RDF data specifications: the simple alternative to complex data standards

Version 1.2, July 24, 2006

The current version of this file is:
http://www.pathologyinformatics.org/Resources/jjb_gwm.pdf

Prepared for APIII
August 16-18, 2006
Vancouver, British Columbia

Jules J. Berman, Ph.D., M.D.
President, Association for Pathology Informatics (2006)
Co-chair, Laboratory Digital Imaging Project
jjberman@alum.mit.edu

G. William Moore, M.D., Ph.D.
Department of Pathology
VA Medical Center
Baltimore, Maryland
George.Moore4@va.gov

key words: ldip, laboratory digital imaging project, rdf, resource description framework, rdf schema, iso11179, iso 11179, xml, xsd, cde, common data element, specification, data standard, ontology, ontologies, semantic web, use-case, metadata, binary image file, image, xml schema, class, classes, property, properties, domain, range, pattern, regex, regular expression, regular expressions, urn, uri, url, lsid, oid, datatype, datatyping, data type, jules berman, jj berman, Bill Moore, pathology, photomicrograph, histology, image specification

Overview of manuscript

The Resource Description Framework (RDF) provides a simple method for specifying information as data triples. The authors believe that much of the time and expense associated with developing and deploying data standards can be eliminated by a consistent implementation of recommended RDF data specification practices.

Necessary background subjects:

1. Meaning in informatics
2. Triples
3. Identifiers
4. Datatyping
5. Classes and Properties
6. Instantiating Classes

Necessary informatics techniques:

1. RDF syntax (specifying data as class instance-property-data triples)
2. RDF schema (formal dictionary for classes and properties)
3. XSD (to constrain data to a defined datatype)

The only implementation tools you really need is your head and a text editor such as notepad or emacs. Optional tools may include:

1. Programming modules for parsing XML and RDF (available soon as short Perl scripts distributed with this manuscript).
2. Publicly available RDF schemas

This manuscript provides a simplified review of the principles and practices of RDF data specification. A use-case from the Laboratory Digital Imaging Project (LDIP), involving the annotation of pathology images, is described.

The authors hope that readers will be able to adopt these principles and methods to specify their own datasets in an efficient and simple manner that supports data integration and software interoperability,

Background

The definition of Meaning

In informatics, assertions have meaning whenever a pair of metadata and data (the descriptor for the data and the data itself) is assigned to a specific subject.

Triples consist of: **Specified subject** then **metadata** then **data**

Some triples found in a medical dataset

"Jules Berman" "blood glucose level" "85"
"Mary Smith" "blood glucose level" "90"
"Samuel Rice" "blood glucose level" "200"
"Jules Berman" "eye color" "brown"

"Mary Smith" "eye color" "blue"
"Samuel Rice" "eye color" "green"

Some triples found in a haberdasher's dataset

"Juan Valdez" "hat size" "8"
"Jules Berman" "hat size" "9"
"Homer Simpson" "hat size" "9"
"Homer Simpson" "hat_type" "bowler"

Triples collected from both datasets whose subject is "Jules Berman"

"Jules Berman" "blood glucose level" "85"
"Jules Berman" "eye color" "brown"
"Jules Berman" "hat size" "9"

Triples can port their meaning between different databases because they bind described data to a specified subject. This supports data integration of heterogeneous data and facilitates the design of software agents. A software agent, as used here, is a program that can interrogate multiple RDF documents on the web, initiating its own actions based on inferences yielded from retrieved triples.

RDF (Resource Description Framework) is a syntax for writing computer-parsable triples. For RDF to serve as a general method for describing data objects, we need to answer the following four questions:.

1. How does the triple convey the unique identity of its subject? In the triple, "Jules Berman" "blood glucose level" "85", The name "Jules Berman" is not unique and may apply to several different people.

2. How do we convey the meaning of metadata terms? Perhaps one person's definition of a metadata term is different from another person's. For example, is "hat size" the diameter of the hat, or the distance from ear to ear on the person who is intended to wear the hat, or a digit selected from a pre-defined scale?

3. How can we constrain the values described by metadata to a specific datatype? Can a person have an eye color of 8? Can a person have an eye color of "chartreuse"?

4. How can we indicate that a unique object is a member of a class and can be described by metadata shared by all the members of a class?

