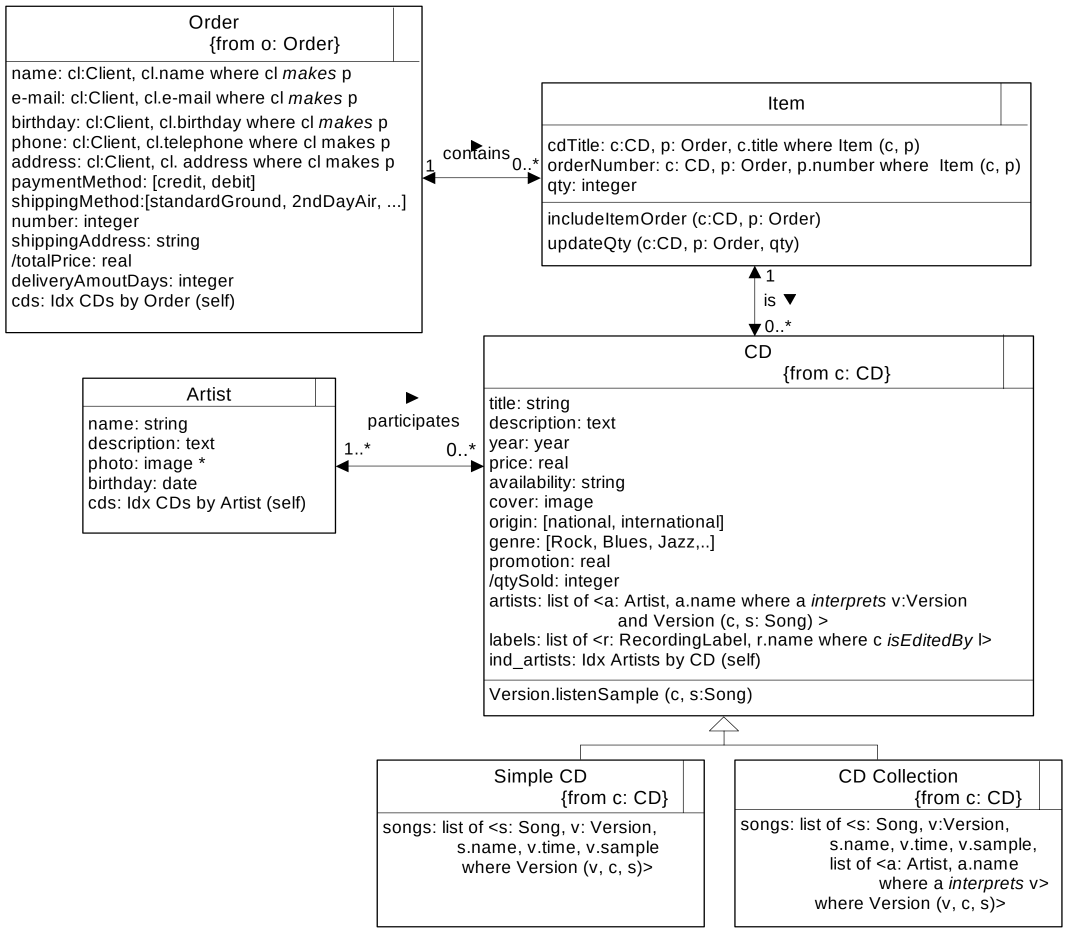
Order
{from o: Order}
name: cl:Client, cl.name where cl *makes* p
e-mail: cl:Client, cl.e-mail where cl *makes* p
birthday: cl:Client, cl.birthday where cl *makes* p
phone: cl:Client, cl.telephone where cl makes p
address: cl:Client, cl. address where cl makes p
paymentMethod: [credit, debit]
shippingMethod:[standardGround, 2ndDayAir, ...]
number: integer
shippingAddress: string
/totalPrice: real
deliveryAmoutDays: integer
cds: Idx CDs by Order (self)
contains
1
0..*
Item
cdTitle: c:CD, p: Order, c.title where Item (c, p)
orderNumber: c: CD, p: Order, p.number where Item (c, p)
qty: integer
includeItemOrder (c:CD, p: Order)
updateQty (c:CD, p: Order, qty)
1
is
0..*
CD
{from c: CD}
title: string
description: text
year: year
price: real
availability: string
cover: image
origin: [national, international]
genre: [Rock, Blues, Jazz,..]
promotion: real
/qtySold: integer
artists: list of <a: Artist, a.name where a *interprets* v:Version and Version (c, s: Song) >
labels: list of <r: RecordingLabel, r.name where c *isEditedBy* l>
ind_artists: Idx Artists by CD (self)
Version.listenSample (c, s:Song)
Artist
name: string
description: text
photo: image *
birthday: date
cds: Idx CDs by Artist (self)
participates
1..*
0..*
Simple CD
{from c: CD}
songs: list of <s: Song, v: Version, s.name, v.time, v.sample where Version (v, c, s)>
CD Collection
{from c: CD}
songs: list of <s: Song, v:Version, s.name, v.time, v.sample, list of <a: Artist, a.name where a *interprets* v> where Version (v, c, s)>

