DAG EKENGREN
E92, KTH IT

1999-02-08

RDF

FOR

MULTIMEDIA BROKER

METADATA ON THE WEB

ABSTRACT

The Internet and the World Wide Web has made enormous amounts of information instantly accessible to millions of users. The sheer wealth of resources makes finding information difficult. Most of the data provided lacks metadata, such as classification and structural information. This report shows how RDF (Resource Description Framework) can be used to give metadata descriptions to resources (documents, images, sounds, products) on the World Wide Web. We describe the basics of metadata, RDF and RDF Schema and develop an RDF data model for a simple product. We then show how RDF metadata can be implemented on a web server. In this project we use the *Multimedia Broker* platform for the implementation, but RDF can be implemented on any web server with scripting capabilities. The report includes a discussion of future RDF clients and RDF servers. As an example of an RDF client we introduce a Java client capable of browsing and editing RDF documents with a graphical user interface.

1. INTRODUCTION

Searching for information on the World Wide Web often proves time-consuming and tedious. Finding the information you want is difficult to automate. Search engines indices information based on occurrences of word in bulk text documents. This can be illustrated by the following example:

> *Example:* You want to find information on documents *written by* William Shakespeare and type "William Shakespeare" into a search engine such as Lycos, Infoseek or Altavista. The search engine returns a large number of references to documents containing the words "William Shakespeare". Alta Vista reports more than 40 000 matches to a query on "William Shakespeare". Most of the matches are irrelevant to our search as they refer to documents *about* Shakespeare rather than *by* Shakespeare. The search engine may even return a match on this document!

Clearly we need some better way to describe our information so that search engines can give results with much better precision. The information describing information is referred to as *metadata*.

The *Multimedia Broker* developed at SITI[1] is a software architecture that would benefit from Metadata and is used as a case study in this report. *Multimedia Broker* is an EU funded research project that aims to integrate a number of multimedia techniques into an infrastructure, which will support publishers with new means of producing, structuring, disseminating and selling content and information products [MMBROK]. The system provides products and services to end users on the Internet by combining components stored in a database in real-time. The services and products can be adapted to individual users based on their user profiles and individual preferences. The profiles are built up from the history of actions requested by the user. The preferences are specific requirements provided by the user.

We have now realized the need of metadata and introduced a particular piece of software that would benefit from a metadata implementation. We haven't yet discussed what metadata *is*. The Macquarie Dictionary defines the prefix *Meta* as meaning "among", "together with", "after" or "behind". This suggests that metadata is data that comes together with other data to describe that data's use or interpretation. The metadata is a "fellow traveler[2]" of our data.

Metadata is often described as "data about data". A good example of metadata is the card system used in public libraries. The cards contain *descriptions* about the books in the library. In this case the books are our data and the cards are "data about our data".

The descriptions on every card follow a certain *schema*. The schema tells us what type of information we can expect to find on each card, i.e. the schema provides the criteria with which the books in our library are described. A good schema for libraries should probably contain elements like *Title*, *Creator* and *Publisher*.

The cards are metadata to the books and the schema is metadata to the cards. This suggests that the distinction between data and metadata is not always clear, rather it depends on the

[1] SITI – The Swedish Institute for Information Technology. http://www.siti.se

[2] *Sputnik* in Russian.

application. What is considered to be metadata in one application may be considered data in another.

One problem today is the great variety of standards and classifications for describing *resources*. Resources can be documents, images, sounds, basically any kind of data. There has been some work for providing simple metadata embedded in HTML[3] documents, but until now there have been no recommendations for how general resources should be described on the web. The problem is often that products or documents are described in existing databases, every database with its own classifications and representation. This is fine for closed systems that are used in a single company or organization. However, with the rise of the web the need for exporting data and metadata has emerged.

1.1. OBJECTIVES AND REQUIREMENTS

The data in existing databases should be mapped and packaged in a general way so that other systems can utilize it. Specifically, the Multimedia Broker needs to package its products with descriptions that can be interpreted by other systems. Some sort of metadata description framework is needed.

The metadata for the Multimedia broker should enable:

Resource Discovery: The products provided by the Multimedia Broker should be described and classified so they can be indexed by web search indexes (for example other brokers).

Resource Authoring: Users should be able to browse and search provided products. The descriptions should enable the user to compose products that are tailor-made for his/her needs.

Resource Exchange: It should be possible to exchange information and data between heterogeneous systems, that is systems that have their own internal representation of the data.

We want to describe the products and services so outside metadata consumers can discover the products and relationships between products. We also want to be able to describe and export products and services to other Multimedia Broker systems or similar systems from other vendors.

The metadata framework should be based on web standards to enable *interoperability* with other web systems, such as search indexes and web browsers. Solutions not dependent on any particular software vendor should be preferred. Classifications that are widely understood should be used.

Another objective is to implement a client that can read, parse and browse RDF descriptions provided by the Multimedia Broker as well as other RDF data sources.

1.2. RDF

XML[4] has emerged as *the* universal data format for the web. XML can be used to store and transport any kind of information, including metadata. XML is a grammar for creating mark-up languages, called XML applications. One such XML application is RDF, Resource Description Framework.

[3] Hypertext Markup Language

[4] XML – The Extensible Markup Language

RDF is a data model for describing *resources*. A resource in this case is anything you can refer to with a URI[5]. Examples of resources are documents, images and sounds. A resource could also be a more abstract thing, such as a service or a process, but it must then be represented by something that can be referred to by a URI, e.g. a document describing the resource.

RDF gives us is a data model for describing our resources and a standardized way of expressing that model in XML.

The World Wide Web Consortium (W3C) has working groups that deal with metadata on the web. The results of their efforts are the "RDF Model and Syntax" [RDF] and "RDF Schema" [RDFSCH] specifications. RDF is the web metadata framework that will be recommended by W3C.

1.3. APPROACH

The approach of this project is to use the RDF specification to describe the products and services provided by the Multimedia Broker system using the *Dublin Core* classification scheme (section 2.2). As a case study educational products of a fictitious company "Educational Products Inc." are described with RDF. The products described in this case were books containing math exercises. This gives us two levels of granularity, which is interesting for the RDF implementation.

Where Dublin Core is not sufficient to fully describe a product, we use a classification scheme, which we create ourselves in this project. We also look at the IMS project (section 2.4) which contains, among other things, a metadata classification scheme for learning objects. However, we decide not use it for the reasons stated in section 2.4.

The RDF description generation is very well integrated with the other components of the Multimedia Broker system. The descriptions are made both on a product level and at a finer grain of resolution.

The RDF client, a sample RDF consumer that is able to browse and navigate any RDF document, is implemented in Java, using XML and RDF parsers from IBM. We come to the conclusion that a hierarchical graphical representation is sufficient for displaying the RDF data model.

1.4. ORGANIZATION OF THIS REPORT

To fully appreciate this report, the reader should have some basic knowledge about the Internet. Some knowledge about HTML and XML helps. The reader doesn't have to know anything about metadata prior to reading this report. First some background is given and related technologies are presented. RDF is introduced and a general discussion on interoperability is given. The Dublin Core classification scheme is introduced. IMS is discussed and rejected. In chapters 3 and 4 we model our case study math exercise step by step. Readers mainly interested in the RDF data model may start from there. The chapters are both an introduction to the RDF Model and Syntax specification and to RDF Schema, as well as a guide to modeling a product in RDF. Examples of RDF applications are given. In chapter 5, we show a complete RDF instance and schema for our math exercise case study. The rest of the chapter describes how RDF can be implemented both on the client side and the server side. Specifically RDF is implemented on the Multimedia Broker server and on a Java client, called *RdfClient*. These sections assume some knowledge about web servers and Java. They are intended for readers who want to implement RDF on their own. In chapter 6 some ideas about the future of RDF applications are presented. Index servers, RDF browsers and Software agents are discussed.

[5] URI – Universal Resource Identifier. See section 2.3.

2. BACKGROUND

The history of metadata at the World Wide Web Consortium started in 1995 with PICS [PICS] (Platform for Internet Content Selection). PICS is a mechanism for communicating ratings of web pages from a server to clients: for example, whether a particular page contains a peer-reviewed research article, or was authored by an accredited researcher, or contains sex, nudity, violence, foul language, etc [RDFINTRO]. The target group for PICS was initially users such as parents worried about their children's web usage. Using PICS they could set their browsers to filter out any web pages not matching their criteria.

PICS is a restricted metadata framework that allows only certain things to be expressed. It is too limited for the metadata required by the objectives of our project. RDF on the other hand is a general metadata framework and a general knowledge representation mechanism for the web. [RDFINTRO]

RDF defines a framework for describing resources with metadata. As mentioned earlier, the distinction between data and metadata is not an absolute one. What is considered metadata by one application would be data to another. The RDF model is not restricted to storing metadata; it can be used to store any data. One example is the Mozilla web browser in which RDF is used for many purposes [MOZ1]: "It's a Swiss army knife and we will use it wherever it makes sense to use the RDF data model as a representation language." In particular RDF is used to store bookmarks, history lists, search results and site maps.

> "At the core RDF is a model for representing named properties and their values. These properties serve both to represent attributes of resources and to represent relationships between resources. The RDF data model is a syntax-independent way of representing RDF statements." [RDF]

Before we take on the RDF basics in chapter 3, let's address the "Why RDF, why not just XML?" issue: XML gives us the freedom to define any markup language. We can define our own tags and express constraints for the syntax of elements and attributes. So let's say we would like to define a markup language form describing resources on the web. We would need to define the elements that constitute our descriptions. We would have to figure out how to express resources and attributes. We would have to publish this new XML application so that others could use it. In effect, we would have to do what the W3C RDF Model and Syntax working group has already done. Remember that RDF *is* an XML application.

2.1. INTENDED USE OF RDF

RDF is a framework for metadata that facilitates automated processing of web resources [RDFFAQ]. RDF is a proposed standard that deals with issues such as *Resource Discovery* to provide better search engine capabilities; in *cataloging* for describing the content and content *relationships* available at a particular Web site, page, or digital library; by *intelligent software agents* to facilitate knowledge sharing and exchange; in *content rating*; in describing *collections* of pages that represent a single logical "document"; for describing *intellectual property rights* of Web pages, and many others. RDF with *digital signatures* will be key to building the "Web of Trust" for *electronic commerce*, *collaboration*, and other applications [RDFFAQ].

Examples of the intended use of RDF according to W3C:

Cataloging	Intellectual Property Rights
Intelligent Software Agents	Digital Signatures
Content Rating	Electronic Commerce
Collections of Pages	Collaboration

Table 1 Intended use of RDF according to W3C

The effective use of metadata in applications requires that we have common conventions and a common syntax, so that users beyond a small community can benefit from it. To allow for machine processing of metadata strict standards are necessary. The common denominator for all intended uses of RDF is *interoperability*. Interoperability is discussed further in section 2.2

RDF imposes structure and syntax, but does not stipulate semantics for each *resource description community* [RDF]. A resource description community can be an individual, an organization or a company. RDF lets each community create the elements needed in a standard manner. To allow for communities to define their own elements, RDF uses *schemas*[6]. Schemas are the set of properties, or metadata elements, defined by resource description communities. One of the key features of RDF is to be able to reuse schemas. The standard way of declaring schemas enables different resource communities to extend and reuse semantics from other communities' schemas. One such schema for RDF is *Dublin Core*, which is discussed later in this paper.

2.2. INTEROPERABILITY

The key word for web technologies is *interoperability*. Interoperability is required for applications where heterogeneous systems are involved. One example of interoperable data is HTML documents. Users can view HTML documents using all kinds of computer hardware, operating systems and browser software. If you follow the W3C recommendations for writing HTML documents, you can be quite certain that they are readable by anyone. However HTML documents are generally not machine-understandable, they are intended to be processed by the human eye and the human brain.

The next step for the web is machine-understandable data. For this, we need interoperability on (at least) three levels:

[6] Schemas are sometimes called *Vocabularies*.

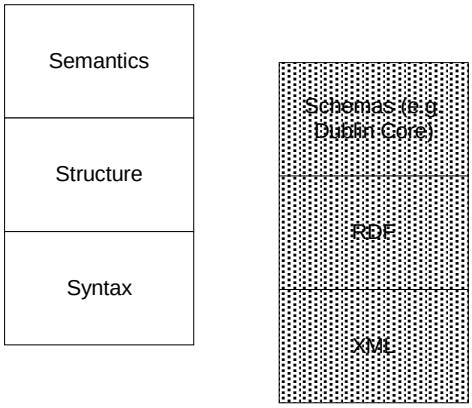


Figure 1: Machine-understandable documents require interoperability on three levels.

The lowest level, *syntax*, provides the required foundation for interoperability. Syntax determines how the documents should be written, for example if they use markup language. Examples of technologies that provide interoperability on this level are SGML and XML.

Interoperability on the st*ructure* level deals with how units of information (resources) are structured and how relationships between them can be expressed. RDF operates on this level. RDF was inspired by *Structured Maps*, a technology invented before the web.

Structured Maps are based on Topic Navigation Maps [ISO13250] defined by the SGML community. Structured Maps provide a layer of semantics and relationships on top of existing data. References to documents can be typed with for example author-of or written-by relationships. A Structured Map can be compared to a normal road map. It presents a *view* of the information. A road map is a simplified view of the world. Depending on our interests, we can have different maps covering the same part of the world, emphasizing different aspects of the existing information. For example, one could have a map that shows all McDonald's restaurants and another map with topological data. Structured Maps inspired the work on RDF. [DELCA]

Interoperability on the *semantic* level requires agreements on the semantics expressed. This requires that both the producer and the consumer of the information agree on what the semantics of an image, a car or a title is. Classification schemes, such as Dublin Core operate on this level. *The Dublin Core (DC)* is a 15-element metadata element set intended to facilitate discovery of resources. Originally conceived for author-generated description of web resources, it has also attracted the attention of formal resource description communities such as museums and libraries. [DCPURL]

The Dublin Core has been discussed through a number of international workshops [DC4] [DC5]. Experts from many different communities have tried to reach consensus around what elements to use and their exact semantics. This is after three years still an ongoing effort.

Design principles for the Dublin Core are [DCPURL] are *Simplicity*, *Semantic Interoperability*, *International Consensus* and *Flexibility*. Simplicity is important for enabling widespread use. It should be easy for non-catalogers to understand the elements involved. The flexibility makes it possible to encode additional structure and more elaborate semantics where needed without sacrificing interoperability with older software.

The Dublin Core has historically had a close relationship with RDF. At the second DC workshop in Warwick, a conceptual foundation for a metadata framework was laid, the so-called *Warwick Framework* [WP] [DC5]. This framework, along with the *Meta Content Framework* [MCF]

formed the nucleus for the development of RDF. The Dublin Core and the RDF working groups share a number of members, and the co-evolution of these projects has added to the progress of each. According to a Dublin Core Workgroup decision (September 1998) "the RDF data model is the foundation for the DC data model".

The Dublin Core Elements are [DCPURL]:

Content	Intellectual Property	Instantiation
Title	Creator	Date
Subject	Publisher	Type
Description	Contributor	Format
Source	Rights	Identifier
Language		
Relation		
Coverage		

Table 2: The fifteen Elements of Dublin Core

Each element is optional and repeatable. Furthermore, metadata elements may appear in any order, and with no significance being attached to that order.

The Dublin Core initiative has achieved wide acceptance on the Internet. It is a vocabulary that is well suited for describing online resources. It is a good idea to support as many Dublin Core elements as possible in any RDF metadata implementation. The number of supported elements is often restricted by the data available in existing databases.

2.3. RESOURCE DISCOVERY AND IDENTIFICATION

A *Resource* is any real or conceptual object that can be identified. *Discovery* involves *finding* and *retrieving* of resources that are relevant to the *user* of the system. Users are usually human users but may be automated processes that have a need to fulfill a resource discovery requirement [RDU].

One reason for imposing *structure* on information is to enable *Resource Discovery* [RDU], i.e. to make data accessible and easy to find and to label resources so that their relevancy can be estimated. Ideally this should be done automatically by agents or search engines with minimal effort from the user. In order for this to happen on the World Wide Web we need standards for all the levels of interoperability (section 2.2).

Resource discovery also involves actually retrieving the discovered resource. To retrieve the resource we need some means of identification. Every resource available on the Web has an address that may be encoded by a *Universal Resource Identifier*, or *URI* [HTML]:

URIs typically consist of three pieces: 1) The naming scheme of the mechanism used to access the resource, 2) the name of the machine hosting the resource and 3) the name of the resource itself, given as a path.

Example of a URI is http://www.ekengren.com/rdf/default.asp

2.4. IMS

The IMS (Instructional Management System) specification is a huge document covering the technical details for an online learning system. Our interest in IMS stems from the fact that our test case implementation is about learning material. Here is a list of the requirements to be *IMS*

compliant: *Group Management*, *Personal Profile Management*, *Activity Management*, *Assessment and Certification Management*, *Content Management*, *Commerce and Licensing Management*, *Security Management*, *Technical Administration Management*.

Creating an IMS compliant system would require significant additions to the Multimedia Broker architecture, which definitely are beyond the scope of this project. Instead we focused on the metadata parts of IMS. After a study of IMS (Appendix E) we decided not to use IMS for the following reasons:

1. The IMS system is more of a metadata infrastructure than a mere classification scheme. The development of IMS metadata predates that of RDF and it is unclear how (and if) IMS should be combined with RDF. RDF and DC (Dublin Core) on the other hand focus on simplicity and ease of implementation. Also, the RDF and the DC working groups are committed to making their parts fit together.

2. We could not make the system IMS compliant because many parts required for such a system isn't supported by the Multimedia Broker architecture. Also, the case study database we use would require much new data just to support the IMS metadata part. It was not acceptable to redesign the database to achieve IMS metadata compliance.

3. Given that the IMS system is for learning material only and Dublin Core can be used for all kinds of resources, Dublin Core is much more likely to become widely used on the web. This increases the probability of our Multimedia Products ever being discovered.

For readers interested in the metadata parts of IMS there is a short survey in Appendix E.

Rather than trying to describe the RDF Model and Syntax in every detail we will provide a step by step explanation of how the description of a math exercise product was modeled in our educational products case study. In each step we will provide examples of how the RDF data model is serialized to XML. We will also try to explain and highlight interesting features. For more details- and formalism, please refer to Appendix B, or the W3C RDF Model and Syntax draft [RDF]. If you are unfamiliar with XML, please refer to Appendix A.

3.1. SIMPLE STATEMENTS – NODES AND ARCS

The foundation for RDF is a model for representing resources with named properties. The same data model can be expressed in several ways: with a graph of nodes and arcs, serialized to XML or as a collection of triples (see Appendix B for details).

The model consists of three object types:

1. *Resources* are the "things" we describe. A resource can be anything we can refer to with a URI. In fact, URIs can be used to reference *anything*. On the web RDF will probably be used to describe documents that are retrieved with HTTP. URLs (which is a subset of URIs) can identify such documents. An example of a URL is http://www.ekengren.com/rdf/math_exercise

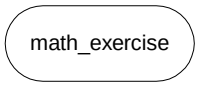


Figure 2: A math exercise resource. Only part of the URL is included for clarity.

2. *Properties* are what we describe our resources with. In the RDF data model we can express properties as an arc, or an arrow. The properties are named, and the names define their meaning.

Figure 3: A named property. Good for giving a title to our math exercise.

3. *Statements*. A statement is a triple consisting of a resource, a named property and a value. The value of the property can be either a string (called *literal*) or another resource. Both values that are strings and values that are other resources are represented as nodes in the RDF data model. Now we are ready to describe our math exercise, using RDF:

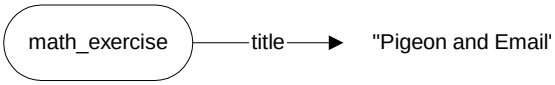


Figure 4: This is our first RDF statement!

That's really all there is to it: Nodes and Archs! Now, let's try to express our math exercise description in XML:

```
<rdf:Description about="http://www.ekengren.com/rdf/math_exercise">
    <title>Pigeon and Email</title>
</rdf:Description>
```

The *rdf:Description* element creates a node that represents our math exercise resource. The next line creates an arch and a *literal* resource. We can make our description slightly more complex by adding more properties:

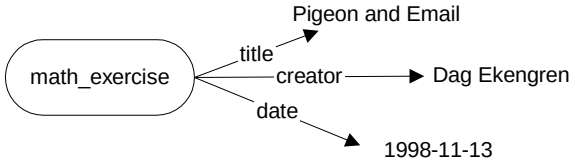


Figure 5: More statements about our math exercise

The XML serialization is:

```
<rdf:Description about="http://www.ekengren.com/rdf/math_exercise">
    <title>Pigeon and Email</title>
    <creator>Dag Ekengren</creator>
    <date>1998-11-13</date>
</rdf:Description>
```

3.2. USE OF THE DATA MODEL IN PARSERS

There is a number of syntactic variations possible when serializing the same RDF description to XML. Things can be in different order and there are syntactic variations such as abbreviation (more on this in section 3.8). The data model can be used internally in an RDF parser to determine if two RDF descriptions that are serialized to XML are the same semantically. The following XML serializations represent the same data model:

```
<rdf:Description about="http://www.ekengren.com/rdf/math_exercise">
    <title>Pigeon and Email</title>
    <creator>Dag Ekengren</creator>
    <date>1998-11-13</date>
</rdf:Description>
```

and

```
<rdf:Description about="http://www.ekengren.com/rdf/math_exercise"
    date="1998-11-13" title="Pigeon and Email"
    creator="Dag Ekengren" />
```

3.3. ADDING A SCHEMA USING XML NAMESPACES

Great, our math exercise already has more metadata than most resources on the web. However, there is a small problem with our description. We have used the elements *title*, *creator*

and *date* from some schema we implicitly invented. *We* know the meaning of these elements, but we would like the rest of the world to know as well.