Much of the remainder of the background section will be devoted to answering these four questions.

Introduction to RDF syntax: RDF triples

RDF is a specialized XML syntax for creating computer-parsable files consisting of triples. The subject of the RDF triple is invoked with the rdf:about attribute. Following the subject is a metadata/data pair.

Let us create an RDF triple whose subject is the jpeg image file specified as: http://www.gwmoore.org/ldip/ldip2103.jpg. The metadata is <dc:title> and the data value is "Normal Lung".

```
<rdf:Description
   rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
   <dc:title>Normal Lung</dc:title>
 </rdf:Description>
```

An example of three triples is proper RDF syntax is:

```
<rdf:Description
   rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
   <dc:title>Normal Lung</dc:title>
 </rdf:Description>
<rdf:Description
   rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
   <dc:creator>Bill Moore</dc:creator>
 </rdf:Description>
<rdf:Description
   rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
   <dc:date>2006-06-28</dc:date>
 </rdf:Description>
```

RDF permits you to collapse multiple triples that apply to a single subject. The following RDF:Description statement is equivalent to the three prior triples:

```
<rdf:Description
   rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
   <dc:title>Normal Lung</dc:title>
   <dc:creator>Bill Moore</dc:creator>
   <dc:date>2006-06-28</dc:date>
 </rdf:Description>
```

An example of a short but well-formed RDF image specification document is:

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description
   rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
   <dc:title>Normal Lung</dc:title>
   <dc:creator>Bill Moore</dc:creator>
   <dc:date>2006-06-28</dc:date>
  </rdf:Description>
</rdf:RDF>
```

The first line tells you that the document is XML. The second line tells you that the XML document is an RDF resource. The third and fourth lines are the namespace documents that are referenced within the document (more about this later). Following that is the RDF statement that we have already seen.

Believe it or not, the manuscript thus far covers 95% of what you need to know to specify your data with RDF. If you seek the simplest approach to specifying data as RDF documents, you can skip down to the section titled, "Use-case examples" and go to the first subsection, "RDF file with pointers to jpeg image file." This is the simplest use-case for specifying images. By modifying this use-case for your own purposes, you can write RDF documents that adequately specify almost any biomedical data.

Common Data Elements (CDEs)

The term, "common data element" is a misnomer. Most people, when they first encounter this term, assume that a data element holds data. Actually, a common data element is the metadata that describes a datum in a data record. In XML parlance, a CDE is an XML tag. The thing that makes a descriptor "common" is its common usage by a scientific community. The way that CDEs are intended to work is that a scientific community creates a list of CDEs that describe the kinds of data that their members use. The members of the community will all use the same CDEs (XML tags) to annotate their data files.

ISO-11179 Specification for CDEs

One of the most calamitous errors in any CDE project is to assume that everyone who reads a metadata tag will automatically understand its intended meaning. ISO-11179 is a standard way of defining CDEs with the necessary information for understanding their meanings.

The most popular CDEs in existence are the Dublin Core CDEs. These are a set of file descriptors that were prepared by a committee of librarians who convened in Dublin, Ohio. The Dublin Core includes basic information about electronic documents, such as: the title of the document, the name of the person who created the file, the date that the file was created, the date that the file was modified, a short description of the file. These are basically the items that a library software agent would need to retrieve if it were building an index of internet documents. The world of informatics would be a better place if everyone who created an HTML, XML or RDF file would remember to include the Dublin Core CDEs.

These Dublin Core CDEs have been prepared to comply with the ISO-11179 specification. Every effort to create a data specification for a knowledge domain should begin with by collecting the common data elements for the domain and annotating each element with the ISO-11179 CDE descriptors. The ISO-11179 descriptors for two of the Dublin Core CDEs (Title and Creator) are shown below.

From: http://dublincore.org/documents/1999/07/02/dces/

Title

Identifier: Title
Version: 1.1
Registration Authority: Dublin Core Metadata Initiative
Language: en
Obligation: Optional
Datatype: Character String
Maximum Occurrence: Unlimited
Definition: A name given to the resource.
Comment: Typically, a Title will be a name by which the resource is

formally known.

Creator

Identifier: Creator
Version: 1.1
Registration Authority: Dublin Core Metadata Initiative
Language: en
Obligation: Optional
Datatype: Character String
Maximum Occurrence: Unlimited
Definition: An entity primarily responsible for making the content of the resource.
Comment: Examples of a Creator include a person, an organisation, or a service. Typically, the name of a Creator should be used to indicate the entity.

RDF Schemas

An RDF schema is a dictionary file that lists the classes and the properties that pertain to RDF documents. In fact, the official long name for RDF Schema is the RDF Vocabulary Description Language. The classes of an RDF schema are formal definitions for the kinds of subjects that are found in the RDF triples. The properties of an RDF schema are the types of metadata descriptors for the data of the RDF triples. Elements in RDF schemas may be subclasses of elements in other RDF schemas.

Things to remember about RDF Schemas

1. RDF Schemas are written in XML but are completely unlike XML Schemas.

2. RDF Schemas contain declarations of the classes and properties that are used in RDF documents.

3. RDF Schemas, like all RDF documents, have no pre-determined order or composition and consist of statements expressed as triples. The subject of every triple in an RDF Schema will be either Class or Property.

4, Every RDF Schema can be thought of as a child of the W3C RDF Schema that defines the "super" classes Resource, Class and Property. All RDF Schemas will refer to the document that defines RDF syntax and to the document that defines the top-level schema, and therefore will begin something like this:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
  <rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
```