Figure 6 - CD Store Navigational schema

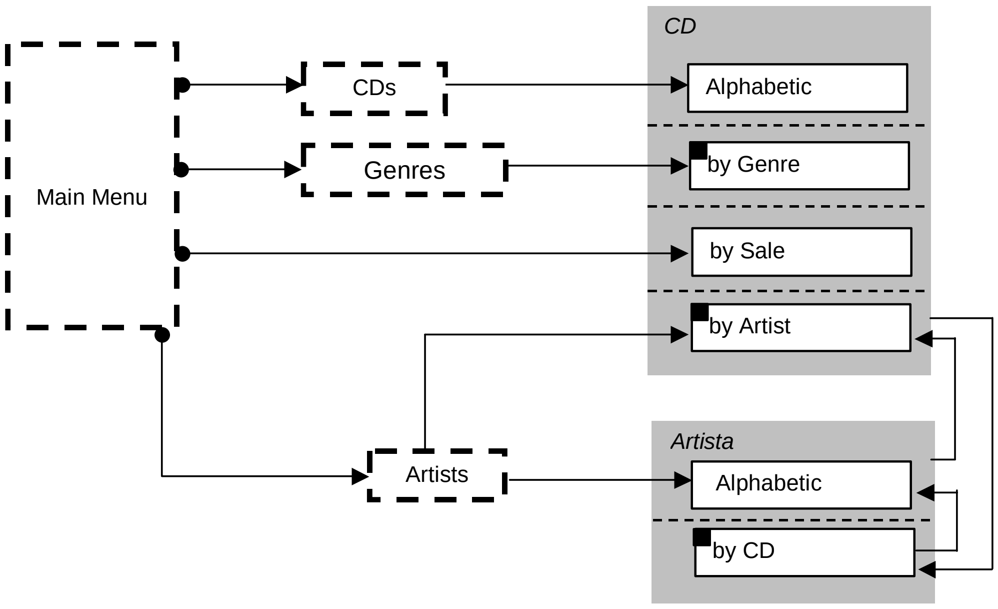

Figure 7 - CD Store Navigational Context schema

The definition of nodes, contexts, and access structures all depend on query specifications, of different expressive power. The next section examines the requirements for such languages.

3.1 Requisites for a query language

In order to represent the navigational model, we need the assistance of a query language capable of dealing with XML. However, simply querying the XML files at a syntactic level or at a structure level would not extract the kind of semantic information we need. Querying at the syntactic level would force us to traverse a node-labeled tree (while RDF is a direct labeled graph) . Querying at structure level would let us explore our Class definitions but would not help us explore subclasses relationships implied in the semantics.

Therefore, we define the following pre-requisites for our query language:

- capability of extracting sets, to represent the contexts;
- capability of querying both schema and instance, to represent contexts and groups of contexts, among other uses;
- inference capability, to extract information that was not previously modeled;
- declarative mode, to allow the description of the query in OOHDM specification cards;
- XML Schema datatype support.

In this work, we chose to use the RQL query language [Karvounarakis+02], a typed language following a functional approach (a la ODMG-OQL[Cattell+00]). It supports generalized path expressions featuring variables on both labels for nodes (i.e., classes) and edges (i.e., properties). RQL relies on a formal graph model (as opposed to other triple-based RDF Query Languages) that captures the RDF modeling primitives and permits the interpretation of superimposed resource descriptions by means of one or more schemas.

The novelty of RQL lies in its ability to smoothly combine schema and data querying. RQL supports: XML Schema data types (for filtering literal values), grouping primitives (for constructing nested XML results), arithmetic operations (for converting literal values), aggregate functions (for extracting statistics), namespace facilities (for handling different schemas), meta-schemas querying (for browsing schemas) and recursive traversal of class and property hierarchies.