What definition of *title* did we use when we chose the title "Pigeon and Email"? What is a *creator*? Is it the company printing the books with math exercises or is it the author of the exercise? In what format is the date stored? These are questions that a machine or a human being would like answers to when they see our descriptions. Our description is of little value to them if we can't provide this *metametadata* called *schema*.

Fortunately, the XML namespace[7] facility comes to our rescue. It lets us associate a URL with each property. The resource referenced by this URL may be a human-readable resource that describes the schema used. It may also be a schema in some machine understandable format. The RDF Schema specification provides such a format [RDFSCH]. For now it will suffice to just give a reference to some imaginary schema document. XML namespaces associates a schema URL with a prefix that we use in our property names:

```
<?xml version="1.0"?>
<rdf:RDF   xmlns:rdf="http://www.w3.org/rdf-syntax"
           xmlns:dc="http://purl.com/dc/elements/1.0" >
    <rdf:Description about="http://www.ekengren.com/rdf/math_exercise">
        <dc:title>Pigeon and Email</dc:title>
        <dc:creator>Dag Ekengren</dc:creator>
        <dc:date>1998-11-13</dc:date>
    </rdf:Description>
</rdf:RDF>
```

Here we have assigned the Dublin Core schema to the prefix *dc*. Please note that we have defined a schema *rdf-syntax* for the RDF-specific elements. An ordinary XML parser doesn't know anything about RDF and treats elements in the *rdf* namespace no differently from elements in for example the *dc* namespace. An RDF parser[8] however knows the semantics of elements such as *rdf:Description*. Also note that the RDF document is enclosed in an *rdf:RDF* element.

We can use several different schemas in the same RDF description. The schemas may be defined by different schema authorities. The schemas can be used even if some elements exist in both schemas, because a different prefix is assigned each schema. Two different schemas should not use the same prefix to avoid ambiguity.

In the math exercise sample we will create a schema of our own for properties and classes not supported by the Dublin Core. This schema will have the prefix *edu*, as in "educational".

3.4. USING RESOURCES AS VALUES

Up to this point, we have used simple literal values. One of the benefits of RDF is that we can use resources to describe resources. We could let the value of the *dc:creator* property be a resource describing a person, in this case me. This resource could contain the name, address, telephone number of the creator. It may even contain digital photographs and other complex data.

[7] Please refer to Appendix A for details on XML namespaces.

[8] RDF parsers are often implemented on top of XML parsers. Because an RDF document is an XML application it can be parsed by an XML parser, which builds a tree structure that represents the document. The RDF parser can then parse that tree and look for elements in the *rdf* namespace.

In our math exercise, it will suffice to use a literal value for *dc:creator*. We could add a picture as a description of the math exercise though:

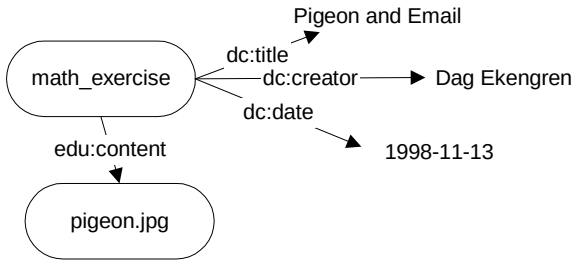


Figure 6: Showing off a resource property value.

```
<?xml version="1.0"?>
<rdf:RDF   xmlns:rdf="http://www.w3.org/rdf-syntax"
           xmlns:dc="http://purl.com/dc/elements/1.0"
           xmlns:edu="http://www.ekengren.com/rdf/schemas/eduschema" >
    <rdf:Description about="http://www.ekengren.com/rdf/math_exercise">
        <dc:title>Pigeon and Email</dc:title>
        <dc:creator>Dag Ekengren</dc:creator>
        <dc:date>1998-11-13</dc:date>
        <edu:image resource="http://www.ekengren.com/rdf/images/pigeon.jpg" />
    </rdf:Description>
</rdf:RDF>
```

The example above demonstrates the syntax for adding a *resource as the value* of a property. The resource value may be in the same document as the RDF description or in another document (as in this example). Remember that RDF descriptions are also resources. This means that the value of a property may be an RDF description. This facility allows for highly structured metadata.

3.5. COLLECTIONS – BAG, SEQ AND ALT

In RDF you often want to refer to a collection of resources. If you're describing a course you may want to include a students property which refers to a collection of the students that attend the course. If you're describing a web page you may want to refer to a collection of web servers where the page is mirrored.

There are three different types of collection objects in RDF:

1. *Bag* is used to declare a collection of resources where the ordering of the resources is not significant. For example, in the collection of students the ordering of the students that attend the course may not be important.

2. *Seq* is used to declare a collection where the ordering of the resources *is* indeed significant. *Seq* could be used if we were describing an exam and wanted an ordering of the students according to their results.

3. *Alt* is used to give alternative values to a single value. An application that is using a property whose value is an *Alt*-collection can choose one (and only one) of the values in the collection.

In our math exercise in Figure 6, we have a *edu:content* property that refers to a single picture resource. What if our math exercise has more than one picture associated to it? Or an audio clip for that matter? It would be nice collect these in a bag resource. In the following example we have added a bag node as the value of the *edu:content* property:

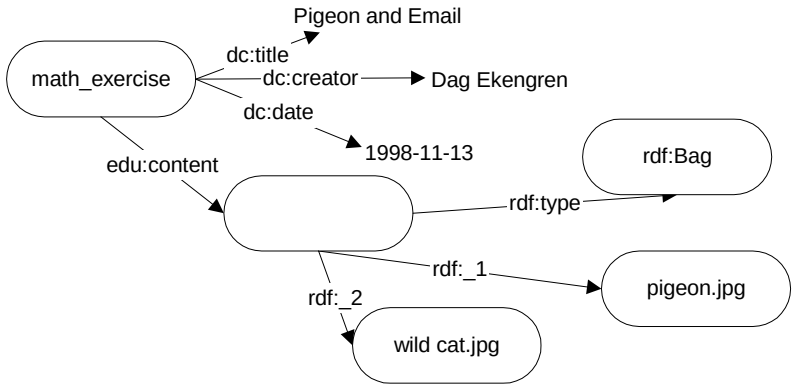


Figure 7: We have used a Bag to collect wild cat.jpg and pigeon.jpg.

Note that the bag node has no identifier and the node appears "empty" in the graph över. We can add an ID-attribute to the Bag if we want to refer to it from other statements. The content in the bag is referred to by the *rdf:_n*-properties. The *rdf:type* property is used to declare that a resource belongs to a certain class of resources, in this case the rdf:Bag class. Classes and the *rdf:type*-property are covered in section 4.3. The XML serialization of the example in Figure 7 is:

```
<?xml version="1.0"?>
<rdf:RDF   xmlns:rdf="http://www.w3.org/rdf-syntax"
           xmlns:dc="http://purl.com/dc/elements/1.0"
           xmlns:edu="http://www.ekengren.com/rdf/schemas/eduschema" >
    <rdf:Description about="http://www.ekengren.com/rdf/math_exercise">
        <dc:title>Pigeon and Email</dc:title>
        <dc:creator>Dag Ekengren</dc:creator>
        <dc:date>1998-11-13</dc:date>
        <edu:content>
            <rdf:Bag>
                <rdf:li resource = http://www.ekengren.com/rdf/images/pigeon.jpg />
            </rdf:Bag>
        </edu:content>
    </rdf:Description>
</rdf:RDF>
```

The rdf:_n properties in the data model appear as `<li>`-tags in the XML serialization.

3.6. STATEMENTS ABOUT STATEMENTS – REIFICATION

It is sometimes interesting to say something about statements, i.e. give statements about statements. We could then express who made the description and when. Perhaps we trust descriptions from a respected web page rating company more than descriptions from some anonymous individual.

How can statements about statements be expressed in the RDF data model? We need to be able to identify a statement so that we can refer to it. This can be achieved by a process called *reification*.

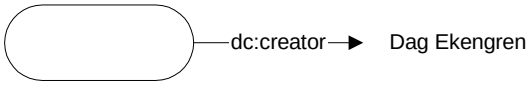


Figure 8: No reification here. This statement cannot be referred to. Since no judgement can be placed on this statement, the statement is considered a fact.

The above statement is reified to:

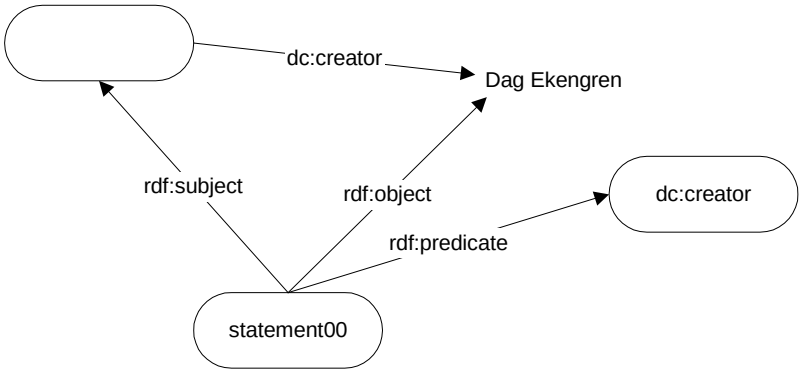


Figure 9: The statement has been reified

This is still the same statement as the one above, but we can now refer to it by the id "statement00". Now let's give a statement about this statement:

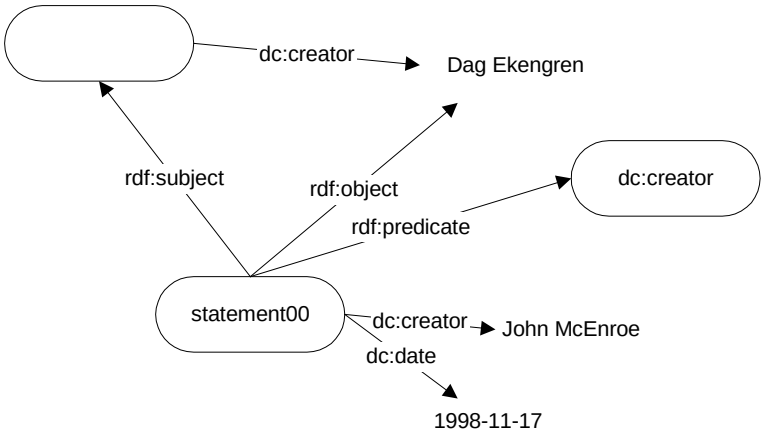


Figure 10: Statements about a statement. Who says that the creator of the resource is Dag Ekengren?

The serialization to XML is very simple:

```
<?xml version="1.0"?>
<rdf:RDF   xmlns:rdf="http://www.w3.org/rdf-syntax"
           xmlns:dc="http://purl.com/dc/elements/1.0" />
    <rdf:Description>
        <dc:creator ID="statement00">Dag Ekengren</dc:creator>
    </rdf:Description>

    <rdf:Description about="#statement00">
        <dc:creator>John McEnroe</dc:creator>
        <dc:date>1998-11-17</dc:date>
```

```
    </rdf:Description>
</rdf:RDF>
```

The statements with properties *rdf:subject*, *rdf:object* and *rdf:predicate* are implicitly constructed by the RDF parser.

3.7. THE DESCRIPTION CONSTRUCT

The description lets us bundle several statements. We can refer to the description to make statements about the collection of statements as a whole. The description construct is simple in XML but the data model is quite complex. Each statement is reified and the new nodes (such as the one called "statement00" in Figure 10 are collected in a bag. The bag is given an id called *bagID*.

Let's return to the math exercise example in Figure 5. We would now like to say who made the description. We reify all the statements and put them in a bag:

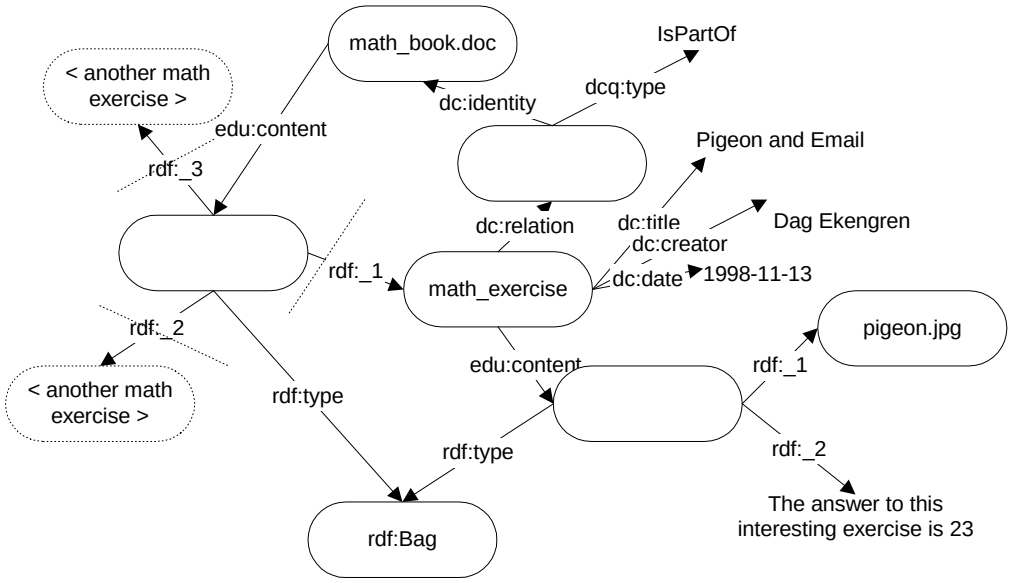


Figure 11: The Description construct. The collection of statements called a description does not need a special construct in the data model. A bag container is used to indicate that a set of statements belongs to the same description.

Serialization to XML is simple because the reification and bag nodes are created implicitly by the parser. The only new thing is the *bagID* attribute in the rdf:Description-element.

```
<rdf:Description about="http://www.ekengren.com/rdf/math_exercise"
           bagID="description00">
    <dc:title>Pigeon and Email</dc:title>
    <dc:creator>Dag Ekengren</dc:creator>
</rdf:Description>
```

We can now make statements about the description identified by "description00".

3.8. ABBREVIATED SYNTAX

RDF allows an abbreviated syntax where the statements are XML attributes in the rdf:Description element. Abbreviation is useful for properties that aren't repeated within a description and where the values of those properties are literals. In this case, the properties may be written as XML attributes of the description element.

The RDF description in 3.5 can be abbreviated to:

```
<?xml version="1.0"?>
<rdf:RDF    xmlns:rdf="http://www.w3.org/rdf-syntax"
            xmlns:dc="http://purl.com/dc/elements/1.0"
            xmlns:edu="http://www.ekengren.com/rdf/schemas/eduschema" >
    <rdf:Description about="http://www.ekengren.com/rdf/math_exercise"
        dc:title="Pigeon and Email" dc:creator="Dag Ekengren"
        dc:date="1998-11-13">
        <edu:content>
            <rdf:Bag>
                <rdf:li resource = "http://www.ekengren.com/rdf/images/pigeon.jpg"
/>
                <rdf:li>The answer to this interesting exercise is 23</rdf:li>
            </rdf:Bag>
        </edu:content>
    </rdf:Description>
</rdf:RDF>
```

4.1. QUALIFYING PROPERTIES – NON-BINARY RELATIONS

The RDF data model intrinsically only supports binary relations; that is, a statement specifies a relation between to resources. There is however a recommended way [RDF] to represent higher arity relations in RDF using just binary relations. The recommended technique is to use an intermediate resource with additional properties of this resource giving the remaining relations.

An example of non-binary relations is when we want to use qualified properties. The Dublin Core *relation* property is an example where this is useful. The *relation* property can be used to indicate a relationship between two resources. We often want to be a little more specific and say something about what *kind* of relationship the resources have. The way to do this is to *qualify* the *relation* property.

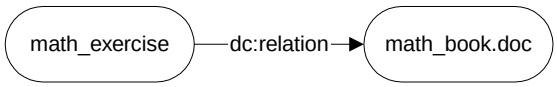


Figure 12: An unqualified dc:relation. What kind of relationship do the resources have?

The dc:relation types come in pairs depending on which "side" the described resource are:

IsPartOf	HasPart
IsVersionOf	HasVersion
IsFormatOf	HasFormat
References	IsReferencedBy
IsBasedOn	IsBasisFor
Requires	IsRequiredBy

After qualification the data model becomes:

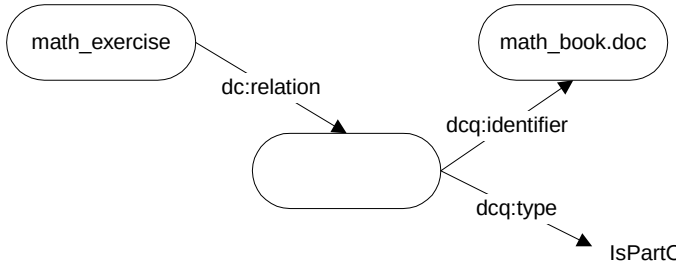


Figure 13: A qualified dc:relation. The relationship consists in the math exercise being part of a math book A new unnamed node has been created. The namespace prefix for DC qualifiers is dcq.

The XML serialization is:

```
<rdf:Description about="http://www.ekengren.com/rdf/math_exercise">
    <dc:relation>
        <rdf:Description>
            <dcq:identifier rdf:resource =
"http://www.ekengren.com/rdf/math_book.doc" />
            <dcq:type>IsPartOf</dcq:type>
        </rdf:Description>
    </dc:relation>
    <dc:creator>Dag Ekengren</dc:creator>
</rdf:Description>
```

The dc:date-element we have used throughout this example can also be qualified to further specify what meaning this date has for the resource. Is it the date of creation or is it date the resource became valid? Let's try it:

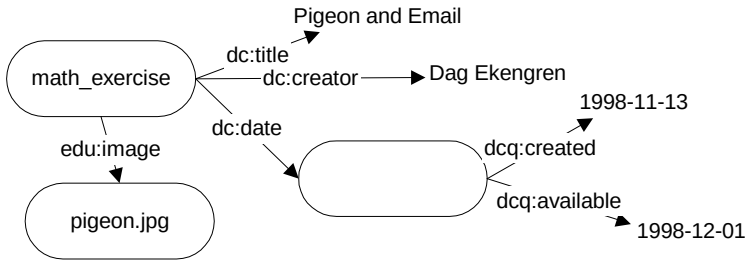


Figure 14: We have qualified the dc:date statement. Great stuff!

The Dublin Core workgroup is working on qualifiers other DC elements as well. Refer to http://purl.org/DC/documents/working_drafts/ for the latest details.

4.2. MATH EXERCISES IN A CONTEXT

One of the interesting things with RDF is the ability to describe resources with different levels of granularity. Some users may want very detailed descriptions where a single resource refers to other descriptions, while others are happy with a shorter description that is easier to manage.

In our educational products case study, we sell books with math exercises. We can describe a book as a complete product or as a product consisting of a number of math exercises, each with its own descriptions. The math exercise descriptions may be placed inline the RDF document that describes the book or it may be linked to the book with rdf:resource-tags. The advantage with the former is simplicity. You can get the entire description, including descriptions for all the math exercises in a single GET-request to the web server. The other approach has the advantage

that the user can choose to get only the book description and perhaps a description to a few of the math exercises.

To achieve this link between math books and math exercises we can use a *dc:relation*-statement as was mentioned in 4.1. Of course, our math exercise can be part of other math books as well, i.e. many math books could refer to our math exercise using a *edu:content*-statement. The dc:relation may not be very useful if the exercise belongs to several books.

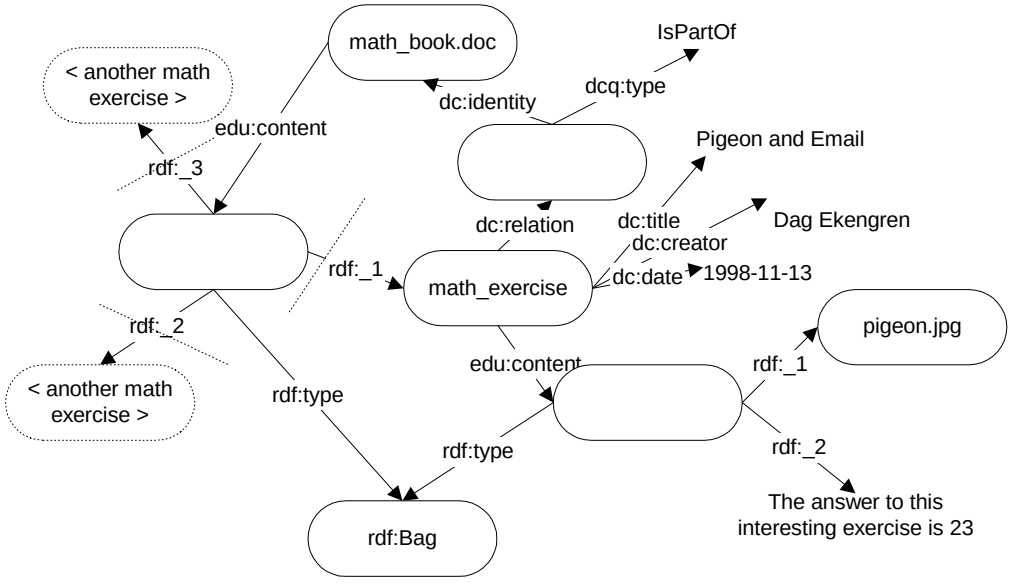


Figure 15: Math exercise in a context. The math exercise is part of a larger structure, in this case a book. The dotted lines are URL-boundaries if the math exercises are not inline in the RDF document describing the math book.