5. A typical RDF document consists of triples (subject, metadata, value). RDF documents usually reference one or more RDF Schemas to instantiate the subject of each triple (i.e., to tell us which class in an RDF schema the subject is an instance of) and to provide subjects with class-appropriate metadata.

6. Documents composed of triples whose components are defined by RDF Schemas can be used to completely specify data objects within a knowledge domain.

7. By completely specifying data objects in a knowledge domain, RDF specifications achieve the functionality of data standards.

In a later section, we will demonstrate the simple method for instantiating classes and for associating class instances with appropriate properties and with datatyped values.

A template for CDEs that can be transformed to RDF Schema elements

Here is a CDE template that satisfies the minimal recommendations for a CDE under ISO-11179 and that that provides all the information for a class or a property in an RDF schema.

The general format for class elements is:

```
Class Label (in standard XML tag format, uppercase 1st letter):
Registration Authority: Association for Pathology Informatics
Obligation: optional
Maximum Occurrence: Unlimited
Comment(must include detailed definition):
subClassOf:
Contributor (your consistent first-name last-name):
Date of your contribution:
```

The general format for property elements is:

```
Property Label (in standard XML tag format, lowercase 1st letter):
Registration Authority: Association for Pathology Informatics
Obligation: optional
Maximum Occurrence: Unlimited
Datatype (can be "Literal", a list, or a regex; default is "Literal"):
Comment(must include detailed definition):
Domain (comma-delimited if multiple):
Range (usually "Literal"):
Contributor (your consistent first-name last-name):
Date of your contribution:
```

The category "Obligation" should contain the word "required" or the word "optional". For the kinds of specifications discussed in this manuscript, including any CDE would always be optional. Similarly, for "Maximum Occurrence", we would think any CDE could occur an unlimited number of times in an RDF document.

Classes and Properties

Here is an example of an ISO-11179-compliant CDE written for a class named "Reagent".

```
Class Label:Reagent
versionInfo (required): 0.1
Registration Authority: Association for Pathology Informatics
```

Obligation:optional
Maximum Occurrence: Unlimited
Datatype: Literal
comment: Histologic_stain_reagents, tissue_fixation_reagents, and other chemicals employed in the laboratory. For example: distilled_water, ethanol, hematoxylin, aluminum_sulphate
subClassOf:Class
Contributor:Bill Moore
Date_of_contribution:05-30-2006

Once we have the CDE, it is a straightforward job to create an RDF Schema Class element:

```
<rdf:Class rdf:about="http://www.ldip.org/ldip_sch#Reagent">
<rdfs:label>Reagent</rdfs:label>
<rdfs:comment>
Histologic_stain_reagents, tissue_fixation_reagents, and other
chemicals employed in the laboratory. For example: distilled_water,
ethanol, hematoxylin, aluminum_sulphate
</rdfs:comment>
<rdfs:subClassOf
rdf:resource="xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#Class"/>
</rdf:Class>
```

A few points bear explanation. **All the information needed to generate an RDF class or property should be contained in an ISO-11179-compliant list of CDEs. An RDF Schema may consist purely of classes and properties.** Classes are defined in RDF exclusively through their ancestral relation. Basically, to build a class in RDF Schema, you announce that the element is a Class, you provide a unique locator (such as a URL) or a unique universally understood descriptor (more on this later) for the element, a description of the element, and the name of the father class of the element.