RQL is defined by means of a set of core queries, a set of basic filters, and a way to build new queries through functional composition and iterators. Other features that will be useful for our examples are the aggregate functions, boolean predicates and set operators. Figure 8 shows some examples of RQL queries:

Query	Description
http://www.icom.com/schema.rdf#Person	retrieves all instances of a class "Person"
subclassOf(http://www.icom.com/schema.rdf#Person)	retrieves all known subclasses of the class "Person"
select x, $x, y, $y from {x: $x} #creates {y:$y}	retrieves all resources which have a property 'creates', the target of that property and the classes to which the source and target value belong

Figure 8 - RQL Query examples

It should be stated that, up to the time of writing of this paper, RQL was the only language capable of querying both schema and data definitions, but it is not yet prepared for dealing with full DAML+OIL.

3.2 Navigational Model query examples

Continuing with our CD Store example, we can create RQL queries that will provide information about our resources, as shown in Figure 9.

Context	RQL Query	Description
CD (alpha)	http://ww.ferlima.com.br/cd#CD	retrieves all instances of a class CD as a seq (ordered by label)

CD by sale	select x, y
	from http://www.ferlima.com.br/cd#CD {x}.
	http://www.ferlima.com.br/cd#sale{y}
	where y != 0
		retrieves all instances of class CD that have property sale different than zero

CD by Artist
parameterA
(a:Artist)
	select x
	from http://www.ferlima.com.br/cd#Artist {y}.
	http://www.ferlima.com.br/cd#participates {x}
	where y = "parameterA"
		retrieves all instances of class CD in which a certain Artist participates

CD by style parameterB (b:Genre)	select x, y from {x} http://www.ferlima.com.br/cd#genre{y} where y ="parameterB"	retrieves all instances of class CD with a certain genre (among enumerated values)

Figure 9 - CD Store RQL Queries

These declarative queries can be written in OOHDM specification cards by the navigation developer, and also used during the implementation phase. This way it would be easy to access an updated version of the documentation, since any change in the implementation query could be easily reflected in the specification cards. This seems trivial, but what usually happens is that the models are utilized before the implementation phase and "thrown away" after implementation due to a mismatch between the paradigms.

In previous versions of OOHDM, during the implementation phase, an OODHM specialist could choose to implement the conceptual model classes and relationships in a relational database and create views to represent the navigational model. The contexts could be obtained by querying those views using SQL. If by any reason there was a need to alter a specific context, it was necessary to look at SQL queries in the implementation and find out which one was the related context.

With the present work, we propose to represent both the conceptual and navigational models using RDF-schema with DAML+OIL. This is possible due to the characteristic of reification, which allows us to first define the classes and relationships of the conceptual model as statements, and later define statements about these statements to express the navigational model.

4 Possible scenarios

The queries mentioned so far are related to instances of the application. However there are situations where it can be useful to query also the application schema.

4.1 Mediator Architecture

A first example is an application querying a mediator [Wiederhold92] over two online stores, like Amazon and Barnes&Noble, as found in shopping bots that find best deals on products sold in several stores. Suppose we want to retrieve all CDs that are on sale in both stores. It would be necessary to build a mediator and make the query as it appears in Figure 9, called "CD by sale". This query would access a canonical model instead of querying the two store models. Nevertheless, either in Amazon.com or in Barnes&Noble, the attribute that we modeled as "sale" might be called something else such as “promotion” or “discount”. We are assuming that both store schemas and data would be documented in RDF Schemas and RDF instances.

We can use RQL queries to obtain the precise term used in each store, querying their schema and discovering which properties are related to CDs. After finding out the property name we can query the instances and verify which CDs have a value different than zero for that property.

A RQL schema query to find all classes under which a given resource (ex.: a CD picture cover) is classified:

```
select  $C
from $C {x}
where x = &http://www.amazon.com/imagesCDs/YellowSubmarine.jpg
```

We could also use a RQL core query to obtain all classes under the Amazon schema just by using a special metaclass:

```
Class
```

To obtain all properties of a CD Class we could use:

```
select  @P, range (@P)
from {;CD} @P
```

When the corresponding attributes are found in both schemas, our canonical model can be updated with a DAML_OIL property called "samePropertyAs":

```
<rdf:Description about="#sale">
    <daml:samePropertyAs rdf:resource="http://www.amazon.com/cd_schema/vocab/discount"/>
</rdf:Description>
```

At the wrapper level it would be necessary to map the attribute name "sale" to a name from the specific store, and make the corresponding query:

```
select  x, y
from http://www.amazon.com/cd#CD {x}.
http://www.amazon.com.br/cd#discount{y}
where y != 0
```

This query would return a "bag" with all the CDs on sale at Amazon CD on-line store.