4.3. RDF SCHEMAS

We have already touched upon RDF Schemas in 3.3 and we're revisiting the subject here. The XML namespace facility allows us to associate a prefix with a URL that defines a schema. We haven't yet looked into how that schema can be defined.

So what is a schema anyway? It is where we define our properties and *classes*. We can give human readable comments and define machine-understandable constraints on their use. We can declare that a resource belongs to a particular class as we have already done in section 3.5 and in Figure 15 above. As you can see in Figure 15, two unnamed (empty) nodes are of the class *rdf:Bag*.

There are many ways in which we could declare our schemas. The RDF Schema Specification [RDFSCH] uses the RDF data model not only for describing resources, but for expressing schemas as well.

4.3.1. TYPE SYSTEM – CLASSES AND PROPERTIES

Let's define some classes for our math exercise example. A good place to start would be to define a math exercise class, called *edu:Exercise* and define our *instance* "math_exercise" to be an instance of that class.

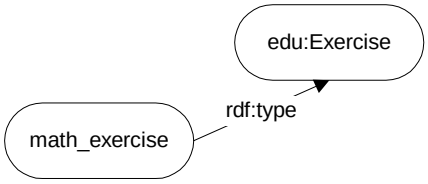


Figure 16: We have defined our math exercise instance to be of the class edu:Exercise. We can have lots of math exercise instances and they will all be of the same class.

This can be serialized to:

```
<?xml version="1.0"?>
<rdf:RDF   xmlns:rdf="http://www.w3.org/rdf-syntax"
           xmlns:dc="http://purl.com/dc/elements/1.0"
           xmlns:edu="http://www.ekengren.com/rdf/schemas/eduschema" >
    <rdf:Description about="http://www.ekengren.com/rdf/math_exercise">
        <rdf:type resource =
"http://www.ekengren.com/rdf/schemas/eduschema#Exercise" >
        <dc:title>Pigeon and Email</dc:title>
    </rdf:Description>
</rdf:RDF>
```

Because RDF Schema is expressed using the RDF data model, it seamlessly extends our RDF data model with the schema definitions. In the data model, the *rdf:type*-property doesn't differ from other properties such as dc:title or dc:subject. There is a special construct for expressing the class of an instance in the XML syntax: This construct makes use of the XML namespace facility:

```
<?xml version="1.0"?>
<rdf:RDF   xmlns:rdf="http://www.w3.org/rdf-syntax"
           xmlns:dc="http://purl.com/dc/elements/1.0"
           xmlns:edu="http://www.ekengren.com/rdf/schemas/eduschema" />
    <edu:Exercise about="http://www.ekengren.com/rdf/math_exercise">
        <dc:title>Pigeon and Email</dc:title>
    </edu:Exercise>
</rdf:RDF>
```

We have replaced the rdf:Description element with a edu:Exercise element and removed the rdf:type property. This doesn't in any way change the data model, hence the serializations are equivalent.

Now, let's see how the edu:Exercise class is defined in the http://www.ekengren.com/rdf/schemas/eduschema. Well, we create a node that has the ID of Exercise by creating a description-element with the ID-attribute, but without the about-attribute. In effect, this is not a description about a resource, rather it is a resource in itself with the specified ID. We declare this resource to be of the class rdfs:Class. The rdfs:Class-resource, is defined as part of the RDF Schema machinery. Every RDF data model includes this resource (and a few other resources) implicitly.

The edu:Exercise node in Figure 16 is defined in eduschema as:

```
<rdf:Description ID="Exercise">
    <rdf:type resource = "http://www.w3.org/TR/WD-rdf-schema#Class"/>
</rdf:Description>
```

Or, using the more compact form, with rdfs defined as the RDF Schema namespace:

```
<rdfs:Class ID="Exercise"/>
```

Properties are declared in a similar way, using the built-in rdf:Property-resource. The recommended convention is to use the first letter capitalized in class names and non-capitalized in property names.

```
<rdf:Property ID="title"/>
```

There are a few "Core Classes", "Core Properties" and "Core Constraints" defined in the RDF Schema specification [RDFSCH]. There are many scenarios where these simple mechanisms are not adequate; a more general schema mapping mechanism for RDF may be developed in future W3C activity [RDFSCH].

4.3.2. CONSTRAINTS

We can place constraints on properties by declaring in what *domain* a particular property is relevant, i.e. on what class of resources it can be used. We can also constrain the *range* of a property by declaring the class allowed for the resource values for the property.

RDF Schema focuses on properties rather than classes. This means that the RDF Schema Specification currently does *not* introduce a way of specifying what properties are required to describe an instance of a particular class. We cannot declare that an instance of the class edu:Exercise must include the properties *title* and *creator*. Also, we cannot in any way prevent people that use *eduschema* from describing instances of edu:Exercise with unknown properties from other schemas. The advantages and disadvantages of this are discussed further in section 4.4 where DCD (Document Content Description) is introduced.

A more complete schema for our math exercise looks like this:

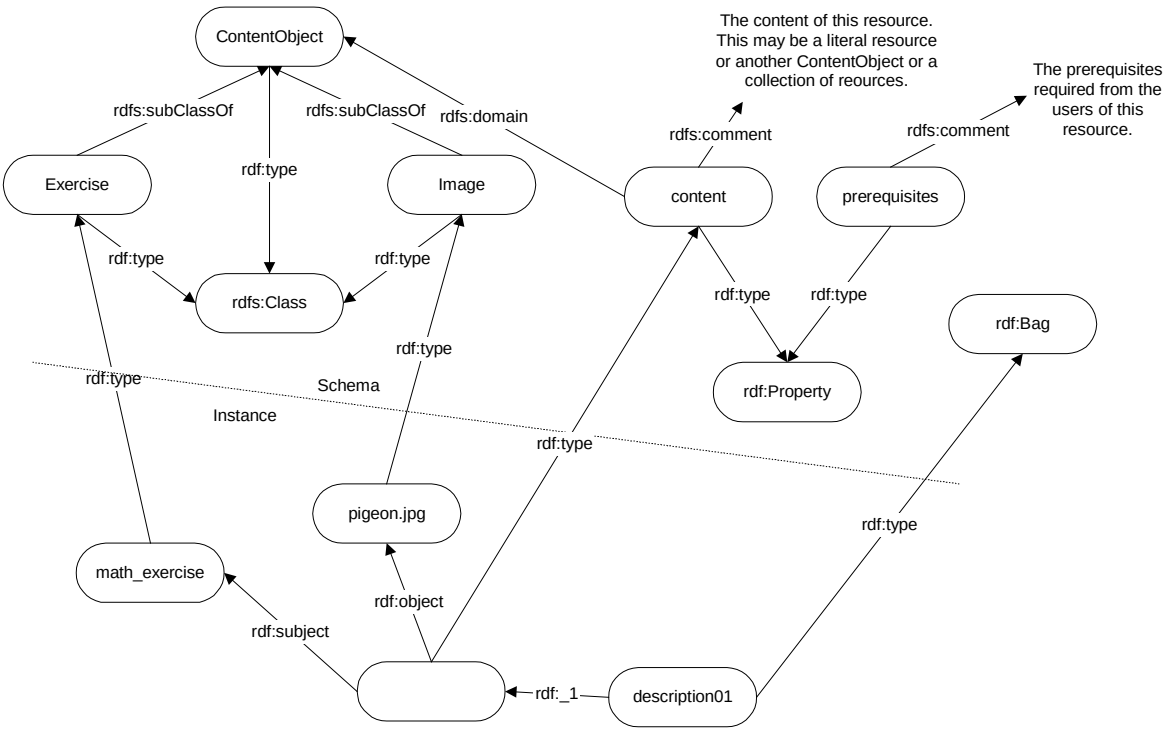


Figure 17: This is part of the schema definition for eduschema. Note that the schema is part of the RDF data model for the math exercise. We have defined Exercise *and* Image *as subclasses of a* ContentObject *class and we have constrained the* content *property so that it can only be used on Instances of ContentObject (or subclasses thereof). The dotted line is the division between schema and instance.*

We have used the *edu:content*-property to describe our math exercise. Apparently, our math exercise contains an instance of the class *edu:Image*. The *edu:content*-property has been constrained to be used only on instances of *ContentObject* (or subclasses of *ContentObject*). This constraint is expressed with the *rdfs:domain* property.

A problem with the current RDF Schema specification is that only one *rdfs:range* statement may be given to a property. In this case we would like the *edu:content* statements to have a value that is *either* an instance of the class *ContentObject* or a string. In the RDF Schema specification, strings are instances of the class *rdfs:Literal* which is implicitly declared by the RDF Schema machinery. If we constrain *edu:content* to have only ContentObject values then the following construct is invalid:

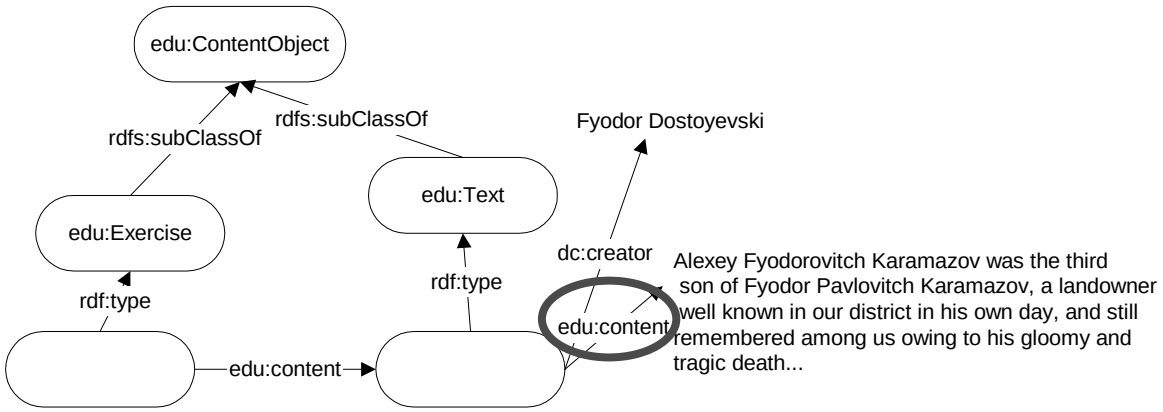


Figure 18 The edu:content property to the right has a string value (class rdf:Literal). This invalid if the edu:content property's range is ContentObject.

A workaround for this problem is to give a "human readable" comment in the schema instead of a machine understandable range-constraint: *"edu:content is the content of this resource. The content may be a literal resource or another ContentObject or a collection of resources."*

4.3.3. VOCABULARIES VS. SCHEMAS

There is some confusion as to what the difference is between vocabularies and schemas. [RDFSCH] states:

"The phrase RDF vocabulary is used here to refer to those resources which evolve over time; RDF Schema is used to denote those resources which constitute the particular (unchanging) versions of an RDF vocabulary at any point of time. Thus we might talk about the evolution of the Dublin Core vocabulary. Each version of the Dublin Core *vocabulary* would be a different RDF *schema*, and would have a corresponding RDF model and concrete syntactic representation."

4.4. DOCUMENT CONTENT DESCRIPTION – DCD

DCD was submitted to the World Wide Web Consortium (W3C) by IBM and Microsoft [DCD]. The document proposes a structural schema facility for specifying rules covering the structure and content of XML documents. As we understand the specification [DCD], DCD is a grammar for expressing RDF vocabularies, rather than an RDF vocabulary in itself. This means that we would refer to a DCD document with our namespace declarations in our RDF documents (section 3.3), i.e. it's schema declarations.

DCD is an interesting alternative to the *RDF Schema* specification [RDFSCH]. Here is a comparison of features:

	RDF Schema	DCD
Approach	Property based	Class based
Syntax	RDF	XML
Constraints on classes	No	Yes
Constrains on properties	Yes	No
Primitive data types	No[9]	Yes[10]

Table 3: Comparison of features: RDF Schema vs. DCD.

A Document Content Description (DCD) is a set of properties used to constrain the types of elements and names of attributes that may appear in an XML document, the contents of the elements, and the values of the attributes. If the XML document follows the RDF model, DCD puts constraints on allowed and required statements in the declared classes.

Here is an example of a DCD schema:

```
<DCD>
    <ElementDef Type="content"/>

    <ElementDef Type="ContentObject" Model="Elements" Content="Open">
        <Description>This is the ContentObject class</Description>
        <Group RDF:Order="Bag">
            <Group Occurs="Required">
                <Element>dc:title</Element>
                <Element>dc:creator</Element>
                <Element>dc:date</Element>
            </Group>
            <Group Occurs="Optional">
                <Element>content</Element>
            </Group>
        </Group>
    </ElementDef>

    <ElementDef Type="Exercise" Model="Elements" Content="Open">
        <Description>This is the Exercise class. It inherits from
ContentObject</Description>
        <Extends Type="ContentObject"/>
    </ElementDef>
</DCD>
```

The above schema declares a *ContentObject* class which should be described by the required elements *dc:title, dc:creator* and *dc:date* (defined in some external schema). The content object can be described with the optional *content* element (declared in this schema). The attribute *Content="Open"* declares that instances of this class can also be described by other elements, defined in this or an external schema.

As Table 3 suggests, RDF Schema and DCD are quite different. RDF Schemas are expressed using RDF Model and Syntax and are close to RDF "in spirit", while DCDs use their own XML syntax and are much closer in spirit to classical databases. The reasons for this conclusion are:

1. *RDF Schema* focuses on *properties*. You declare a *vocabulary* (schema) which is a collection of properties and constraints on those properties. It's possible to constrain the instances

[9] RDF Schema may be extended in the future to support data types.

[10] Examples of DCD data types are: *string, number, date* and *time*.

of which classes a property may be used. It is also possible to say that a property must have a value that is an instance of a certain class. Often no such constraints are given, because one wants the vocabulary to be used on all kinds of resources. One example of such a vocabulary is the Dublin Core schema. In RDF Schema there is (currently) no way to express that a certain class *must* or *should* be described by a certain property. Given the decentralized nature of the web, it wasn't considered feasible to restrict a class to be described in a certain way or with certain properties. The RDF Schema doesn't contain any data types but may be extended to support them.

2. In *DCD* the focus is on classes. We can say that a class *must* be described by certain properties (elements in the DCD terminology). We can even put a constraint *Content="Closed"* which means that the class can *only* be described by the supplied properties. We can also put constraints on the values of properties. DCD also defines data types.

So which schema definition language should we use, RDF Schema or DCD? In the math exercise case study we chose RDF Schema. The reasons for this can be illustrated by an example:

With RDF Schema we can subclass our "MathBook" class from a well known "Book" class from some widely used schema. Let's say for example that the online bookstore Amazon provides such a schema. By subclassing the Book class from the Amazon schema we declare that we are selling a book that complies with Amazon's definition of what a book is. This does not mean that we have to use properties from the Amazon schema to describe our book (or math book) resources. Instead we may want to use the Dublin Core schema and a few properties that we declare in our own schema (edu). It is impossible for the creator of the Amazon schema to perceive every possible property that users on the web will want to describe the book class with.

The next section (4.5) describes a scenario where DCD is useful, i.e. when we use RDF to transport data between servers. Then we would like to validate the RDF descriptions so that they contain all data the receiving server requires and that the data are of the required data types.

DCD is likely to become widely used if IBM and Microsoft continue to back them. DCD is already implemented in Microsoft Internet Explorer 5.

4.5. RDF AS MEANS OF TRANSPORT

As we add more resources to the edu:content-bag of our math exercise, such as an instance of the class Text, an instance of Image and an instance of Solution we may start to realize that this is no longer a description of a math exercise defined in math_exercise. The RDF document *is* the math exercise! This highlights the fact that RDF is a data model general enough to be used for other things than just describing resources. We can use RDF as the *transport* format of math exercises.

As the RDF document is in fact our math exercise, we can remove the "about"-attribute from the rdf:Description-element and add an ID that identifies this new math exercise resource.

RDF can be used to transport math exercises or books containing math exercises between different servers. A math exercise broker could import math exercises expressed in RDF from many different servers. Of course, the market for math exercises may not be big enough to make this commercially viable.

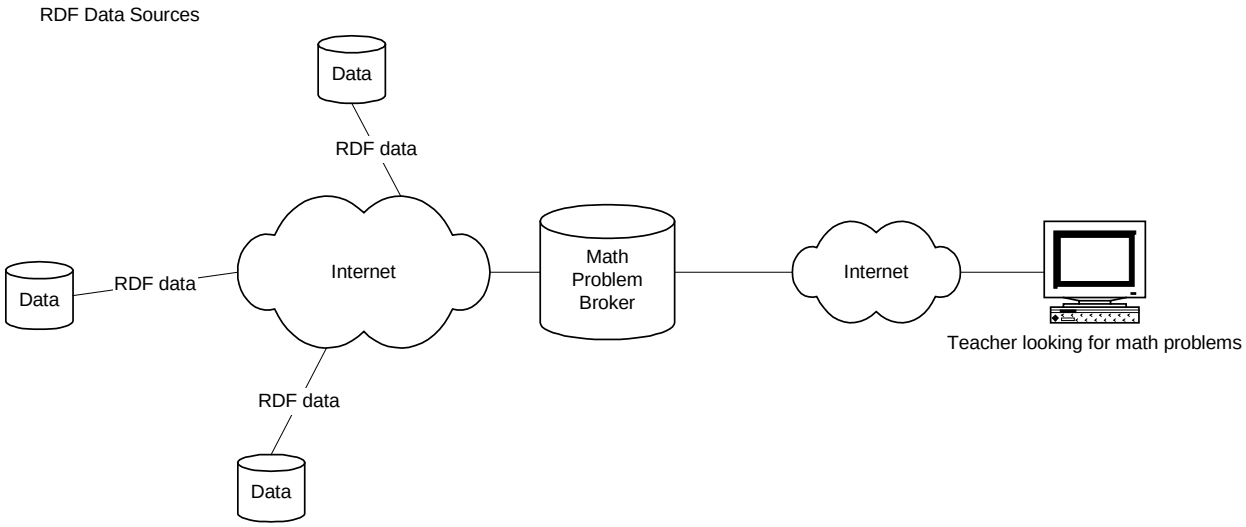


Figure 19: The RDF Data sources are providers of math exercises. They are capable of delivering the math exercises in RDF according to some agreed upon schema. The Math Exercise Broker reads the RDF data from the different sources and provides an interface to the Teacher looking from math exercises.

Figure 19 highlights why RDF is interesting to a system like the Multimedia Broker [MMBROK]. The Multimedia broker would benefit from being able to present its products in a format that is recognized by external systems (for example other Multimedia Brokers).

4.6. SUMMARY

We have now completed the RDF math exercise example. We have modeled a math exercise using the RDF data model and provided a schema using the RDF Schema specification. We have showed that RDF is capable of more than just describing resources. RDF can be used for storage, transport and retrieval of any data.

When modeling the math exercise we used the dc:relation property to indicate relations between different resources. The math exercise is a part of a larger context, the math book. The exercise in turn consists of a number of resources such as texts and images.

In constructing a schema for the math exercise we briefly discussed the benefits of RDF Schema over other schema definition models such as DCD. We also noted a problem with allowing only one rdfs:range property in the schema definition of a property.

We conclude this chapter with examples of RDF applications. An RDF application is a vocabulary and additional semantics that form a layer on top of RDF. Examples of RDF applications are:

1. PICS – Platform for Internet Content Selection [PICS] which we discussed in chapter 2.

2. P3P - Platform for Privacy Preferences [P3P]. The P3P specification will enable Web sites to express their privacy practices and users to exercise preferences over those practices. P3P products will allow users to be informed of site practices, to delegate decisions to their computer when possible, and allow users to tailor their relationship to specific sites.

3. DSig – Digital Signature Initiative [DSIG]. The W3C Digital Signature Working Group developed a standard format for making digitally-signed, machine-readable assertions about a particular information resource. More generally, it is the goal of the DSig project to provide a mechanism to make the statement: signer believes statement about information resource.

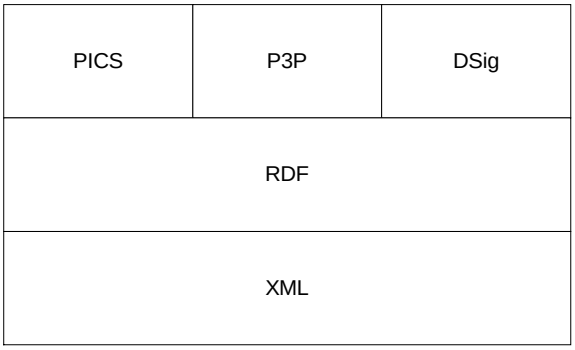


Figure 20: RDF applications are a layer on top of RDF

5. DESIGN AND IMPLEMENTATION

In this section we will discuss how RDF was implemented in our project. Let's recapture the objective of the implementation part of this project from the introduction in section 1:

> "We want to describe the products and services so outside metadata consumers can discover the products and relationships between products. We also want to be able to describe and export products and services to other Multimedia Broker systems or similar systems from other vendors"

We have already discussed the how the products in the Multimedia Broker can be modeled with the RDF data model. We have also defined a simple schema suitable for the educational product case in the Multimedia Broker. A complete description of the math exercise model and schema can be found in the next section.

RDF was implemented both on the Multimedia Broker and in a simple client, which we call the *RdfClient*. Details about the server side implementation, i.e. the implementation on the Multimedia Broker can be found in section 5.2. The RdfClient is an RDF parser capable of showing RDF descriptions of all kinds of resources, i.e. the client is not tied to any particular application or schema. A user can edit the RDF descriptions using a simple graphical user interface. Details about the client side implementation, the RdfClient, can be found in section 5.3. These sections assume some knowledge about web servers and programming techniques.