That's all there is to do for classes. You don't need to list the subclasses of the class because the subclasses will list the class as their father in their own schema entry. You don't need to list the properties of the class because the properties will list the classes whose data they describe.

Do classes in RDF schema remind you of anything? The classes in an RDF schema comprise an ontology. An ontology is a list of classes and their relationships You can think of an ontology as the "classy" half of an RDF schema. Classes become most useful when they have Properties (the other half of the RDF schema)..

Creating Properties

A property is a metadata element that is used to describe the data assigned to one or more class objects. Here is the CDE for a property named "dateTime".

Identifier:ldip:dateTime
Property Label:dateTime
versionInfo: 0.1
Registration Authority: Association for Pathology Informatics
Language:en

Obligation:optional
Maximum Occurrence: Unlimited
Datatype: /[\+\-]{1}[\d]{8}\.[\d]{6}Z[\+\-]{1}[\d]{4}/
comment: ISO 8601 format of data and time.
domain:Event
range: http://www.ldip.org/ldip_xsd.xsd#iso8601
Contributor:Bill Moore
Date_of_contribution:05-30-2006

An RDF Schema declaration for the dateTime property might be:

```
<rdf:Property rdf:about="http://www.ldip.org/ldip_sch#dateTime">
  <rdfs:label>dateTime</label>
  <rdfs:comment>
   The date and time at which an event occurs, in ISO8601 format
  </rdfs:comment>
  <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#Event"/>
  <rdfs:range
rdf:resource="http://www.ldip.org/ldip_xsd.xsd#iso8601"/>
</rdf:Property>
```

Let's look at the Time property. The first line announces that we will be declaring a Property. The second line tells us the name of the Property (Time) and it's URL (the current RDF schema document). The third line provides the label by which we will refer to the Property. This might come in handy if we had different names for the property in different languages. The comment includes a definition for the element.

The next line specifies the domain (class) for the property. The domain of a property is the class for which the property may be used. In this case, the domain class for which the dateTime property applies is Event. This makes sense. If you need to describe an event, you would want to include the time that the event occurred.

A property for a class may serve as a property for all of the subclasses of the class (because all the subclass instances are members of the ancestor class). Every Property must have a domain (a class or classes for which the Property may be used) and a Range (a specified kind of data that is described by the Property). A property may have multiple classes in its domain. **When a property has multiple classes in its domain, all the classes in the domain share the same property (obviously). This achieves some of the functionality of multi-class inheritance without actually needing to instantiate multiple classes under a single object.** This is a subtle concept, and does not need to be mastered at this time. **Suffice it to say that as you create your own RDF Schemas, you should try to design your Properties to apply to multiple classes, and you should try to instantiate objects under a single class.**

Specifying a datatype from within an RDF Schema Property Element

Let us continue to examine the dataTime property. Recall that a triple consists of a subject followed by metadata (the property element) followed by the data. The property element describes the data. The range of the property element tells us what kind of data is described. In RDF schemas, the range of a property is often "Literal" an element defined in the RDF syntax document that refers to any character string.

You can see immediately that describing the range of a property as a character string does little to constrain or structure the expected values for a data element.

In the dateTime property, we want the range of the property to be data that conforms to the ISO8601 date/time format. How do we convey the datatype of the data/time element in RDF?

RDF has no intrinsic datatyping facility. So for our property range, we provide a resource (URL) that specifies an element in an .xsd file that defines the datatype we need.