4.2 Specification of Application Frameworks

Another possible example is in the definition of Web Application Frameworks [Schwabe+01a, b], in which "parameterized" context definitions are given. Consider, for instance, a catalog for Electronic Commerce. In [Schwabe+01c] a catalog framework is described with several abstract classes and hot spots to be specialized for each specific catalog. The abstract class Product has to be specialized in each catalog definition, where a whole hierarchy of product types can be defined. We do not know, a priori, how many levels of Product Types will be defined in each catalog.

We could see this framework as a meta-schema for catalogs. At each framework instantiation we can obtain a schema for a specific catalog and later an instance with all the products. For example, it is possible to define restrictions on all sub-classes of "Product", such as "define a context with all Product sub-classes that have a 'color' attribute, and whose instances have 'color'= 'blue'".

The OOHDM approach using RDF, RDF-Schema and DAML+OIL as described in this work can be useful to model all levels of this framework.

4.3 Specification of Adaptable Web Applications

Adaptable applications have the property that either contents, structure, navigation or interface change depending on a number of parameters, such as user identity, profile, or even location. Adaptation is typically achieved through rules, which can be expressed as queries over the schema, instances or both.

For example, common need is to pose questions such as:

```
If object belongs to "Student" sub-class "Beginner",
    then the link should be to "Content" instance "Introduction",
    else the link should be to "Content" instance "Concept A".
```

This is a typical query over the schema where we need to know the type of a specific resource.

4.4 Meta Schemas

As a last example we would like to mention one of our future related works: user defined metaclasses in DAML+OIL language. We plan to define OOHDM metaclasses for both conceptual and navigational models. We envision that this will help OOHDM developers create their own schema definitions according to the OOHDM approach.

5 Conclusion

In this paper, we have argued the need and convenience to have a more semantically rich specification language for both conceptual and navigation objects in Web applications. We have also shown the requirements for a query language that allows the definition of OOHDM-like primitives such as navigation nodes, access structures and contexts, in this new model. We have shown how RQL can be used to specify the mapping of these OOHDM primitives to the semantic web data model.

We are currently completing the definition of this model, following the same philosophy as OOHDM, but incorporating additional functionality as outlined in section 4. We are also investigating implementation frameworks and environments, such as the ICS-FORTH RDFSuite [ICS-FORTH02] and the Sesame Architecture [Sesame01].

6 References

[Abiteboul+00] Abiteboul, S.; Buneman, P.; Suciu, D.; *Data on the Web*, Morgan Kaufmann, 2000, ISBN 1-55860-622-X

[Berners-Lee00] Berners-Lee, T.; *Weaving the Web: The original design and ultimate destiny of the World Wide Web*, New York, NY: HarperCollins

[Gomez+00] Gómez, J.; Cachero, C.; Pastor O.; *Extending a Conceptual Modeling Approach to Web Application Design*, Proc. of the 12th International Conference CAISE 2000, LNCS 1789, Springer Verlag, pp. 79-93, 2000

[Cattell+00] Cattell, R.; Barry, D.; Berler, M.; Eastman, J.; Jordan, D.; Russell, C.; Schadow, O.; Stanienda, T.; Velez, F.; *The Object Database Standard ODMG 3.0*, Morgan Kaufmann, January 2000

[Ceri+2000] Ceri, S.; Fraternali, P.; Bongio, A.; *Web Modeling Language (WebML): a modeling language for designing Web sites;* Proc. of WWW9, and Computer Networks, 3 (1-6), 137-157 (2000).