5.1. RDF DATA MODEL AND SCHEMA

The interested reader should have a pretty good understanding of RDF after reading the previous chapters and is probably eager to see the RDF data model of math exercise instances actually generated by the Multimedia Broker. Let's look at a math exercise instance and the edu schema implementation before we dive into the server-side implementation in section 5.2:

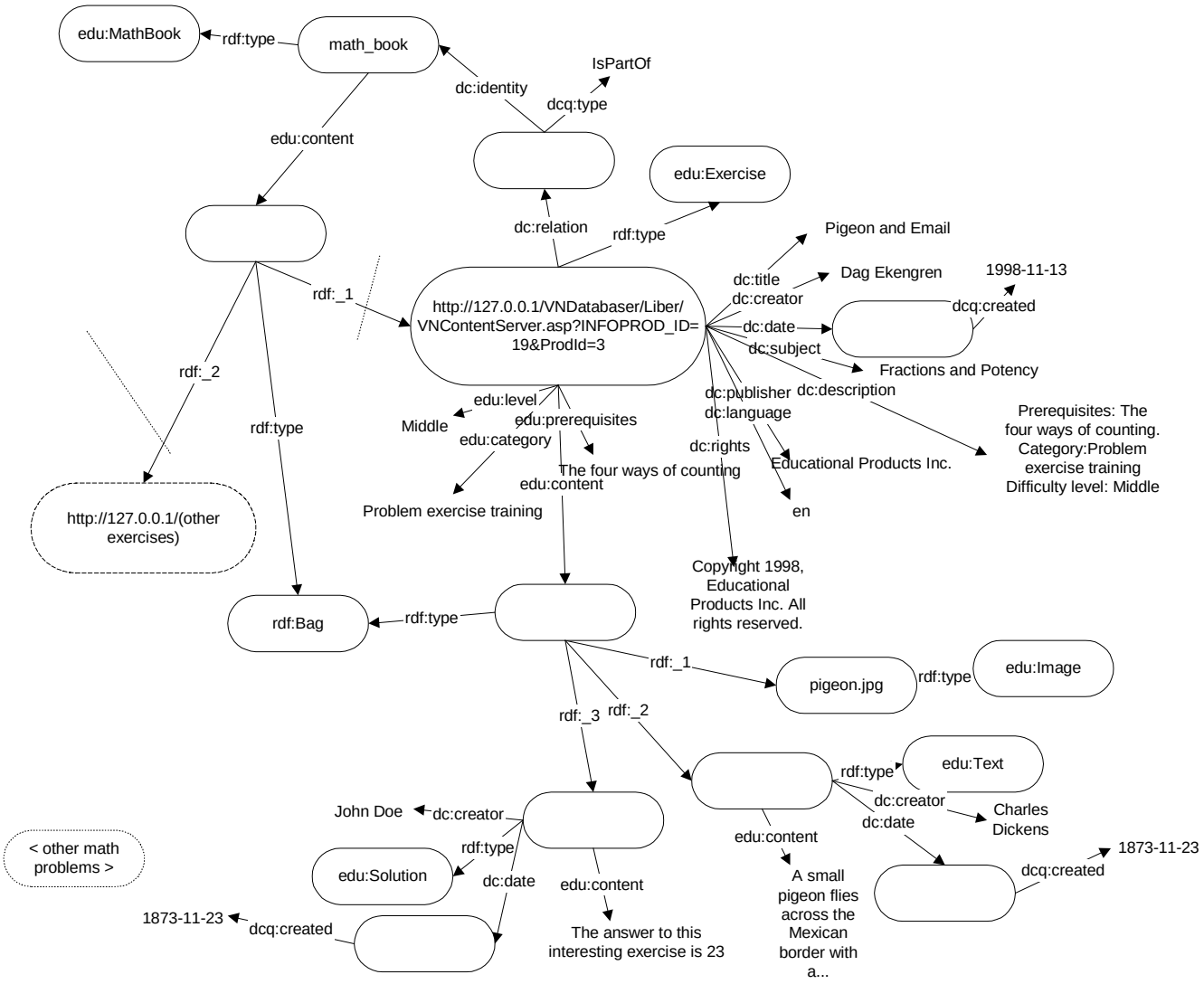


Figure 21: A math exercise instance generated by the Multimedia Broker.

The math exercise uses the Dublin Core RDF schema and the following *edu* RDF schema:

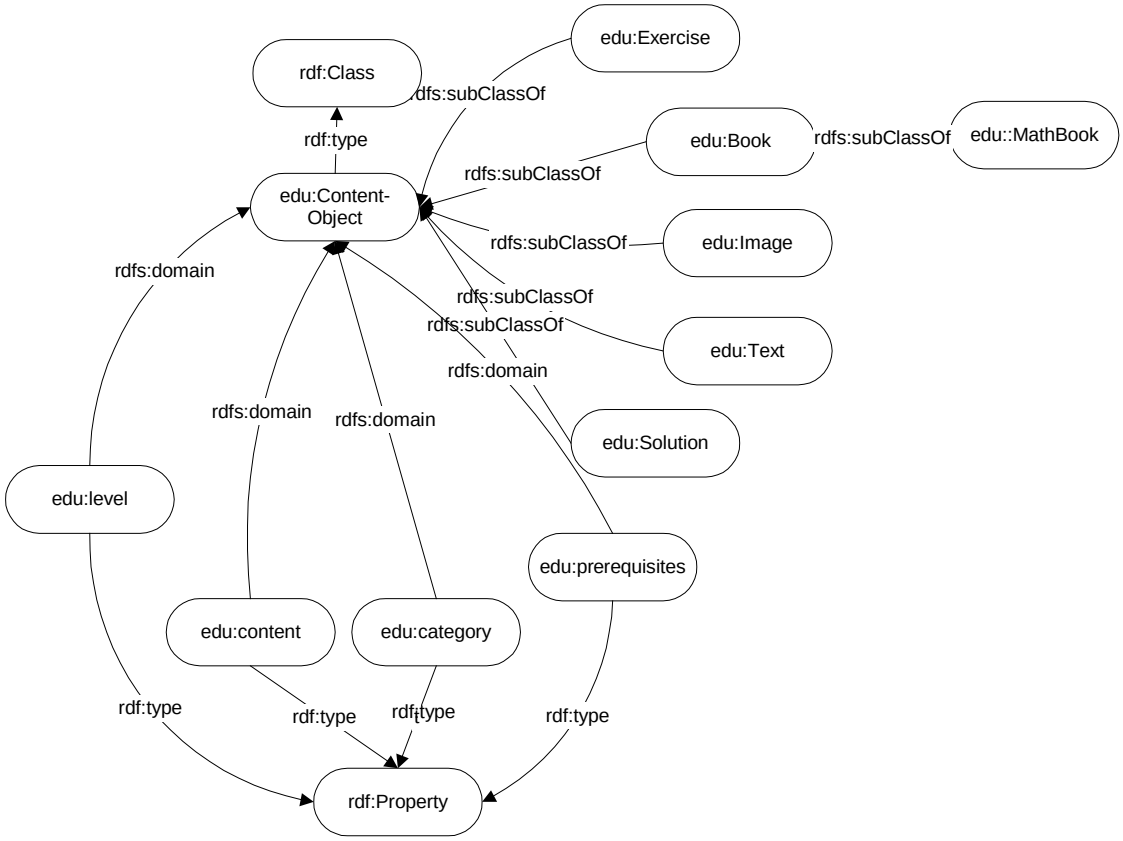


Figure 22: RDF schema for the edu namespace

5.2. SERVER SIDE – MULTIMEDIA BROKER

The server-side implementation was simplified by the fact that the Multimedia Broker is a flexible tool when it comes to choosing how products are presented. The system separates content from presentation. This means that the same content may be displayed in several different ways, depending on whom the user is.

In the broker system, there are *Content Products* and *Information Products*. The Content Products are products in a traditional way. They can be books, math exercises, or things like cars, ships or space stations. The Content Products are defined in the Multimedia Broker metadata database. The products can be presented in many different ways and the combination of 1) a product or a collection of products and 2) the presentation of that product or collection of products makes an *Information Product*. The presentation of a product is determined by a *Presentation Model*. An information product can be the HTML presentation of a book. Another Information Product would be the RDF description of the same book. A collection of products, for example the result of a database query, with a presentation model is another Information Product.

The Content Products and Information Products are defined in a metadata database.

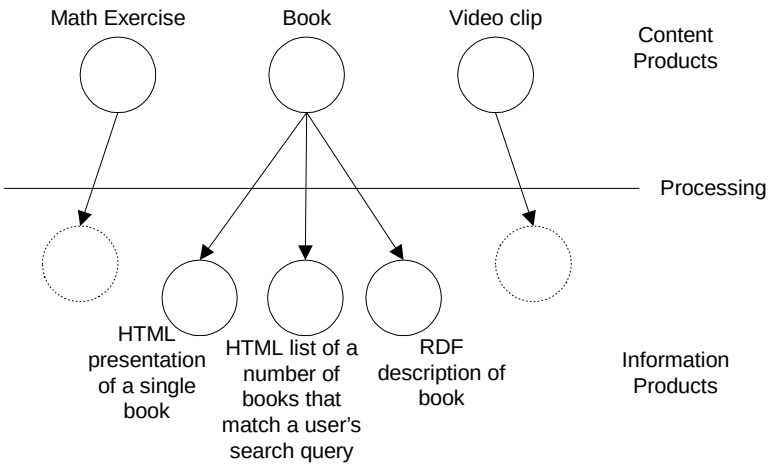


Figure 23: Schematic drawing showing the Multimedia Brokers product structure. Each of the Content Products above can be presented in any number of ways, called Information Products

The approach in this project is to implement RDF in the Multimedia Broker server by creating an Information Product that has a Presentation Model that creates RDF output. The generation of RDF is then completely integrated with the rest of the system: RDF generation doesn't differ from generating HTML, VRML or any other output format. That's one of the advantages the Broker has over a traditional web server. Fortunately it simplifies server-side implementation in this project.

The process from client request to a returned RDF document is shown in the following (much simplified) flow chart:

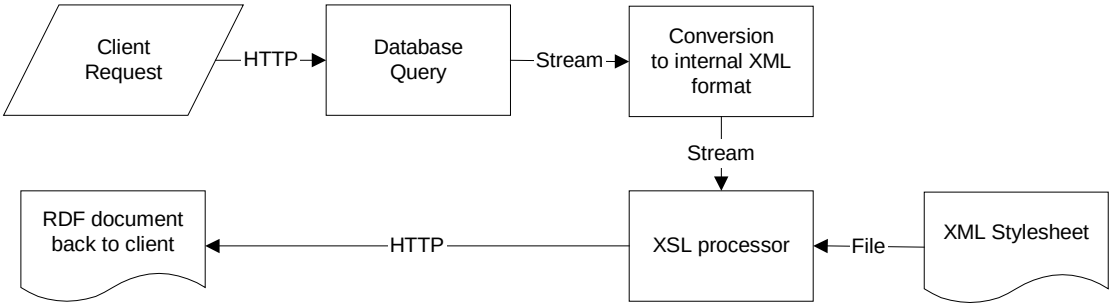


Figure 24: The process from client request to an RDF document describing a product.

The steps of implementing RDF on the Broker are:

Create a database *query* that returns all the data needed for the product being described. The Multimedia Broker uses CQL (Conceptual Query Language) which makes creating the query particularly straightforward.

Create an Information Product by associating the query with a Presentation Model. The Presentation Model is implemented with an XML Stylesheet (XSL). Readers that aren't familiar with XSL may want to read the section on XSL in Appendix A.

5.2.1. THE XSL PROCESSOR

The Multimedia Broker uses an internal XML format to exchange information between the server's different components. The result of our database query is wrapped up in an XML document in a format internal to the Broker. If we made the proper query, this XML document contains all the information needed for the RDF description of the product.

We simply feed that XML document to the XSL processor together with a stylesheet, that transforms the XML document into an RDF document[11]. The XML document's attributes, which are specific to the Multimedia Broker, are mapped to well-known schemas (most notably the Dublin Core schema). The XSL processor lets us package and rearrange the information in a way that creates output that is valid RDF descriptions of our products.

A separate XSL stylesheet for conversion to RDF has to be made for each product in the system. So, in our educational products case, we made one stylesheet for the math exercises and another for the math books. The schema is in effect hardwired into the XSL stylesheet scripts. An area of future improvement is to allow the schemas to be defined in the metadata database. The XSL stylesheet would then be generated from the metadata database automatically.

Here is part of an XML to RDF stylesheet for Microsoft's XSL processor:

```
<xsl>

<rule>
 <root/>
 <![CDATA[
 hello
 ]]>
 <?xml version="1.0"?>
 <rdf>
 <children/>
 </rdf>
</rule>

<rule>
 <target-element type="CATTRIBUTE">
  <attribute name="ID" value="190"/>
 </target-element>
 <select-elements>
  <target-element type="A"/>
 </select-elements>
</rule>

<rule>
 <target-element type="A"/>
 <![CDATA[

 <edu:Exercise ID="]]><eval>getAttribute('HREF')</eval><![CDATA[">
 ]]>
</rule>

<rule>
 <target-element type="ROW"/>

 <![CDATA[
 <!-- Instance of class edu:Exercise -->]]>

 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="190"/>
  </target-element>
 </select-elements>

 <![CDATA[ <dc:Title>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="199"/>
  </target-element>
```

[11] Please note that RDF documents are indeed XML documents, with the added requirement that they follow the W3C RDF Model and Syntax specification.

```
 </select-elements>
 <![CDATA[</dc:Title>
 ]]>

 <![CDATA[ <dc:Creator>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="106"/>
  </target-element>
 </select-elements>
 <![CDATA[<"</dc:Creator>
 ]]>

 <![CDATA[</edu:Exercise>
 ]]>
</rule>


<rule>
 <target-element type="RESULT"/>
 <children/>
</rule>

<rule>
 <targem-element type="QUERY"/>
 <children/>
</rule>

</xsl>
```

A more complete XML to RDF stylesheet and an example of the internal XML format in the Multimedia Broker can be found in Appendix C.

5.2.2. OTHER SERVER IMPLEMENTATIONS

RDF descriptions can be created on the server quite easily even if you don't have access to tools as powerful as the Multimedia Broker. This section gives examples of such implementations.

Server Scripting

One way to generate RDF is to use server-side scripting. Server-side scripting is available for most web servers, including the widely used Microsoft's IIS4[12] and Apache[13]. The scripts are mostly used to generate HTML today. The scripts can however generate any text-based format.

We will now show a script that opens a database connection and packages to information directly into RDF statements.

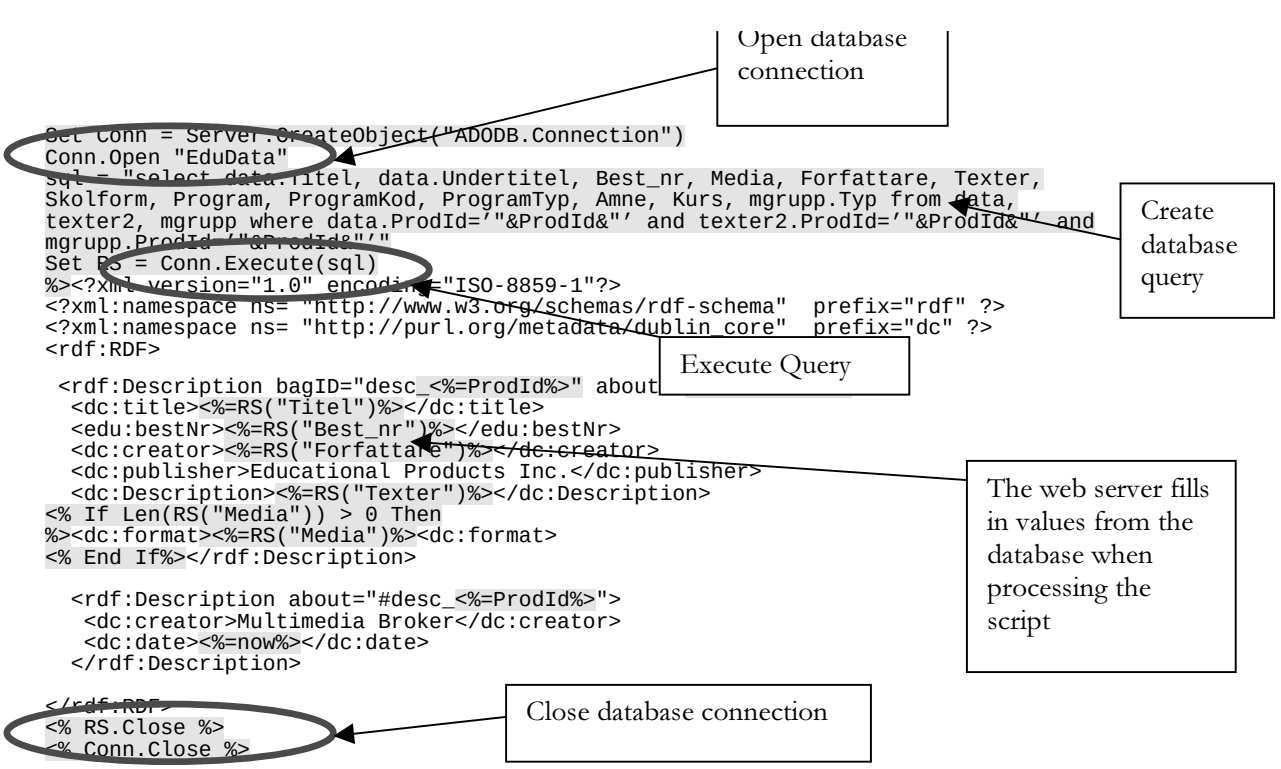
The sample script is written in VBScript[14]. The shaded parts are script code executed by the web server. The white parts are returned to the client as text.

```
<%@LANGUAGE="VBSCRIPT"%>
<% Option Explicit %>
<% ' This script generates RDF metadata %>
<%
Dim ProdId
Dim Conn
Dim RS
Dim sql
Set ProdId = Request.QueryString("ProdId")
```

[12] IIS4 – Microsoft Internet Information Server version 4. Visit http://www.microsoft.com for details.

[13] Apache – available to several platforms, including Linux and Windows NT. Visit http://www.apache.org for details.

[14] VBScript – Visual Basic Scripting Edition, a Microsoft server scripting technology. Similar scripts can be written in JavaScript and other scripting languages.



This simple script doesn't handle things like missing values in the database. The script is run by the web server when a web browser accesses it via a URL. The particular instance is identified with ProdId, a product id.

```
http://www.ekengren.com/rdf/testscript.asp?ProdId=T10023
```

Here is an example of an instance generated by the script:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml:namespace ns= "http://www.w3.org/schemas/rdf-schema"  prefix="rdf" ?>
<?xml:namespace ns= "http://purl.org/metadata/dublin_core"  prefix="dc" ?>
<rdf:RDF>
 <rdf:Description bagID="desc_T10023" about="T10023">
  <dc:title>Macbeth</dc:title>
  <edu:bestNr>231231</edu:bestNr>
  <dc:creator>William Shakespeare</dc:creator>
  <dc:publisher>Educational Products Inc.</dc:publisher>
  <dc:Description>A simplified version of Macbeth which includes all speeches but
the longer ones have been abridged</dc:Description>
  <dc:format>paperback<dc:format>
</rdf:Description>

 <rdf:Description about="#desc_T10023">
  <dc:creator>Multimedia Broker</dc:creator>
  <dc:date>1998-12-08</dc:date>
 </rdf:Description>
</rdf:RDF>
```

As you can see, the script has generated an RDF description of a book with the title "Macbeth". The script also generates a description for the description.

5.3. CLIENT SIDE – THE RDF CLIENT

As part of the project we implemented a client application for reading and parsing RDF documents. The RdfClient is very useful for prototyping new schemas and for checking syntax in RDF documents. It is also useful as a learning tool for the RDF data model.

The requirements are that the RdfClient:

- Is generic enough to show RDF descriptions of all kinds of resources and has an easy to use graphical user interface to browse the RDF document

- Shows the RDF document in a way that closely resembles the RDF data model
- Allows the user to search the RDF document
- Allows the user to edit the RDF document
- Provides some RDF schema support
- Runs in a Browser on a wide range of hardware/software configurations

To meet the last design goal we decided to write the RdfClient in Java 2. This allows the client to run in browsers or with the Sun Java plug-in. The graphical user interface of the client uses the Java Foundation Classes from Sun, which are included in the Java 2 platform[15]. Specifically we wanted to use the "Tree" and "Table" graphical components and the menu components, as shown in the following section. RdfClient runs both as an applet and as a standalone application.

The client can be run from http://www.ekengren.com/rdf/client.html. Follow the guided tour in the next section by pointing a web browser to this URL.

5.3.1. RDFCLIENT FUNCTIONALITY – A GUIDED TOUR

Loading RDF documents

Let's start the guided tour by loading an RDF document. RDF documents can be loaded into the client either from a local file or via HTTP from an URL. A local file is loaded by selecting *Open* in the *File menu*:

[15] The Java 2 platform is also known as JDK 1.2. Sun changed the name when the product went from beta version to release.

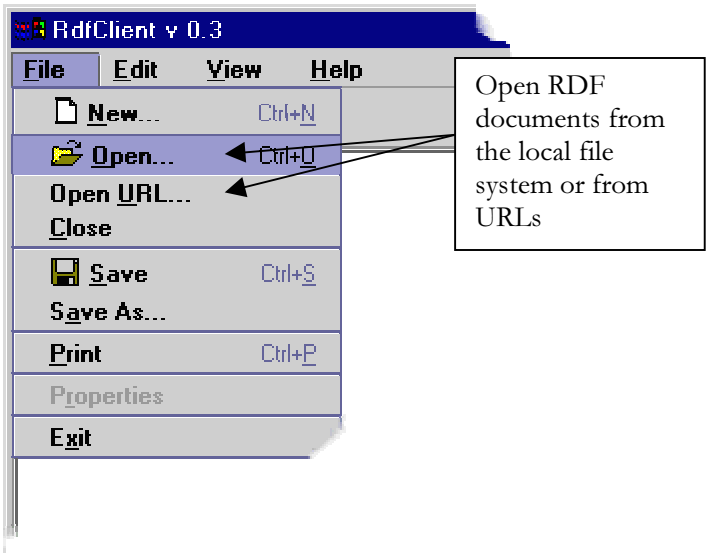


Figure 25: Open File in the RdfClient

Other features in the *File Menu* are *Save* and *Print*. Print enables the user to print out the current RDF document serialized to XML to standard output. Java Applets are not permitted to access local files. If you load the RdfClient as a Java Applet, select *Open URL* and type http://www.ekengren.com/rdf/math_exercise.xml.

Browsing an RDF document

One of the significant features in RdfClient is being able to browse any RDF document to demonstrate various RDF features. We load our famous math exercise instance into the RdfClient. This is how it turns out:

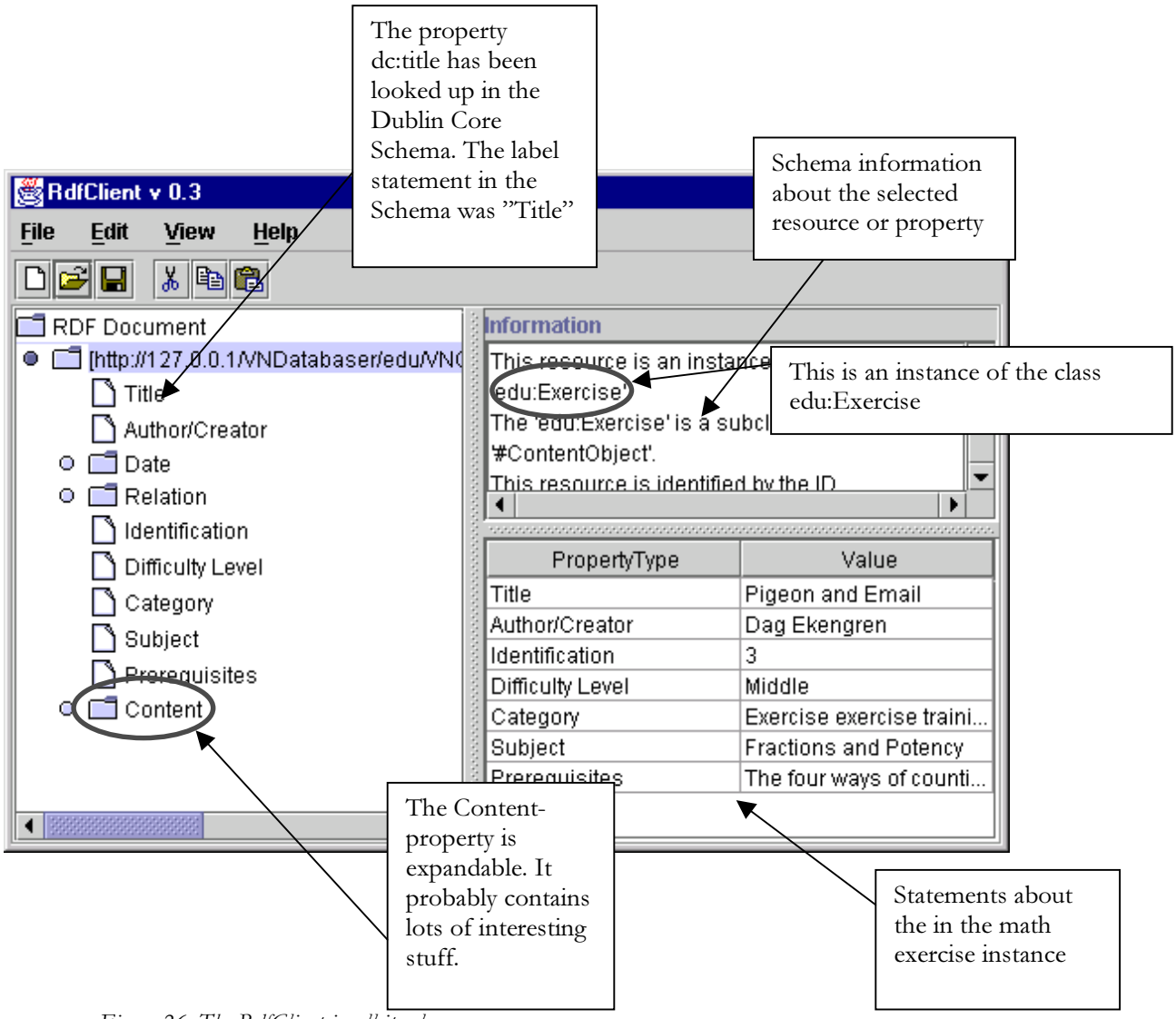


Figure 26: The RdfClient in all its glory.

The left pane shows the RDF document. It is displayed as a *TreeView*. The children nodes can be expanded and collapsed by clicking on the *bullets* next to the nodes. The *TableView* to the right shows the values of the properties for this particular instance. The instance can be edited by the user and the changes are propagated to the internal RDF data model.

The schema information window tells us that we are viewing an instance of the class *edu:Exercise*, which is a subclass of the class *ContentObject*. The math exercise schema has been described in section 4.3. The property names are looked up in the corresponding RDF Schemas. That's why we see Title instead of *dc:title* in the property column of the Table in Figure 26.

By clicking on the *Content* property node in the *TreeView* we can expand it to reveal its contents:

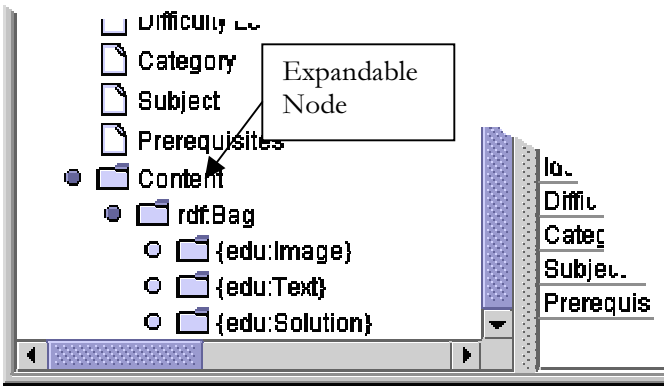


Figure 27: The value of the edu:content property is an rdf:Bag, which contains instances of the classes edu:Image, edu:Text and edu:Solution

Let's click on the bullet next to the node of the class *edu:Text* to reveal the statements inside. We can now see that the *edu:Text*-instance included in this math exercise has a description with three statements. One of the statements uses the *edu:content*-property. The value of this property is the actual exercise text. We have also given the exercise text *dc:creator* and *dc:date* statements which may or may not match the corresponding statements about the math exercise. This demonstrates the various levels of granularity we can achieve in the descriptions.

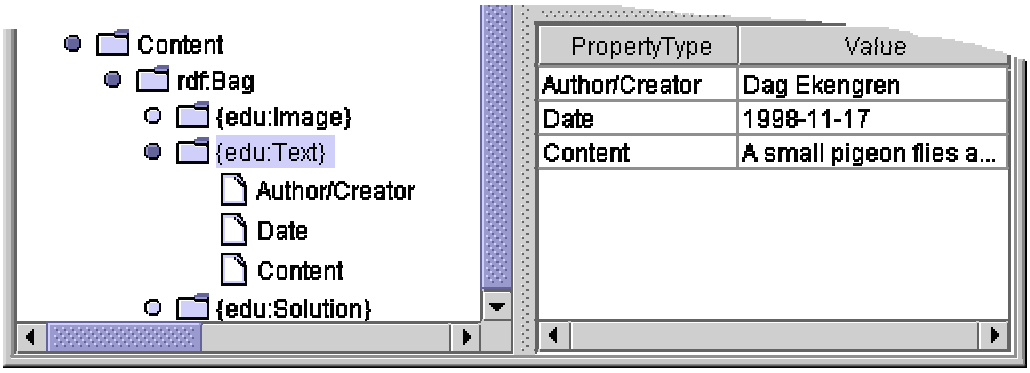


Figure 28: The Bag which was the value of the edu:content property contains three objects. We have clicked on the object of class edu:Text. This text object could be a part of other math exercises as well.

Editing an RDF document

The RDF client gives the user the ability to edit any RDF document. This functionality can be used as a basis for creating an RDF metadata authoring tool. The newly edited value is immediately stored in the internal RDF data model.

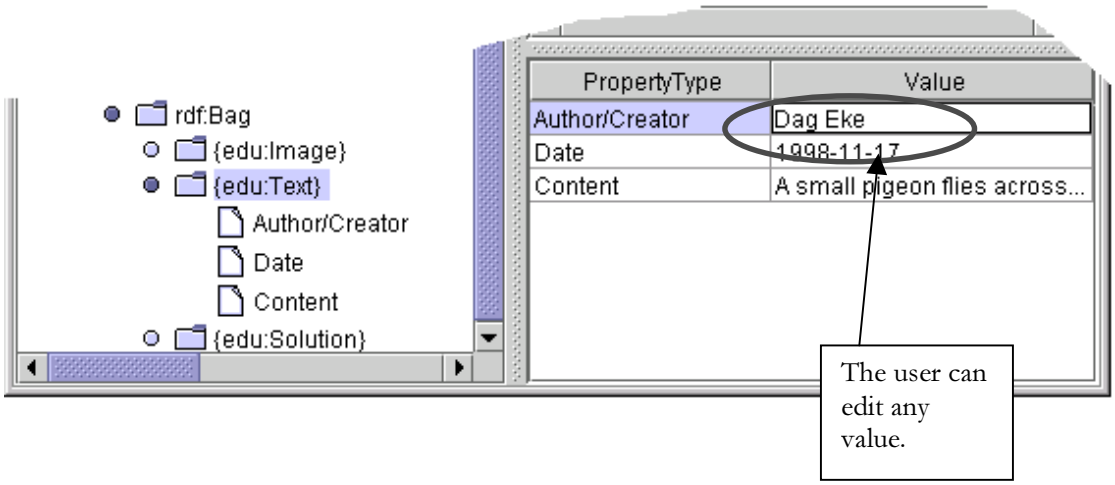


Figure 29: The dc:creator property has been selected for editing. The user finishes editing by pressing return. The change is immediately propagated to the internal RDF data model.

If we change the *dc:creator* to "Dag Ekenben", the change can be checked in RDF data model by viewing the source as shown in the next paragraph.

View Source

In web browsers you can select the menu item *View Source* to see the HTML-markup for the currently loaded web page. In RdfClient you can select *Source* from the *View*-menu. This shows the XML serialization of the currently loaded RDF data model in a separate window. You can also right-click on any node and select *View Source from here* in the popup menu that appears:

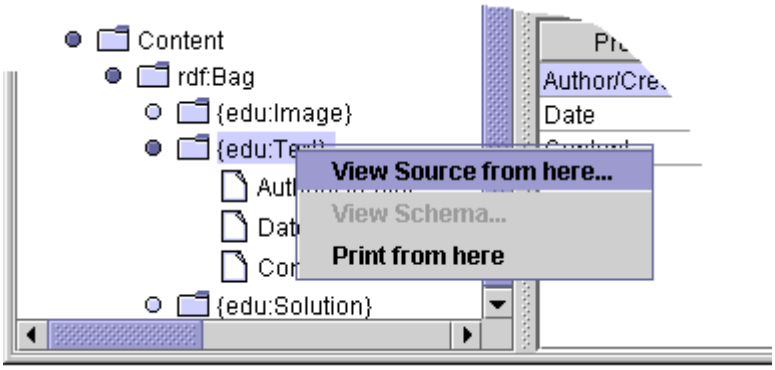


Figure 30: When the user right-clicks on a node a popup window appears. Selecting "View Source from here" opens a window with the XML serialization of the current node and all its children.

In our example, we changed the value of the *dc:creator* property in our instance of *edu:Text* to "Dag Ekenben". As we might expect, the change is properly propagated to the RDF data model:

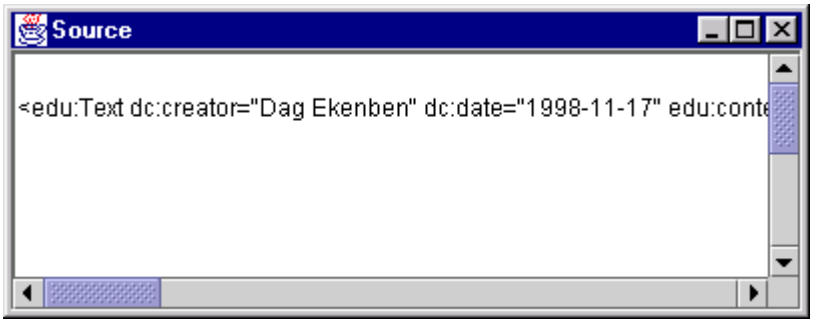


Figure 31: The value of the dc:creator property has been changed to "Dag Ekenben". Cool stuff.

Schema Support

The RdfClient *Information* window shows the information from the RDF schemas associated with each property or class. This information is provided to help the user interpret or edit the instance values:

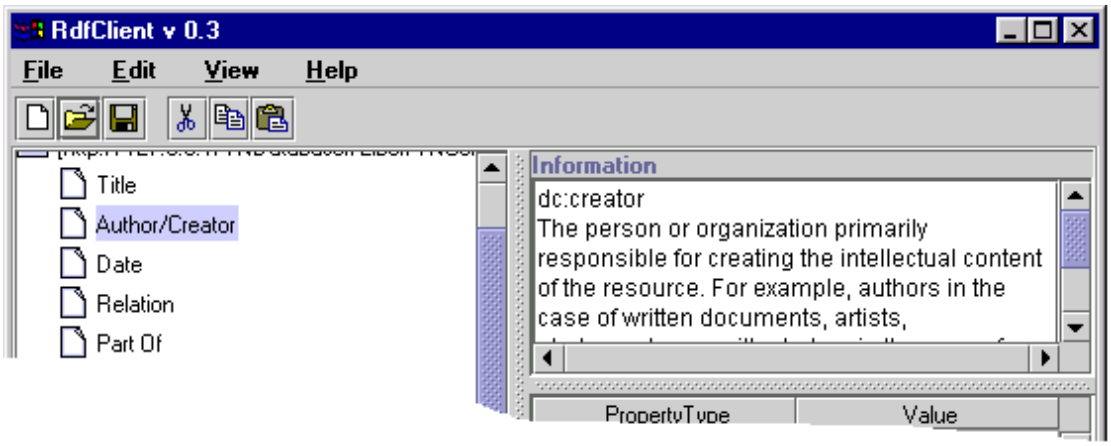


Figure 32: The user has selected the dc:creator property. The "Information" window shows the definition of dc:creator from the Dublin Core schema. The full text can be viewed by using the scrollbar.

5.3.2. INSIDE RDFCLIENT

First a short summary for those who already know a lot about XML and Java. In the following sections we will describe the process in more detail for the rest of us.

Summary: When *RdfClient* reads an RDF document it is first processed as an XML document. The IBM XMLforJava [XML4J] parser does this and creates a DOM[16] tree. This tree is processed by the RDF parser, also from IBM [RDF4XML]. The RDF parser builds the RDF data model in memory with Java objects. The RDF parsers data objects are wrapped up in classes that comply

[16] DOM – Document Object Model. A W3C API for accessing data from a tree data structure [DOM].

with the JFC[17] MutableTreeNode model. This means that the RDF data model is directly browsable and modifiable by the JFC Tree GUI component.

Let's take a simple example of an RDF document and see how it is handled step by step by *RdfClient*. Figure 35 summarizes this process. Readers unfamiliar with XML may want to read Appendix A.

1. *The XML parser reads the RDF document.*

In this first step the RDF document is read in the same way the XML parser would read any XML document. This RDF document is yet another flavor of our now famous math exercise:

```
<?xml version="1.0"?>
<rdf:RDF    xmlns:rdf="http://www.w3.org/rdf-syntax"
            xmlns:edu="http://www.ekengren.com/rdf/schemas/eduschema" >
            xmlns:dc="http://purl.org/dc/elements/1.0"
    <edu:Exercise about="http://www.ekengren.com/rdf/math_exercise">
        <dc:title>Pigeon and Email</dc:title>
        <edu:content>
            <rdf:Bag>
                <rdf:li>
                    <edu:Text edu:content="A small pigeon flies across the mexican
border with a floppydisk" dc:creator="Dag Ekengren"/>
                </rdf:li>
            </rdf:Bag>
        </edu:content>
    </edu:Exercise>
</rdf:RDF>
```

The XML parser reads this in the same way it reads any XML document. It doesn't recognize any RDF-specific tags. The XML parser builds a data structure while parsing the document. This strictly hierarchical data structure is a DOM (Document Object Model) tree:

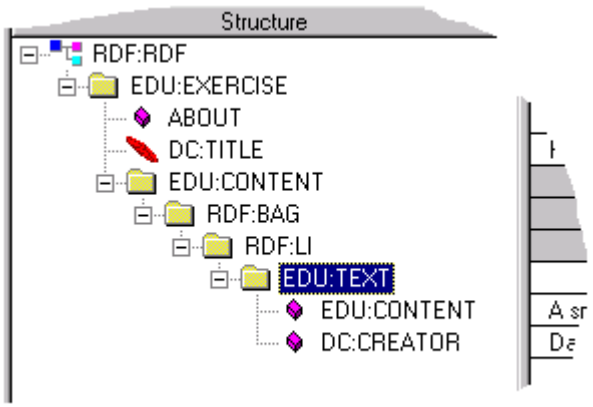


Figure 33: This the DOM of our RDF document. The example document was read by Microsoft's XML notepad. The XML notepad doesn't know about RDF and doesn't treat the RDF document any differently from any other XML document.

This tree can be navigated and manipulated by a standard API[18] proposed by the W3C [DOM]. This navigation and manipulation is the RDF parser's job.

[17] JFC – Java Foundation Classes. GUI component classes downloadable from Sun Microsystem's web server http://www.sun.com. JFC is included in Java SDK 1.2 from Sun.

[18] API – Application Programmer's Interface.

2. The RDF parser traverses the DOM tree

The RDF Parser has knowledge about the RDF-specific nodes and their semantics. It builds the RDF data model in memory by traversing and parsing the DOM tree. The RDFforJava parser from IBM builds this data model with Java objects of various classes.

The class hierarchy used in the RDF parser is described in Appendix D. For our sample document the following RDF data model is built:

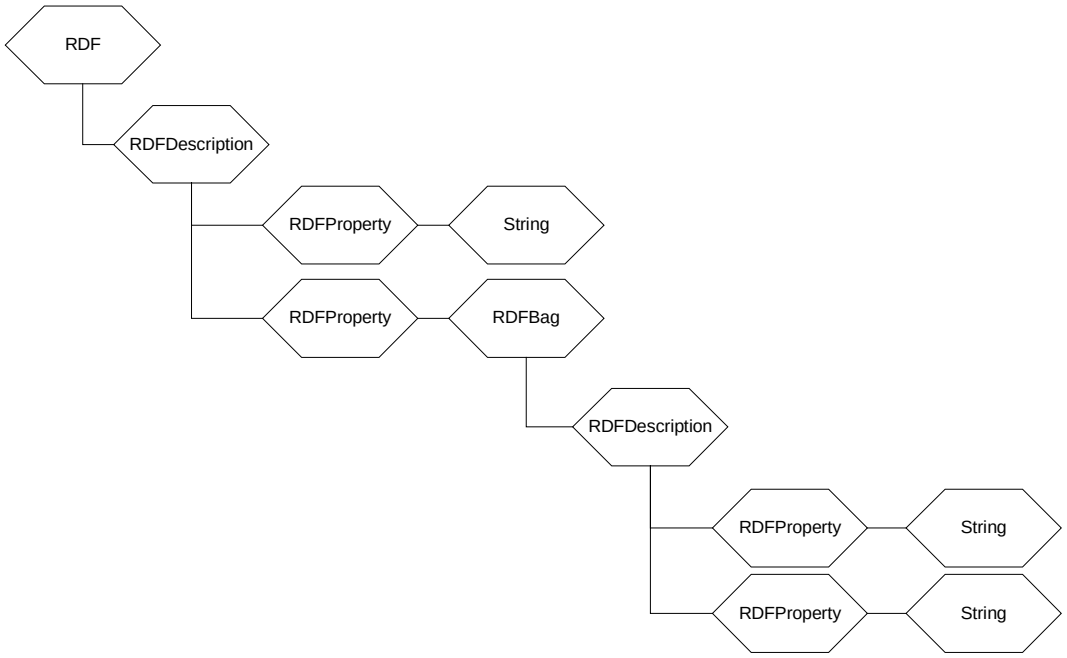


Figure 34: The RDF data model built from our sample RDF document. The boxes are Java object instances of the named classes. The RDF document's root node is of class RDF. It contains one RDFDescription instance (describing math_exercise). The RDFDescription instance is parent for two RDFProperty instances. The first instance (corresponding to dc:title) has a string value. The other property instance (edu:content) has an RDF bag as value.

One of the design goals of *RdfClient* was to provide a Graphical User Interface that lets user browse and edit RDF documents. The graphical representation of the RDF data model can be done in many ways. The representation closest to the data model itself would be a GUI with nodes and arcs. The disadvantage of this type of GUI is that layout and navigation can become quite complex.

A tree view was chosen for the following reasons:

- Ease of implementation.
- Most users are familiar with how to navigate the tree from previous use of tools such as the Microsoft Windows Explorer.
- Ability to use standard Java tree view classes.

3. *The JFC MutableTreeNode Classes wrap the RDF data model classes*

To make the RDF data model built by the RDF parser editable by the JFC Tree component, each object in the RDF data model is wrapped up in a Java object of a class that implements the *MutableTreeNode* interface. The RdfClient software accomplishes this task.

The *RdfClient* has an equivalent MutableTreeNode class for each of the RDF data model classes presented in Appendix D. The user can manipulate each MutableTreeNode by using the GUI. The changes are actually made directly to the RDF data model nodes acting as MutableTreeNodes. There is no data redundancy and therefor no risk for inconsistencies.

The figure below sums up the data flow in *RdfClient* when loading an RDF document. When a document is edited and saved, the data flows in the opposite direction.

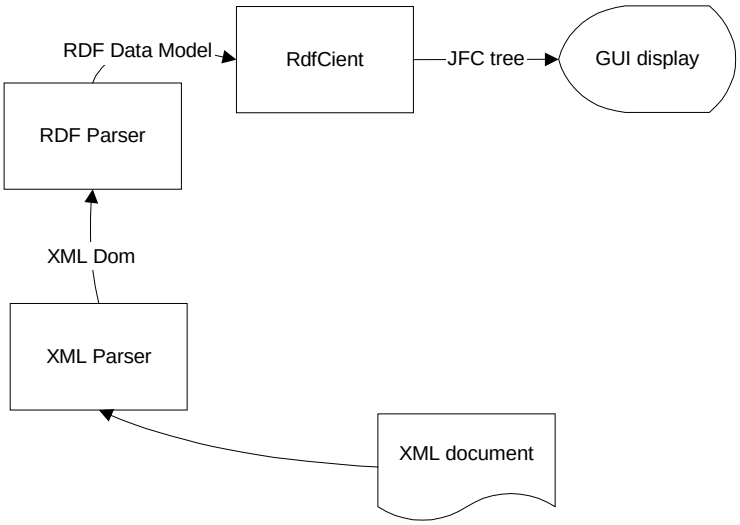


Figure 35: The internal processing done in the RdfClient

The sample document looks like this in the TreeView of *RdfClient:*

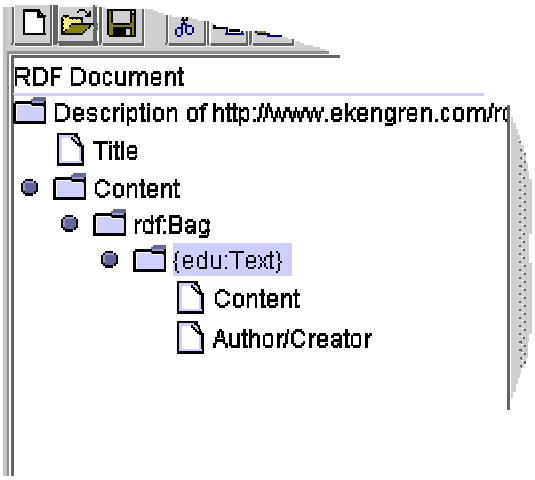


Figure 36: Our sample RDF document has been wrapped in MutableTreeNodes and then handed to the JFC Tree Component.

4. *Descriptions are wrapped in JFC TableModel*

When the user clicks on a description object in the Tree component, the RDF description is wrapped in a Java object that implements the TableModel interface. This allows the RDF description to be directly edited by the JFC Table Component. Please refer to Figure 29 where the TableView is shown "in action".

5. *RDF Schemas*

One of the design goals of *RdfClient* was to provide some support for the RDF Schema proposition [RDFSCH]. When *RdfClient* loads an RDF document it looks at the namespaces declared in the beginning of the document. Each schema is loaded from the URLs provided in the declarations. The XML and RDF parser parses the schemas (which are themselves RDF documents) independently from each other and from the main RDF document. They are used for schema lookups only and are therefor not shown to the user in the Tree component.

When the user clicks on a node in the TreeView (which represents the RDF document and not the associated schemas), the *RdfClient* searches the schema associated with that node. It displays any schema information it can find about the node in the *Information* frame. The *RdfClient* currently recognizes the following parts of RDF Schema [RDFSCH]:

- The *Class* of a resource
- If that class is a *Subclass* of another class.
- *Comments* about properties and classes
- *Labels* for properties
- The *Range* and *Domain* for properties.

The range of a property is the allowed class of the RDF value of that property. The domain is the class of the objects that the property can be applied to.

5.3.3. DIFFERENT VIEWS OF DATA

RdfClient presents the data in a way that is very close to the XML serialization of the RDF data model. The same RDF data can be displayed in any way the application designer wants. The data can also be presented in different ways and with different levels of detail depending on who the user is.

In our RDF example with math exercises we could present the teacher with a full view of the exercises and their solutions. The teacher picks the exercises he finds interesting and puts together a collection of exercises for the semester.

When the students are presented with the math exercises they see a friendly user interface with less information (and probably with the solutions removed).

5.3.4. SUMMARY

In this chapter we have showed the complete RDF data model and Schema for our case study math exercise. We have showed how RDF descriptions should be implemented on the Multimedia Broker. The implementation, which is very well integrated in the Multimedia Broker, is based on the Broker's internal XML document format in combination with XML Stylesheets. We have showed how RDF can be implemented on servers that support server-side scripting languages such as VBScript or JavaScript. This chapter introduced the RdfClient software, which can be used to browse RDF documents on any computer and operating system. In the design of the RdfClient we chose to present the RDF documents in a Tree GUI component, which proved to be a presentation that worked well with our math exercise case study.

The RdfClient, although a nice piece of software, may not be the typical RDF consumer in the future. It is very much tied to the RDF data model and its XML serialization. This is good for instructional purposes, but the end user may not be that interested in the internal workings of RDF.

If the RDF consumer were a machine, it would want another interface than the Graphical User Interface that *RdfClient* provides. Clearly there is a need for more specialized consumers of RDF data. In this section we will outline some ideas on RDF consumers.

6.1. INDEX SERVERS

One of the more obvious RDF consumers is providers of search engines like AltaVista Lycos, Excite and Webcrawler. Web servers inline RDF metadata information in HTML documents and this metadata is used by the search spiders when indexing the documents. This takes us back to the example discussed in the Introduction when we tried to find web documents *authored by* William Shakespeare. One of the major objects of describing resources is of course to enable *Resource Discovery*.

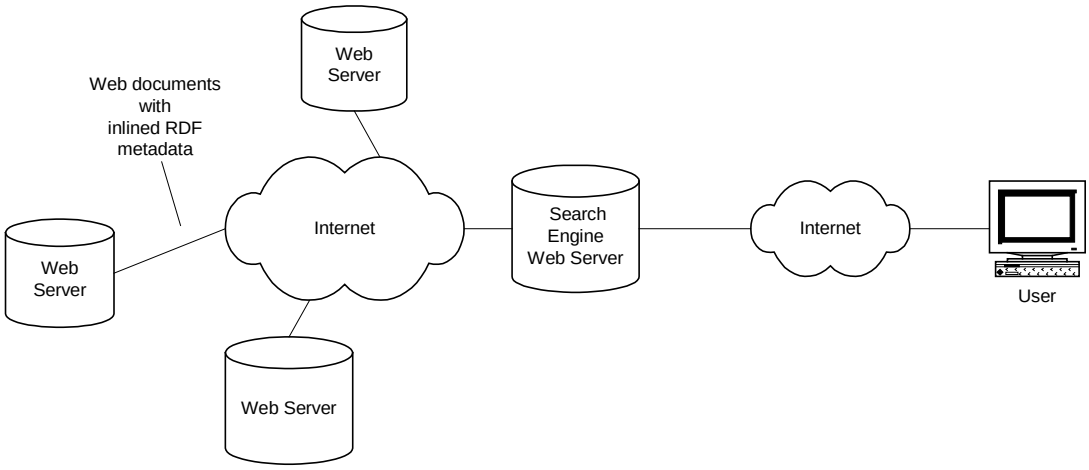


Figure 37: A bright future with RDF data sources

Metadata expressed in RDF or in other formats may be the next big thing for the web. The potential is enormous. Think about the web as a massive resource of information, with the data classified according to a few well-known schemas, where the users can find *exactly* the information they require. The web would no longer be the library "with all the books on the floor".

6.2. RDF AWARE WEB BROWSERS

Another RDF consumer could be the web browser. When visiting a web site the browser can download an RDF description of the site with any level of granularity. This description can be used to provide better ways to navigate the site from the browser. The browser can search the RDF description of the site locally and minimize the number of roundtrips required to the web site.

The Mozilla[19] web browser has RDF functionality built in. The currently recognized RDF format does not conform to recommendations from W3C. This is likely to change in later releases and in Netscape Communicator 5.0. RDF will be the data format of choice for all the Mozilla storage needs[MOZ1]. It will be used to store bookmarks, history lists, site maps, address books etc. This emphasizes the fact that RDF is a transportable data structure suitable for any data, not just metadata.

Currently, Netscape Communicator 4.5 sends back RDF data to Netscape's servers to track the users movements on the net. The information is used to provide users information in the "what's related" menu. The browser let's the user know about web sites that are related to the one currently viewed.

The Netscape "what's related" system is currently in full use and Netscape serves over 2 million requests in RDF every day.[20]

6.3. SOFTWARE AGENTS

A domain with great need for metadata descriptions is that of "Software Agents". A description of Software Agents and their use is definitely beyond the scope of this paper. Let me just point out that the agents need to be able to present themselves and their objectives. Other agents must understand this presentation, i.e. there is need for machine-readable metadata. This is a domain in which RDF may be useful. Software Agents from different vendors can agree upon using one or more well-known schemas they both know about.

SICS (Swedish Institute of Computer Science) has research on Software Agents. As part of my thesis I visited SICS and discussed metadata in this domain. SICS has developed a lisp-like metadata instance and schema language of their own. There is clearly a need to be able to present agents in a format that is more widely used, such as RDF[21].

6.4. EXISTING DOCUMENT ANALYSIS

Documents saved in popular office applications often contain some metadata. This metadata is stored inside the document files, which often are of some proprietary binary document format. Microsoft Office applications let the user specify properties such as title, subject, author, company and category. These properties can later be extracted and mapped to Dublin Core properties such as dc:title, dc:subject, dc:creator, dc:publisher and dc:category. This extraction can be done "on the fly" by a web server component when RDF metadata for the document is requested by a client.

This means that a large number of existing documents already have some metadata description. It is just a matter of making this metadata accessible from the web using RDF.

---

[19] Developers can download the source code from http://www.mozilla.org. The Mozilla source code will be used for Netscape Communicator 5.0.

[20] According to email, November 05, 1998, from Mr. R Guha., Netscape.

[21] As an exercise, I and Nicklas Finne (SICS) translated SICS agent metadata descriptions to RDF and RDF Schema. The results can be admired in Appendix B.

7. CONCLUSIONS

In section 1.1 we defined the objectives of this project: "We want to describe the products and services so outside metadata consumers can discover the products and relationships between products. We also want to be able to describe and export products and services to other Multimedia Broker systems or similar systems from other vendors". This led us to look into the RDF and RDF Schema specifications. We wanted to use a schema that was general enough to become widely used on the web. The Dublin Core is a schema that fits that requirement.

In our math exercise data model we added the property, *edu:content*, to indicate the relationships between the resources that constitute our math exercises. We wanted to restrict the range of the edu:content property to resources of class *edu:ContentObject* or simple string values, called literals. We realized that we had a problem here, because in RDF Schema strings are considered to be of class *rdfs:Literal* and the RDF Schema specification allows only *one* range for a property. This means that we would have to choose between having edu:ContentObject or rdfs:Literal as range for the edu:content property. We solved this by not restricting the range of edu:content property at all. A better solution perhaps would be to think of a string (or literal) as class-less. This would make strings possible values for all properties, regardless of their range. Currently, this is not allowed for in the RDF Schema specification.

After working with RDF and RDF Schema we have come to the conclusion that RDF is about expressing *relationships* between resources. It is not intended to express full database models. The heterogeneous and distributed nature of the web makes relationships between resources important. What is the context of this resource? What other resources are related to this resource, and in what ways? How can the related resources be retrieved? RDF's ability to express relationships between resources is more powerful than the hypertext linking facility in HTML, because in RDF we have the power to express *why* the resources are linked. This can be done in a way that is "machine-understandable".

The emphasis on relationships is reflected by the RDF Schema specification where focus is set on properties (which express relationships between resources) rather than on the resources (or classes of resources) themselves. In RDF Schema, the classes differ from classes in the traditional "object oriented" world where we define classes to have a certain set of properties. In RDF Schema any number of properties from any number of schemas can be used to describe a resource. Classes don't have a fixed set of properties. The class defines the resource's *relationship* with other resources, because of the subclassing facility. Subclassing let's us build taxonomies such us the ones used in libraries.

In modeling the math exercise example we realized that the RDF data model can be used to model and transport any kind of data not just metadata. In the Mozilla web browser RDF is used for all kinds of data, such as bookmarks, history lists and site maps.

When we implemented RDF on the Multimedia Broker we successfully used an XSL processor to convert the Broker's internal XML data structures to RDF. Increasingly, XML is used for transporting data between different components of web servers and this project has shown that such internal data can be mapped and converted to RDF using XML stylesheets.

ACKNOWLEDGEMENTS

This document is the report of a master's project for Department of Teleinformatics KTH. The project was held at SITI, the Swedish IT Institute, Stockholm.

REFERENCES

[DC4] Weibel, S., Iannella, R., Cathro, W., The 4th Dublin Core Metadata Workshop Report, Jun-1997, ISSN 1082-9873, http://www.dlib.org/dlib/june97/metadata/06weibel.html

[DC5] Weibel, S., Hakala, J., DC-5: The Helsinki Metadata Workshop, D-Lib Magazine, Feb-1998, ISSN 1082-9873, http://www.dlib.org/dlib/february98/02weibel.html

[DCD] Document Content Description for XML, 31-Jul-1998, Submission to the World Wide Web Consortium, http://www.w3.org/TR/1998/NOTE-dcd-19980731.html

[DCPURL] The Dublin Core Metadata Homepage, 1-Jan-1999, http://purl.oclc.org/metadata/dc/

[DELCA] Delcambre, L. M. L., Maier, D., Reddy, R.., Anderson, L.: Structured Maps: modeling explicit semantics over a universe of information. In: *International Journal on Digital Libraries*, Springer-Verlag, 1997

[DOM] World Wide Web Consortium: Document Object Model Specification 1.0, W3C Recommendation 20-Jul-1998, http://www.w3.org/TR/1998/WD-DOM-19980720

[DSIG] World Wide Web Consortium: Digital Signature Initiative Overview, http://www.w3.org/DSig/Overview.html

[ISO13250] ISO JTC1/WG4, Information Processing – SMGL Applications – Topic Navigation Maps, *ISO/IEC CD 13250, Committee Draft,* http://www.hightext.com/tnm/psjan98.htm

[MCF] World Wide Web Consortium: Meta Content Framework Using XML, W3C Note, 24-Jun-1997, http://www.w3.org/TR/NOTE-MCF-XML-970624

[MILL] Miller, E.: An Introduction to the Resource Description Framework. In: *D-Lib Magazine*, ISSN 1082-9873, May 1998, http://www.dlib.org/dlib/may98/miller/05miller.html

[MMBROK] Swedish Institute for Systems Development, SISU: Multimedia Broker, Developing Critical Support Tools for Multimedia Publishing, http://www.sisu.se/projects/mmbroker/what.htm

[MOZ1] Guha, Churchill, R., Giannandrea, J.,:RDF. *The Mozilla Organization*, http://www.mozilla.org/rdf/doc/index.html

[P3P] World Wide Web Consortium: P3 Project Overview, http://www.w3.org/P3P/

[PICS] World Wide Web Consortium: PICSRules 1.1, W3C Recommendation 29-Dec-1997, http://www.w3.org/TR/REC-PICSRules-971229

[RDF] World Wide Web Consortium: Resource Description Framework (RDF) Model and Syntax, W3C Working Draft 7-Oct-1998, http://www.w3.org/1998/10/WD-rdf-syntax-19981008

[RDF4XML] http://www.alphaworks.ibm.com

[RDFFAQ] World Wide Web Consortium: Frequently Asked Questions about RDF, http://www.w3.org/RDF/FAQ

[RDFINTRO] World Wide Web Consortium: Introduction to RDF Metadata, W3C Note 13-Nov-1997, http://www.w3.org/TR/NOTE-rdf-simple-intro-971113.html

[RDFSCH] World Wide Web Consortium: Resource Description Framework (RDF) Schema specification, W3C Working Draft 30-Oct-1998, http://www.w3.org/TR/1998/WD-rdf-schema-19981030/

[RDU] Research Data Network, Resource Discovery Unit, DSTC: Resource Discovery – A definition, DSTC Symposium, 1995, http://www.dstc.edu.au/RDU/RD-Defn/

[WP] Lagoze, C., Digital Library Research Group, Cornell University, D-Lib Magazine, Jul/Aug-1996, ISSN 1082-9873, http://www.dlib.org/dlib/july96/lagoze/07lagoze.html

[XML4J] http://www.alphaworks.ibm.com

[XMLDATA] World Wide Web Consortium: XML-Data, W3C Note, 5-Jan-1998, http://www.w3.org/TR/1998/NOTE-XML-data-0105

APPENDIX A. THE EXTENSIBLE MARKUP LANGUAGE

The Extensible Markup Language (XML) is a simplified version of SGML, specifically designed with the World Wide Web in mind. An XML document is also an SGML document. XML has brought with it a number of other standards (or proposed standards), e.g. XLink (*XML Linking Language*) and XSL (*XML Stylesheet Language*).

> XML differs from HTML in that it doesn't mix *content* with *appearance*.

WHY XML? WHY NOT JUST HTML OR SGML?

It's all the same, isn't it? No, not quite. XML is a simpler version of SGML. It omits the more complex and less-used parts of SGML in return for the benefits of being easier to write applications, easier to understand, and more suited to delivery and interoperability over the web [XMLFAQ].

With both XML and SGML you can use DTD to define your own markup language. HTML is just one of those markup languages possible to define with XML/SGML. In fact, in the specs for HTML 4.0 [HTML] an HTML document is defined as:

> An HTML document is an SGML document that meets the constraints of this specification (HTML 4.0 specs)

XML or Extensible Markup Language has really taken off with major software vendors supporting it. The media hype has resulted in a large number of documents about XML on WWW. We will look at XML as it is the preferred way to transport RDF [RDF].

XML allows people to create their own markup languages, tailor made for the particular needs of a particular community. One example of such a markup language is *MathML*, which can express advanced mathematical formulas. Other markup languages can be developed, e.g. one for chemical formulas and another for musical scores.

XML is not concerned with how these documents should be displayed. It's up to the applications that read the documents to decide what to do with them. A document describing music (a "MusicML" document) may be *played*, rather than *displayed*, by the application

If we return to our example on searching the WWW for information on books *written by* Winston Churchill we realize the benefits of XML. We could invent a markup language for describing documents about books. Part of a document could look as follows:

```
<book>
    <title>Liberalism and the Social Problem</title>
    <author>Winston Churchill</author>
    <publisher>Hodder & Stoughton</publisher>
    <comment>This early speech collection of the fighting radical("a traitor to his
class") is now extremely scarce and many predict it will soar in value over the
next decade.</comment>
</book>
```

An "intelligent" search engine could immediately conclude that this document is relevant for our query. Of course, for this to be useful we need to standardize how documents (and other resources) should be described, and that's where RDF and *schemas* come into play.

XML DOCUMENTS

The XML specification defines a class of data objects called *XML documents*. Each XML document has both a logical and a physical structure. *Physically* the document is composed of units called *entities*. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity. *Logically*, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup [XML].

This is an example of a complete XML document:

```
<?xml version="1.0"?>
<story>The rabbit used to hang out in bars</story>
```

The first line is an example of a *Document Type Declaration*. The document type declaration isn't mandatory. This is also a complete XML document:

```
<story>The rabbit used to hang out in bars</story>
```

DOCUMENT TYPE DEFINITION – DTD

The XML document type declaration contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a *Document Type Definition (DTD)*. The document type declaration can point to an external subset containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. The DTD for a document consists of both subsets taken together [XML].

The DTD defines constraints on the logical structure of the document. The following document has an external DTD that defines it syntax:

```
<?xml version="1.0"?>
<!DOCTYPE story SYSTEM "standard_story.dtd">
<story>The rabbit used to hang out in bars</story>
```

The DTD could look like this:

```
<!DOCTYPE standard_story.dtd
<!ELEMENT story (#PCDATA)>
>
```

For further information on XML and DTD please refer to W3Cs specification [XML].

WELL-FORMED VS. VALID XML

There are essentially two related types of XML documents: *Well-Formed* and *Valid*. A well-formed XML document conforms to the general rules of XML syntax, which are more rigorous than those of either HTML or SGML. XML character data is never left an ending markup designation of some sort, either an end tag such as in the element `<story> </story >` or a special empty element tag with a forward slash before the right-angle bracket, such as `<story/>.` [CNET1] [XMLFAQ]

Valid XML documents are ones that conform to a specific DTD. Confirming the validity of XML documents is largely the work of authoring and publishing tools, whereas XML-capable browsers need only check for well-formedness in order to read XML documents. [CNET1] [XMLFAQ]

XML NAMESPACES

Although *XML namespaces* specs [XMLNSP] is still a working draft, the notion of namespaces is fundamental to implementing RDF on top of XML.

XML namespaces provide a simple method for qualifying names used in XML documents by associating them with namespaces identified by URI. Namespaces provide universal names, whose scope extends beyond their containing document. The combination of the universally managed URI namespace and the local name produces names that are guaranteed universally unique [XMLNSP].

Here is an example of an XML namespace in use:

```
<?xml:version="1.0"?>
<books      xmlns="http://books.com/schemas/"
            xmlns:de="http://ekengren.com/schemas">
    <Title>Liberalism and the Social Problem</Title>
    <Author>Winston Churchill<Author>
    <de:Rating>Interesting</de:Rating>
</books>
```

XML namespaces allows us to combine elements and attributes whose semantics (schemas) are defined by different authorities. In this example an XML processor can look up the semantics for the de prefix and find out that the possible ratings are 1) interesting 2) cool or 3) crazy. It is up to the processor or application to decide what to do with this information.

The example declares books as the default namespace. This means that *title* and *author* will be resolved as *books:title* and *books:author*.

Currently there is no standard way for RDF processors to read and process such schemas. Document Object Model

The *Document Object Model (DOM)* specification defines a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents [DOM].

The DOM applies both to HTML and XML documents to provide a single document metaphor. Browser vendors such as Microsoft [MSDOM] and Netscape have developed proprietary documents models which has caused problems for site builders who want their content to work on both platforms. A more formal description of the DOM for XML, check out W3C [DOMXML]

XML STYLESHEETS – XSL

XML documents don't specify how they are to be displayed. That is the role of the *Extensible Stylesheet Language (XSL)*. The W3C has issued a working draft for the XSL requirements. As usual the working draft is subject to change [XSL].

How are stylesheets used? The simplest application is to feed an XML document and an XSL stylesheet into an XSL processor. The processor produces a displayable document in a format determined by the rules in the stylesheet:

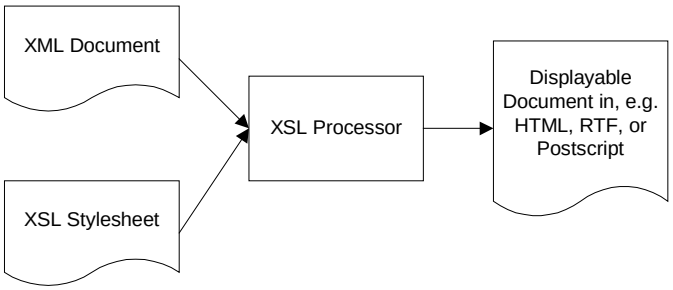


Figure 38: XSL processing producing HTML, RDF or Postscript

You can make web browsers capable of reading an XML document an XSL stylesheet in the same way current browsers read and display HTML documents. The beauty of the XML/XSL approach is that a document's *content* is separated from its *appearance*.

A good introduction to XSL and the Microsoft XSL processor can be found in [XSLTUTOR]. For an example of what an XSL Stylesheet might look like, take a look in Appendix C. Note that the XSL Stylesheet is expressed in XML and is in fact an XML document.

REFERENCES

[CNET1] Gorman, T.: 20 Questions on XML, 10-Mar-1998, http://www.builder.com/Authoring/Xml20/index.html

[DC5] Weibel, S., Hakala, J., DC-5: The Helsinki Metadata Workshop, D-Lib Magazine, Feb-1998, ISSN 1082-9873, http://www.dlib.org/dlib/february98/02weibel.html

[DOM] World Wide Web Consortium: Document Object Model Specification 1.0, W3C Recommendation 20-Jul-1998, http://www.w3.org/TR/1998/WD-DOM-19980720

[DOMXML] World Wide Web Consortium: Document Object Model (XML), W3C Working Draft 16-Apr-1998, http://www.w3.org/TR/WD-DOM/level-one-xml.html

[HTML] World Wide Web Consortium: HTML 4.0 Specification, W3C Recommendation, revised on 24-Apr-1998, http://www.w3.org/TR/1988/REC-html40-19980424

[MSDOM] Microsoft Corporation: The XML Object Model in Internet Explorer 4.0, 7-Jan-1998, http://www.microsoft.com/xml/articles/xmlmodel.asp

[RDF] World Wide Web Consortium: Resource Description Framework (RDF) Model and Syntax, W3C Working Draft 8-Oct-1998, http://www.w3.org/1998/10/WD-rdf-syntax-19981008

[RDFFAQ] World Wide Web Consortium: Frequently Asked Questions about RDF, http://www.w3.org/RDF/FAQ

[XML] World Wide Web Consortium: Extensible Markup Language (XML), 8-Dec-1997, http://www.w3.org/TR/PR-xml-971208

[XMLFAQ] World Wide Web Consortium: Frequently Asked Questions about the Extensible Markup Language, the XML FAQ, Version 1.41 (6-oct-1998). http://www.ucc.ie/xml/

[XMLNSP] World Wide Web Consortium: Namespaces in XML, W3C Working Draft 16-Sep-1998, http://www.w3.org/TR/1998/WD-xml-names-19980916

[XSL] World Wide Web Consortium: XSL Requirements Summary, W3C Working Draft 11-May-1998, http://www.w3.org/TR/WD-XSLReq

[XSLTUTOR] Microsoft Corporation: XSL Tutorial, 7-jan-1998, http://www.microsoft.com/xml/xsl/tutorial/tutorial.asp

APPENDIX B. RDF

RDF IN DETAIL, TRIPLES

RDF CORE; "LAYER 0"

The core data model is formally defined in [RDF] by W3C. In short, every *Statement* is a 3-tuple, or *triple*, whose elements are:

```
{Predicate,Subject,Object}
```

We can view a set of statements as a directed-labeled graph. Each triple is a {p, s, o} is an arc from s to o labeled by p:

[s] – p --> [o]

This can be read as *o is the value of p for s*, *s has a property p with a value o*, or the *p of s is o*.

The RDF data model is built up entirely on these triples. The first element of a triple (Predicate above), is called *Property*. The second element is a *Resource* and the third is a Resource or a literal. The tripe as a whole is called a *Statement*.

{Property, Resource, Resource | Literal}

In the RDF data model both the resources being described and the values describing them are represented as nodes in a directed labeled graph. The arcs connecting pairs of nodes correspond to the names of the properties. Each arc *labeled* by the corresponding property.

The RDF Layer 0 gives us the most fundamental tools to describe our resources. An RDF statement can be written as:

```
[Resource R] --- Property P--> [Value V]
```

This would make the triple:

```
{[Property P], Resource R, [Value V]}
```

We describe the resource R with a Property arc to a Value V. The value may be either a resource or a literal. A literal should be stored using a direct encoding of ISO/IEC 10646 or an encoding which can be mapped to ISO/IEC 10646 [ISO10646]. In [RDF]: "Language tagging is part of the string value; it is applied to sequences of characters within an RDF string and does not have an explicit manifestation in the data model."

Now we are ready to start describing our resources (although we haven't yet defined what properties to use):

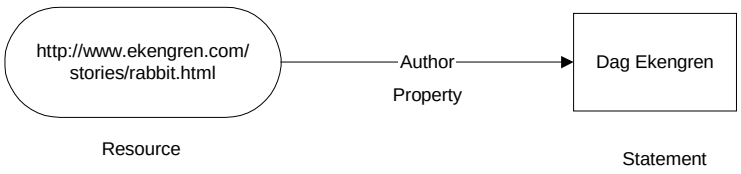


Figure 39: Simple RDF statement

The semantics of the above is "The author of the resource rabbit.html is Dag Ekengren". The same can be expressed with a triple as:

```
{Author, [http://www.ekengren.com/stories/rabbit.html], "Dag
Ekengren"}
```

The RDF data model is independent of any particular syntax. At the core RDF is a syntax-independent model for representing resources and their corresponding descriptions [RDF]. Two expressions in different syntax can be converted to the RDF Data Model and then it can be determined if the expressions' semantics are the same, unambiguously. A good illustration of this is found in [MILL]: Consider the following statements:

> "The author of Document 1 is John Smith"
> "John Smith is the author of Document 1"

To humans, these statements convey the same meaning. To a machine, however, these are completely different strings. RDF attempts to provide an unambiguous method of expressing semantics in a machine-readable encoding. The triple for both expressions is:

> {Author, [Document 1], "John Smith"}

If a number of triples all refer to the same the same resource, they can be grouped as a single unit. [RDF]. This makes things a little easier as a single resource often has many properties:

```
{Author; [rabbit.html], "Dag Ekengren"}
{Title,  [rabbit.html], "The nosy rabbit"}
{Rating, [rabbit.html], "Cool"}
```

These three Triples can be considered a *Description*. The Description can have its own identification, in effect, the Description is also a Resource. This Resource, the Description, can therefor be described by Properties, i.e. a Description can itself be used as the source node of other arcs describing properties of the Description [RDF]].

One or more schemas define the properties in a given Description, as well as any characteristics or restrictions of the property values themselves. Each property used in a Description is declared to be from exactly one schema. Schemas will be discussed below.

UTILITY RELATIONS; "LAYER 1"

The formal definition for layer 1 is given by W3C in [RDF]. The utility relations enable the "Descriptions of Descriptions" feature of RDF by a process called *reification*. Reification allows us to express modalities (e.g. beliefs about properties) or simply attach any properties to other properties.

What we would like to do is to take a property (i.e. a triple) and in some well-defined way create an identifiable node from that property.

The property {Author, [rabbit.html], "Dag Ekengren"} can be reified as:

```
{rdf:type,      [Statement1], [rdf:statement]}
{rdf:predicate, [Statement1], [Author]}
{rdf:subject,   [Statement1], [rabbit.html]}
{rdf:object,    [Statement1], "Dag Ekengren"}
```

The property is now a named node (Statement1) which in turn has four properties. This allows us to give properties to Statement1. We can express semantics such as "*John says* that Dag Ekengren is the author of rabbit.html":

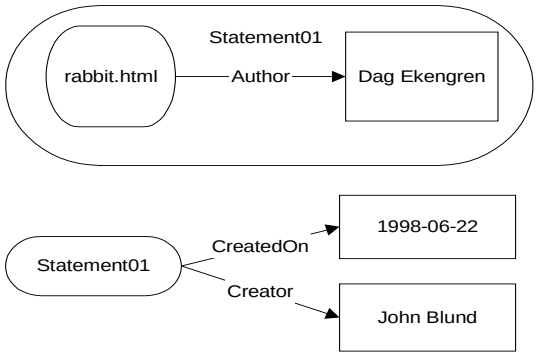


Figure 40: Example of "Description of Description"

More examples of *reification* can be found in [RDF][MILL].

RDF SCHEMA

The standard for RDF Schemas is still being discussed by the W3C [RDFSCH]. The specifications are currently in a working draft which should only be used to implement experimental software. There are currently no RDF parsers available that conform to the RDF Schema standard.

In short, RDF Schemas are used to declare the properties and semantics for a vocabulary. Each metadata community can develop an RDF Schema for their particular needs. These Schemas should be machine understandable in a standard way to provide interoperability between

different RDF Schema capable software. RDF Schemas define the valid properties in a given RDF description, as well as any characteristics or restrictions of their values [MILL].

An RDF Schema parser will be able to load Schemas from the metadata community that provides them and validate documents that use the Schemas.

Type System

The RDF Schema introduces a relation called *rdfs:subClassOf.* In the Model and Syntax paper [RDF] W3C defined *rdf:type* which enabled a node to be an instance of a particular class. However, in an object-oriented fashion, classes may be used in a hierarchical manner. The RDF Schema property type *rdfs:subClassOf* indicates such a relationship. Another development of classes would be to introduce a set of "data types", such as *integer* or *date.* [RDFSCH]

Constraints

The other aspect of RDF Schemas is the ability to declare constraints associated with classes and property types. Examples of constraints are:

> The value of a statement should be a resource of a designated class.
>
> Constraints on the properties that can be used to describe a resource is expressed with allowedProperties.
>
> The cardinality of a property, that is the number of properties of a given type that a resource may have.

In a way, the RDF Schema is to RDF what the DTD *(Document Type Definition)* is to XML. XML will be discussed later as a tool to expressing and *transporting* RDF semantics.

RDF SAMPLES

SOWTWARE AGENTS

This is an RDF description of a contract for a software agent. The contract specifies the interests of the agent's owner. Other trade contracts specify what for example web sites and online stores have to offer. When a user who has the agent client software installed visits a web site with a web browser, the client and server agent software exchanges machine understandable trade contracts. In a process called negotiation, the client and server agents tries to agree on the terms of the other parties contract. If an agreement is made, the user can be notified that the web site has something to offer that might interest him. The user can then decide what to do, for example make a purchase. Of course, such actions can be triggered automatically.

Such agent trade contracts are great RDF applications. A sample instance of a trade contract is:

```
<?xml version="1.0"?>
<!-- Some instances of agent contracts and other cool stuff -->
<rdf:RDF   xmlns:rdf="http://www.w3.org/schemas/rdf-schema"
    xmlns:agent_contracts="http://miller.sisu.se/schemas/agent_contracts.xml"
    xmlns:agent_trade_objects="
http://miller.sisu.se/schemas/agent_trade_objects.xml">
<agent_contracts:Trade_contract>
```

```
 <agent_trade_objects:trade_object>
  <agent_trade_objects:Cd>
   <agent_trade_objects:title>Oxygene</agent_trade_objects:title>
   <agent_trade_objects:artist>Jarre</agent_trade_objects:artist>
  </agent_trade_objects:Cd>
 </agent_trade_objects:trade_object>
 <agent_contracts:price>200</agent_contracts:price>
 <agent_contracts:buyer>
  <agent_contracts:Person>
<agent_contracts:agent_address>map://scheutz.sics.se:11000/auto</agent_contracts:a
gent_address>
   <agent_contracts:name>NF</agent_contracts:name>
  </agent_contracts:Person>
 </agent_contracts:buyer>
</agent_contracts:Trade_contract>
</rdf:RDF>
```

REFERENCES

[ISO10646] - ISO/IEC 10646

[MILL] Miller, E.: An Introduction to the Resource Description Framework. In: *D-Lib Magazine*, ISSN 1082-9873, May 1998, http://www.dlib.org/dlib/may98/miller/05miller.html

[RDF] World Wide Web Consortium: Resource Description Framework (RDF) Model and Syntax, W3C Working Draft 8-Oct-1998, http://www.w3.org/TR/WD-rdf-syntax/

[RDFSCH] - World Wide Web Consortium: Resource Description Framework (RDF) Schemas, W3C Working Draft 30-Oct-1998, http://www.w3.org/TR/1998/WD-rdf-schema-19981030/

APPENDIX C. THE XML TO RDF STYLESHEETS

This section provides examples of XSL stylesheets that convert the internal XML format used in the Multimedia Broker to RDF. Similar stylesheets can be created to generate RDF from other platforms that use XML.

The following is an example of the internal XML [illegible] a Broker by a database query. It represents a math exercise.

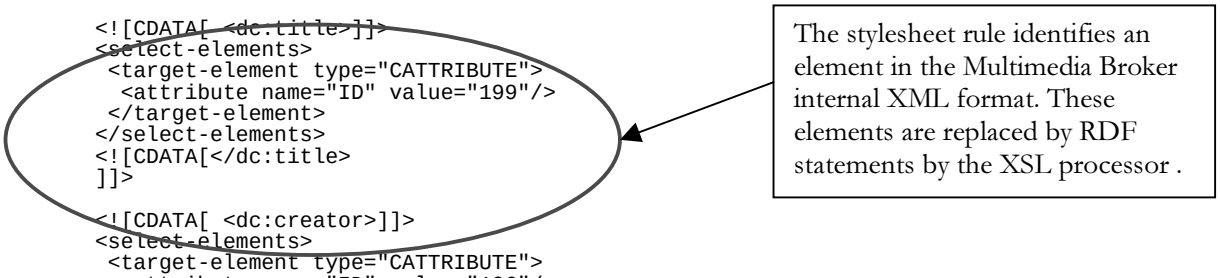
This CATTRIBUTE element is matched by the marked stylesheet rule below.

```
<ROW>
<CATTRIBUTE INTERNALID="124" DT="4" SQLDT="4"
    ISNULL="false">3</CATTRIBUTE>
<CATTRIBUTE ID="9001" NAME="Uppgift.Presentation" ENTITY="Uppgift"
ATTRIBUTE="Presentation" DT="10" SQLDT="12" SQLPREC="255"
    ISNULL="false">127.0.0.1/VNDatabaser/Edu/VNContentServer.asp?INFOPROD_ID=19&a
    mp;ProdId= 3</CATTRIBUTE>
<CATTRIBUTE ID="199" NAME="Uppgift.Namn" ENTITY="Uppgift" ATTRIBUTE="Namn" DT="10"
    SQLDT="12" SQLPREC="50" ISNULL="false">Problem exercise: Problem 2</CATTRIBUTE>
<CATTRIBUTE ID="190" NAME="Uppgift.Uppgift" ENTITY="Uppgift" ATTRIBUTE="Uppgift"
    DT="10" SQLDT="12" SQLPREC="255" ISNULL="false">
<A HREF =
    "http://127.0.0.1/VNDatabaser/Edu/VNContentServer.asp?INFOPROD_ID=19&amp;Prod
    Id= 3">Problem exercise: Problem 2</A>
</CATTRIBUTE>
<CATTRIBUTE ID="176" NAME="Uppgift.Skapad" ENTITY="Uppgift" ATTRIBUTE="Skapad"
    DT="10" SQLDT="12" SQLPREC="50" ISNULL="false">1998-12-09</CATTRIBUTE>
<CATTRIBUTE ID="105" NAME="Uppgift.Uppgift_Id" ENTITY="Uppgift"
    ATTRIBUTE="Uppgift_Id" DT="4" SQLDT="4" SQLPREC="10"
    ISNULL="false">3</CATTRIBUTE>
<CATTRIBUTE ID="106" NAME="Uppgift.Upphovsman" ENTITY="Uppgift"
    ATTRIBUTE="Upphovsman" DT="10" SQLDT="12" SQLPREC="255" ISNULL="false">Peter
    Rosengren</CATTRIBUTE>
<CATTRIBUTE INTERNALID="134" DT="4" SQLDT="4" SQLPREC="10"
    ISNULL="false">7</CATTRIBUTE>
<CATTRIBUTE ID="175" NAME="Sv&aring;righetsgrad.Grad"
    ENTITY="Sv&aring;righetsgrad" ATTRIBUTE="Grad" DT="10" SQLDT="12" SQLPREC="50"
    ISNULL="false">Middle</CATTRIBUTE>
<CATTRIBUTE INTERNALID="140" DT="4" SQLDT="4" SQLPREC="10"
    ISNULL="false">6</CATTRIBUTE>
<CATTRIBUTE ID="126" NAME="Moment.Beskrivning" ENTITY="Moment"
    ATTRIBUTE="Beskrivning" DT="10" SQLDT="12" SQLPREC="255"
    ISNULL="True">NULL</CATTRIBUTE>
<CATTRIBUTE ID="125" NAME="Moment.Namn" ENTITY="Moment" ATTRIBUTE="Namn" DT="10"
    SQLDT="12" SQLPREC="255" ISNULL="false">Fractions and potency</CATTRIBUTE>
<CATTRIBUTE INTERNALID="135" DT="4" SQLDT="4" SQLPREC="10"
    ISNULL="false">1</CATTRIBUTE>
<CATTRIBUTE ID="123" NAME="Kategori.Namn" ENTITY="Kategori" ATTRIBUTE="Namn"
    DT="10" SQLDT="12" SQLPREC="50" ISNULL="false">Tr&auml;na
    Probleml&ouml;sning</CATTRIBUTE>
<CATTRIBUTE ID="195" NAME="Kategori.Beskrivning" ENTITY="Kategori"
    ATTRIBUTE="Beskrivning" DT="10" SQLDT="12" SQLPREC="255"
    ISNULL="True">NULL</CATTRIBUTE>
<CATTRIBUTE INTERNALID="139" DT="4" SQLDT="4" SQLPREC="10"
    ISNULL="false">4</CATTRIBUTE>
<CATTRIBUTE ID="194" NAME="F&ouml;rkunskapskrav.Beskrivning"
    ENTITY="F&ouml;rkunskapskrav" ATTRIBUTE="Beskrivning" DT="10" SQLDT="12"
    SQLPREC="255" ISNULL="True">NULL</CATTRIBUTE>
<CATTRIBUTE ID="193" NAME="F&ouml;rkunskapskrav.Namn"
    ENTITY="F&ouml;rkunskapskrav" ATTRIBUTE="Namn" DT="10" SQLDT="12" SQLPREC="50"
    ISNULL="false">The four ways of counting</CATTRIBUTE>
<CATTRIBUTE INTERNALID="136" DT="4" SQLDT="4" SQLPREC="10"
    ISNULL="false">2</CATTRIBUTE>
<CATTRIBUTE ID="122" NAME="Bild.Upphovsman" ENTITY="Bild" ATTRIBUTE="Upphovsman"
    DT="10" SQLDT="12" SQLPREC="255" ISNULL="false">Yngve Pavasson</CATTRIBUTE>
<CATTRIBUTE ID="121" NAME="Bild.Bildtyp" ENTITY="Bild" ATTRIBUTE="Bildtyp" DT="10"
    SQLDT="12" SQLPREC="255" ISNULL="false">GIF</CATTRIBUTE>
<CATTRIBUTE ID="192" NAME="Bild.Namn" ENTITY="Bild" ATTRIBUTE="Namn" DT="10"
    SQLDT="12" SQLPREC="255"
    ISNULL="false">D:&Ouml;Inetpub&Ouml;wwwroot&Ouml;VNDatabaser&Ouml;Edu&Ouml;Bl
    ob&Ouml;0331-31.gif</CATTRIBUTE>
<CATTRIBUTE INTERNALID="128" DT="4" SQLDT="4" SQLPREC="10"
    ISNULL="false">2</CATTRIBUTE>
<CATTRIBUTE ID="108" NAME="Uppgiftstext.Text" ENTITY="Uppgiftstext"
    ATTRIBUTE="Text" DT="10" SQLDT="65535" SQLPREC="1073741824" ISNULL="false">A
rope is 120 cm. The rope is to be cut into two parts.<BR/>One of the parts shall
be twice as long as the other.<BR/>State the lenghts of the two
parts.</CATTRIBUTE>
<CATTRIBUTE INTERNALID="138" DT="4" SQLDT="4" SQLPREC="10"
```

```
    ISNULL="false">2</CATTRIBUTE>
<CATTRIBUTE ID="196" NAME="L&ouml;sning.Text" ENTITY="L&ouml;sning"
    ATTRIBUTE="Text" DT="10" SQLDT="12" SQLPREC="255" ISNULL="false">120 = 80 +
    40</CATTRIBUTE>
<CATTRIBUTE ID="189" NAME="L&ouml;sning.Kommentar" ENTITY="L&ouml;sning"
    ATTRIBUTE="Kommentar" DT="10" SQLDT="12" SQLPREC="255"
    ISNULL="True">NULL</CATTRIBUTE>
<CATTRIBUTE INTERNALID="137" DT="4" SQLDT="4" SQLPREC="10"
    ISNULL="false">3</CATTRIBUTE>
<CATTRIBUTE ID="173" NAME="Matematiktal.Tal" ENTITY="Matematiktal" ATTRIBUTE="Tal"
    DT="10" SQLDT="12" SQLPREC="255" ISNULL="True">NULL</CATTRIBUTE>
</ROW>
```

The XML document and the XSL stylesheet below are fed to the XSL processor. This creates the RDF document.