[Garzotto+93] Garzotto, F.; Paolini, P.; Schwabe, D.; *HDM - A Model-Based Approach to Hypertext Application Design*, ACM Transactions on Information Systems 11, 1 (January 1993), pp. 1-26

[Gil+01] Gil, Y.; Ratnakar, V.; *Markup Languages: Comparison and Examples*, USC/Information Sciences Institute, TRELLIS project, http://trellis.semanticweb.org/expect/web/semanticweb/comparison.html

[Harmelen+01] van Harmelen, F.; Horrocks, I.; Patel-Schneider, P.; *Reference Description of the DAML+OIL (March 2001) Ontology Markup Language*, http://www.daml.org/2001/03/reference.html

[Henniker+00] Hennicker R.; Koch N.; *A UML-based Methodology for Hypermedia Design;* Proceedings of the Unified Modeling Language Conference, UML´2000, Evans A. and Kent S. (Eds.). LNCS 1939, Springer Verlag, pp. 410-424

[ICS-FORTH02] The ICS-FORTH; *RDFSuite: High-level Scalable Tools for the Semantic Web* , http://139.91.183.30:9090/RDF/index.html

[Isakowitz+95] Isakowitz, T.; Stohr, E.; Balasubramanian, P.; *RMM: A methodology for structuring hypermedia design*, Commun. ACM 38, 8 (August 1995), pp. 34-44

[Karvounarakis+02] Karvounarakis, G.; Alexaki, S.; Christophides, V.; Plexousakis, D.; Scholl, M.; *RQL: A Declarative Query Language for RDF*, The Eleventh International World Wide Web Conference (WWW2002), Honolulu, Hawaii, USA, May 7-11, 2002, http://139.91.183.30:9090/RDF/RQL/index.html

[OMG99] OMG; *Unified Modeling Language Specification version 1.3* (UML 1.3), June 1999

[Rossi+99] Rossi, G.; Schwabe, D.; Lyardet, F.; *Web Application Models Are More than Conceptual Models*, in Proceedings of ER'99 (Paris, France, November 1999), Springer, pp. 239-252

[Rumbaugh+91] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy F.; Lorensen, W.; *Object Oriented Modeling and Design*, Prentice Hall Inc., 1991

[Schwabe+98] Schwabe, D.; Rossi, G.; *An object-oriented approach to Web-based application design*, Theory and Practice of Object Systems (TAPOS) (October 1998), pp. 207-225

[Schwabe+01a] Schwabe, D.; Rossi, G.; Esmeraldo, L.; Lyardet: F. ; *Engineering Web Applications for reuse*, IEEE Multimedia 8(1) – Special Issue on Web Engineering, Jan-Mar 2001, pp. 20-31, ISBN 1070-986X

[Schwabe+01b]	Schwabe, D.; Rossi, G.; Esmeraldo, L.; Lyardet, F., *Web Design Frameworks: An Approach to Improve Reuse in Web Applications*, Lecture Notes in Computer Science (Hot Topics) 2016 - Proc. of the Second International Workshop on Web Engineering, WWW9 Conference, Springer Verlag, 2001, pp. 335-352
[Schwabe+01c]	Schwabe, D.; Medeiros, A. Laufer, C; Lima, F.; Condack, J.; Jacyntho, M.; *IBM Project Technical Report* (in portuguese), 2001
[Sesame01]	Sesame.aidministrator bv; *Sesame: A Generic Architecture for Storing and Querying RDF and RDF-Schema*, Technical Report, http://sesame.aidministrator.nl/, October 2001
[W3C98]	W3C; *A Discussion of the Relationship Between RDF-Schema and UML*, http://www.w3.org/TR/NOTE-rdf-uml/
[W3C99]	W3C; *Resource Description Framework (RDF) Model and Syntax Specification*, W3C Recommendation 22 February 1999, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
[W3C00a]	W3C ; *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C; Recommendation 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006
[W3C00b]	W3C; *Resource Description Framework (RDF) Schema Specification 1.0*, W3C; Candidate Recommendation 27 March 2000, http://www.w3.org/TR/2000/CR-rdf-schema-20000327/
[W3C01a]	W3C; *XML Schema Part 1: Structures*, W3C Recommendation 2 May 2001, http://www.w3.org/TR/xmlschema-1/
[W3C01b]	W3C; *XML Schema Part 2: Datatypes*, W3C Recommendation 02 May 2001, http://www.w3.org/TR/xmlschema-2/
[W3C02]	W3C; *RDF Primer*, W3C Working Draft 19 March 2002, http://www.w3.org/TR/2002/WD-rdf-primer-20020319/
[Wiederhold92]	Wiederhold, G. ;*Mediators in the Architecture of Future Information Systems*, IEEE Computer, March 1992