```
<xsl>
<rule>
 <root/>
 <![CDATA[
 ]]>
 <?xml version="1.0"?>
 <rdf>
 <children/>
 </rdf>
</rule>

<!-- Don't display header -->
<rule>
 <target-element type="ROW" position="first-of-type"/>
 <empty/>
</rule>

<rule>
 <target-element type="CATTRIBUTE">
  <attribute name="ID" value="190"/>
 </target-element>
 <select-elements>
  <target-element type="A"/>
 </select-elements>
</rule>

<rule>
 <target-element type="A"/>
 <![CDATA[

 <edu:Exercise ID="]]><eval>getAttribute('HREF')</eval><![CDATA[">
 ]]>
</rule>

<rule>
 <target-element type="ROW"/>

 <![CDATA[
 <!-- Instance of class edu:Exercise -->]]>

 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="190"/>
  </target-element>
 </select-elements>

 <![CDATA[ <dc:title>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="199"/>
  </target-element>
 </select-elements>
 <![CDATA[</dc:title>
 ]]>

 <![CDATA[ <dc:creator>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="106"/>
  </target-element>
 </select-elements>
 <![CDATA[<"</dc:creator>
 ]]>

 <![CDATA[ <dc:date>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
```



```
   <attribute name="ID" value="176"/>
  </target-element>
 </select-elements>
 <![CDATA[</dc:Date>
 ]]>

 <![CDATA[ <dc:relation rdf:resource="xxx"/>
]]>

 <![CDATA[ <edu:id>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="105"/>
  </target-element>
 </select-elements>
 <![CDATA[</edu:id>
 ]]>

 <![CDATA[ <edu:level>]]>
 <select-elements>  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="175"/>
  </target-element>
 </select-elements>
 <![CDATA[</edu:level>
 ]]>

 <![CDATA[ <edu:category>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="123"/>
  </target-element>
 </select-elements>
 <![CDATA[</edu:category>
 ]]>

 <![CDATA[ <edu:subject>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="125"/>
  </target-element>
 </select-elements>
 <![CDATA[</edu:subject>
 ]]>

 <![CDATA[ <edu:prerequisites>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="193"/>
  </target-element>
 </select-elements>
 <![CDATA[</edu:prerequisites>
 ]]>

 <![CDATA[ <edu:content>
   <rdf:Description>
    <edu:text>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="108"/>
  </target-element>
 </select-elements>
 <![CDATA[</edu:text>
    <edu:image resource="]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="192"/>
  </target-element>
 </select-elements>
 <![CDATA["/>
    <edu:solution>]]>
 <select-elements>
  <target-element type="CATTRIBUTE">
   <attribute name="ID" value="196"/>
  </target-element>
 </select-elements>
 <![CDATA[</edu:solution>
   </rdf:Description>
  </edu:content>
 ]]>

 <![CDATA[</edu:Exercise>
 ]]>
</rule>
```

```
<rule>
 <target-element type="RESULT"/>
 <children/>
</rule>

<rule>
 <targem-element type="QUERY"/>
 <children/>
</rule>

</xsl>
```

The XSL processor produces the following output after processing the XML document and stylesheet above:

```
<rdf:RDF>
 <edu:Exercise ID =
"http://127.0.0.1/VNDatabaser/Edu/VNContentServer.asp?INFOPROD_ID=19&ProdId= 3">
  <dc:title>Problem exercise: Problem 2</dc:title>
  <dc:creator>Peter Rosengren<"</dc:creator>
  <dc:date>1998-12-09</dc:date>
  <dc:relation rdf:resourc e=
"http://127.0.0.1/VNDatabaser/Edu/VNContentServer.asp                    Id= 33"/>
  <edu:id>3</edu:id>
  <edu:level>Middle</edu:level>
  <edu:category>Tr&#228;na Probleml&#246;sning</edu:c
  <edu:subject>Fractions and potency</edu:subject>
  <edu:prerequisites>The four ways of counting</edu:p
  <edu:content>
   <rdf:Description>
    <edu:Text>A rope is 120 cm. The rope is to be cut into two parts.  One of the
parts shall be twice as long as the other.  State the lenghts of the two
parts.</edu:Text>
     <edu:Image resource =
"D:&#246;Inetpub&#246;wwwroot&#246;VNDatabaser&#246;Edu&#246;Blob&#246;0331-
31.gif"/>
    <edu:solution>120 = 80 + 40</edu:solution>
   </rdf:Description>
  </edu:content>
 </edu:Exercise>
<rdf:RDF>
```

This refers to the math book which this exercise is a part of

APPENDIX D. THE RDFCLIENT SOFTWARE

This appendix provides some additional information about the RdfClient software that was developed as a part of this project.

SYSTEM REQUIREMENTS

- Java Virtual Machine 1.2 or later
- IBM's XML4j (XML for Java) 1.1.4 or later
- IBM's RDFforXML November 1998

THE RDFFORXML PARSER CLASSES

The main classes for building the RDF data model are:

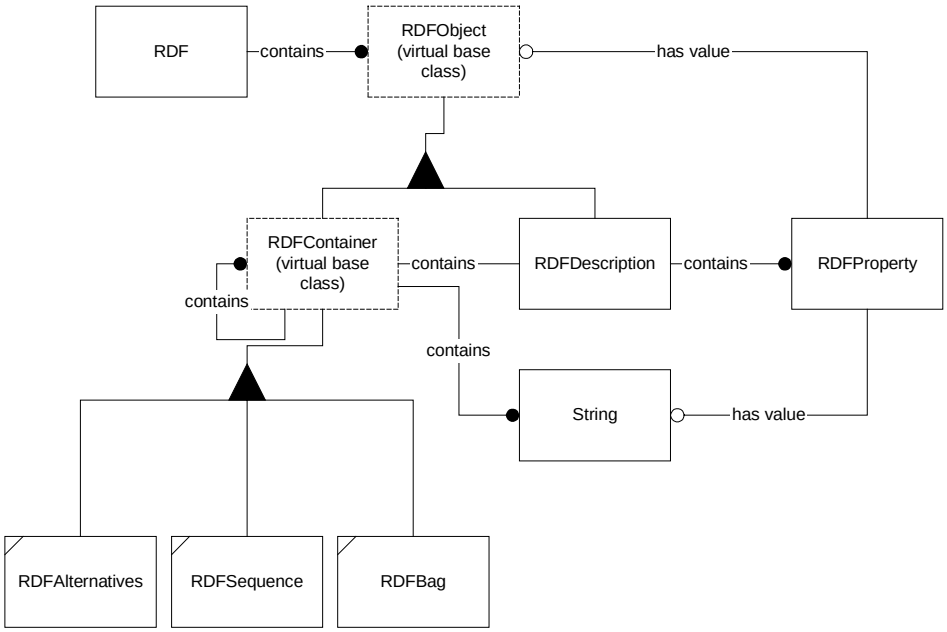


Figure 41: The RDF parser's class hierarchy

APPENDIX E. INSTRUCTIONAL MANAGEMENT SYSTEM – IMS

The Instructional Management System Project (IMS) is an investment membership by a number of US academic, commercial and government organizations dedicated to facilitating the growth of distributed learning on the Internet [IMSSPEC]. The IMS specification is a huge document covering the technical details for an online learning system. Here is the list of requirements, just to give you a feel of the complexity of IMS: *Group Management, Personal Profile Management, Activity Management, Assessment and Certification Management, Content Management, Commerce and Licensing Management, Security Management, Technical Administration Management* [IMSREQ].

We are mainly concerned with the metadata parts of IMS. The metadata elements are based on Dublin Core, but include additional elements with semantics for associating and aggregating resources to form complex structures. They can be found at [IMSMETA].

> The IMS Metadata Specification consists of several parts: *Repository System, Dictionaries* and *Schema Libraries*.

REPOSITORY SYSTEM

The Tiered[22] Repository System [TIERS] is a hierarchy of *Repositories*. A Repository is a place where *dictionaries*, *schema libraries*, and a *list of entities* are maintained.

DICTIONARY

The *Dictionary* [IMSDIC] contains definitions for a number of *fields* that are used to describe resources. An example of such a dictionary is the *Dublin Core*, which is described elsewhere in this document. The dictionaries in IMS are the same as the *vocabularies* used in RDF. Examples of fields in the Dublin Core dictionary are *Title*, *Subject* and *Creator*.

SCHEMA LIBRARY

A *Schema Library* is a collection of metadata structures called *Schemas*. All schemas are derived from *The Master Schema* [IMSMS] defined by the IMS metadata specification. The schemas are organized after OOD (Object Oriented Design) principles by allowing inheritance from other schemas. This is illustrated by the example in the following section.

A SIMPLIFIED IMS METADATA STRUCTURE EXAMPLE

We have a master schema, which is a collection of fields from our dictionary. The master schema in our example contains the fields *Identifier, Title, Language, Granularity* and *Status*. The fields in the master schema have different levels of *obligation*. The (m) after a field indicates that this field is *mandatory* for all schemas derived from this master schema. An (o) indicates that a field is optional. In our example, the fields *Identifier, Title* and *Language* are mandatory.

We derive *My Base Schema* from the master schema. My Base Schema contains a subset of the fields provided in the master schema. We have chosen not to include the field *Granularity*, which was marked as optional. Note that in My Base Schema, we have made the *Status* field mandatory. This illustrates an important rule: We are allowed to make our base schema more restrictive (with

[22] *Tiered* means "arranged in layers".

more fields marked as *mandatory*). However, the derived schema can never be made less restrictive. My Base Schema serves as our *Base Schema*. It is not used directly but is inherited (and extended) by more specific schemas such as *My Image Schema* and *My Book Schema* in the figure below.

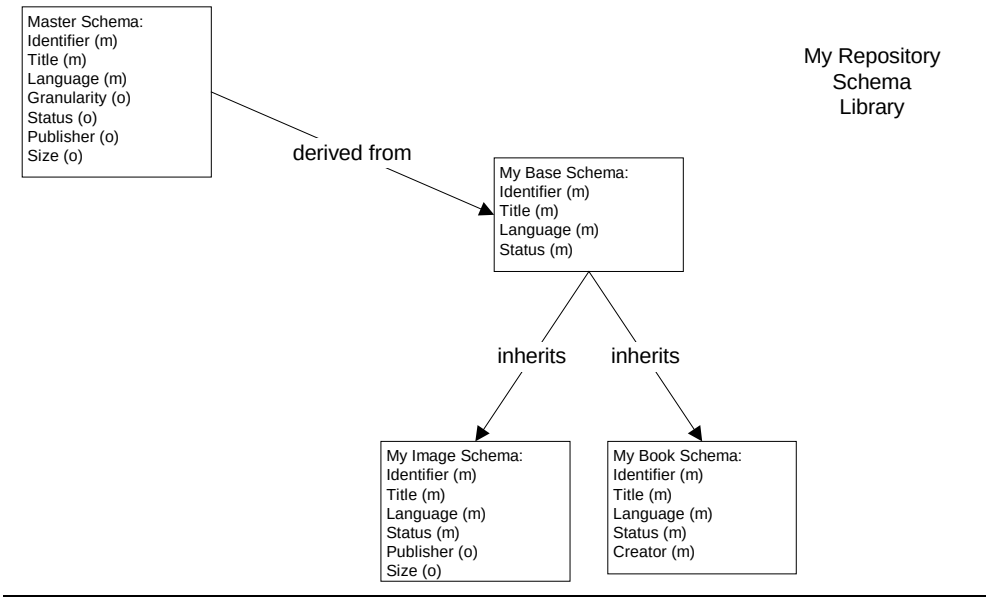


Figure 42: This is a simplified example of a Master Schema, a Base Schema derived from the Master Schema. My Base Schema doesn't contain all *the fields from the Master Schema. It contains a selection of fields. Note that My Base Schema is more restrictive than the Master Schema, making the field Status mandatory. My Book Schema inherits from My Base Schema, adding the field Creator, which is marked as mandatory. My Image Schema adds the fields* Publisher *and* Size.

The rule that we only can make schemas *more* restrictive (but not less) applies to inheritance as well.

Now that we have defined a schema library we can make instances of our schemas to describe a particular image or book:

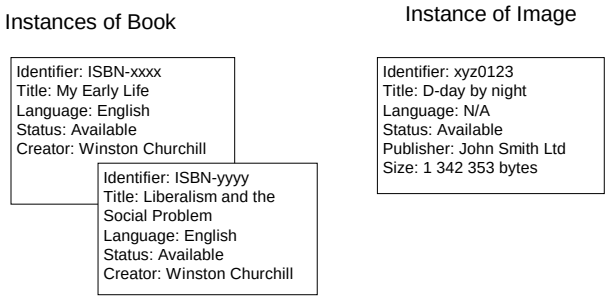


Figure 43: Metadata instances of the schemas Book and Image.

Please note that this is a simplified example. The IMS Schema Library (the schema library in the IMS Repository) requires many more fields from the dictionary. In reality *My Depository* would

have to include these fields (and more) if it were to inherit from the IMS Repository. The next figure shows a hierarchy of repositories. *My Repository* has inherited dictionaries and schema library from the IMS Repository. The organization responsible for *My Repository* may extend the dictionaries with additional fields. However, they are not allowed to define new fields for metadata that could be expressed with the parent's dictionaries (in this case the IMS Repository's).

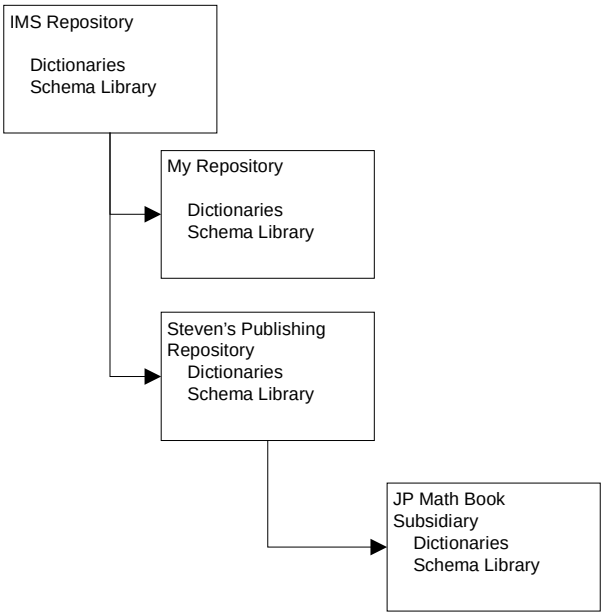


Figure 44: The Tiered Repository System. Each Repository inherits Dictionaries and Schema Library from its parent Repository. Usually a company or organization is responsible for the maintenance of a Repository. The inheritance of dictionaries and schemas ensures some level of interoperability. A metadata tool capable of parsing IMS metadata will at least understand the parts that e.g. My Repository has inherited from the IMS Repository.

THE IMS MASTER SCHEMA

When we extend schemas with inheritance, adding additional fields, we must still follow the structure imposed by the master schema. The master schema structure was not shown in our simple example. It is needed because fields from the dictionary may have different semantics depending on *where* (in what context) they appear in the master schema.

"The Schema Table uses a *context notation* to indicate that one element is contained under another element." [IMSMS]

The following is an extract from the IMS Master Schema:

```
General
   Identifier
   Title
Characteristics
   Language
   Description

<…cut…>

   Discipline
      Description

<…cut…>
```

The field *Description* is defined by its context. In this extract *Description* appears in the context of *Characteristics* as will as *Discipline*. If we inherit from a schema (e.g. *My Book Schema)* and add new fields that exist in the IMS Master Schema, we must follow the structure imposed by this schema. This makes it possible for parsers that know about the IMS Master Schema to understand *My Book Schema*. We are allowed to use fields from our own dictionaries as well, but they must be other than those in the IMS Master Schema.

THE IMS SCHEMA LIBRARY

The Schema Library in the IMS repository currently contains 5 schema (4 of which are called *Types*). They are *IMS Meta-Data Master Schema, IMS Base Type, IMS Item Type, IMS Module Type* and *IMS Tool Type*. The IMS Base Type contain the minimum required (marked as *mandatory*) in the master schema. The base schema is not used directly but is inherited by the Item, Module and Tool types.

IMS ITEM TYPE

"The IMS Item Type is intended to describe a single element such as a picture, an audio clip, a video clip, a text block, or HTML file. An item can contain more than one file (e.g. multiple images), but is not intended to hold contents of a complete educational nature." [IMSSETS]

IMS MODULE TYPE

"The IMS Module Type of metadata describes learning resources with a particular educational value or purpose. A module may consist of a single or multiple elements or learning resources. Examples of a module include a course, a topic, an assessment, an assignment or an activity." [IMSSETS]

IMS TOOL TYPE

The IMS Tool Type of metadata describes a learning resource that provides a function for the user. Examples of a tool include a word processor, calculator, statistical analysis package, or composition guide. A tool may be domain or task specific or the tool may serve a general purpose. [IMSSETS]

REFERENCES

[IMSDIC] Wason, T., IMS Metadata Dictionary, University of North Carolina, 3-Apr-1998, http://www.imsproject.org/technical/Metadata/DID170.html

[IMSMETA] Educom: IMS Metadata Specification, http://www.imsproject.org/md_overview.html

[IMSREQ] Educom: Educom's IMS Design Requirements, 19-Dec-1997, http://www.imsproject.org/reqv2/index.html

[IMSMS] Wason, T., IMS Meta-Data Master Schema, University of North Carolina, 22-Jul-1998, http://www.imsproject.org/technical/Metadata/library/DID173.html

[IMSSPEC] Educom: EDUCOM/NLII Instructional Management Systems Specification Document, Version 0.5, 29-Apr-1998, http://www.imsproject.org/specs/spec7.pdf

[IMSSETS] IMS Metadata Set v 1.1.0, University of North Carolina, http://www.imsproject.org/metadata_sets.html

[TIERS] Wason, T., TIERS Architecture, University of North Carolina, 16-Nov-1997, http://www.imsproject.org/technical/Metadata/Tiers/did162.html

APPENDIX F. SOFTWARE TOOLS

SIRPAC (W3C)

The SiRPAC homepage at W3C gives the following introduction: "This program compiles RDF/XML documents into the 3-tuples of the corresponding RDF data model. The documents can reside on local file system or at a URI on the Web. Also, the parser can be configured to automatically fetch corresponding RDF schemas from the declared namespaces. This version is suitable for embedded use as well as command line use. SiRPAC builds on top of the Simple API to XML documents (SAX)."

Try it out at: http://www.w3.org/RDF/Implementations/SiRPAC/

XMLFORJAVA (IBM)

IBM offers a free *validating* parser for XML written in 100% pure Java. Validating means that it checks the XML document against the DTD and reports any errors. Many other parsers currently don't read the DTD.

The parser comes with full source code and may be downloaded from:

http://www.alphaworks.ibm.com/formula.nsf/toolpreview/7BC35F3E4E69996A882565A700035C56

RDFFORXML (IBM)

RDF for XML is an RDF processor written in Java for building, querying, and manipulating RDF structures and reading and writing them in XML forms. The current implementation conforms to the working draft dated 10/8/98 of the RDF Syntax and Model working group of the W3C.

A free download is available at:

http://www.alphaworks.ibm.com/formula.nsf/alpharequirements/28EFAC36442F1AD8882565E10059AE39#Install

INTERNET EXPLORER 5 (MICROSOFT)

The newest version of Internet Explorer (currently the second beta release) contains an XML parser, an XSL processor and a DCD schema validator.

Microsoft will be happy to tell you more at http://www.microsoft.com

COMMUNICATOR 4.5 (NETSCAPE)

Communicator's "What's Related" feature send RDF descriptions back to Netscape's servers to track users movements on the net.

http://www.netscape.com

XML STYLER (ARBORTEXT)

XML Styler is a tool for creating XSL stylesheets. It can be freely downloaded from http://www.arbortext.com/xmlstyler/index.htm.