The range for the dateTime property is a resource:

```
<rdfs:range
rdf:resource="http://www.ldip.org/ldip_xsd.xsd#iso8601"/>
```

The resource points us to an xsd file on the web, and to a particular element within the xsd file, labeled iso8601. Let's pretend we visit the file and extract the iso8601 element. We might find the following:

```
<simpleType name='iso8601'>
<!-- values of a data_time must contain          -->
<!-- a plus or minus sign occurring zero or one  -->
<!-- times followed by 8 digits                  -->
<!-- followed by a perios                        -->
<!-- followed by 6 digits                        -->
<!-- followed by the a letter Z, T or a space    -->
<!-- followed by a plus or minus sign occurring  -->
<!-- zero or one time, followed by 4 digits      -->
  <xsd:restriction base='string'>
    <pattern value=''[\+\-]?[\d]{8}\.[\d]{6}[ZT ][\+\-]{1}[\d]{4}"/>
  </xsd:restriction>
</simpleType>
```

The essence of the datatype is found in the pattern value line:

```
<pattern value=''[\+\-]?[\d]{8}\.[\d]{6}[ZT ][\+\-]{1}[\d]{4}"/>
```

This line uses a Regular Expression (RegEx) that provides a pattern to which the element must conform. RegEx is beyond the scope of this manuscript.

Don't be intimidated by .xsd and Regex rules. For most purposes, simply describing the range of a property with the RDF syntax-defined element, "Literal" will be all that you need.

The .xsd element definition imposes a datatype pattern on the value of the data described by the property. A validating software agent would check an RDF document to determine if the data described by a property conforms to the range of the property element defined by the element in the .xsd resource for the the property range.

If we want to employ this trick, we'll need to prepare an XSD file that contains elements for all the datatypes referred under the property ranges included in our XML Schema.

Creating the external XSD document to datatype our property ranges

XSD datatype files are very easy to prepare. Basically, you just list your datatypes and provide descriptors. The following generic file contains samples of the kinds of datatypes you will probably need (patterns, inclusive values, and unions).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#">

<simpleType name='sp_pattern'>
<!-- values of an accession number must contain -->
<!-- the letters sp followed by a hyphen followed   -->
<!-- by two digits followed by a hyphen            -->
<!-- followed by any number of digits              -->
  <xsd:restriction base='string'>
    <pattern value='sp\-[0-9]{2}\-[0-9]+'/>
  </xsd:restriction>
</simpleType>


<xsd:simpleType name="adult_age">
<!-- an adult is at least 18 years old             -->
  <xsd:restriction base="xsd:positiveInteger">
    <xsd:minInclusive value="18"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="serialNumber">
  <!--  may be either integers or mixed alphanumeric strings -->
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base='integer'/>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base='string'/>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="EnumerationObjectives">
<!--  may be either integers or mixed alphanumeric strings -->
  <xsd:restriction base="string">
    <xsd:enumeration value="2.5x"/>
    <xsd:enumeration value="6.3x"/>
    <xsd:enumeration value="20x"/>
    <xsd:enumeration value="40x"/>
    <xsd:enumeration value="100x"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

The differences between Classes and Properties

The most difficult step in building any schema is determining whether a candidate element is a Class or a Property. Generalizations do not hold for all cases. For example, Classes tend to be nouns, while Properties (that describe data) tend to be adjectives. However, a Property can be a noun (e.g. Time) if it's role is to describe a data value (4:00 PM EST). Furthermore, we sometimes assign active processes to classes (e.g. birth, death), and we cannot assume that classes are always static objects.

There is a strong tendency to assign subclass status to things that are not examples of their ancestral class. For instance, if Person is a class, someone may think that Leg is a subclass of Person (because a Leg is in a class of things that are parts of a Person). No! Leg is never a subclass of Person because a Leg is not a Person. A subclass of Person must be composed of types of Persons. So, Patient is a subclass of Person, and Pathologist is a subclass of Person, because they are both examples of Persons and because there are instances of Patients and instances of Pathologists. Remember, a class is a construct whose chief job is to provide specified instances.

How about Friend? Is Friend a subclass of Person? Yes and no. Friend can be a subClass of person if you want to organize Persons based on whether they are Friends or not-Friends. However, if you think that being a friend is just one of many features of any Person, you would be much better off defining friend as an RDF property. The data-type of the friend property may be a Boolean (true or false).

Here are some general recommendations for distinguishing RDF Schema Classes and Properties.

1. If something has instances of itself, it is almost always a class.

2. If a candidate class is a subclass of more than one class lineages (so-called multiple inheritance), think very hard before making it a class. In most cases, you will be better off if it is assigned as a Property or if it is excluded from the RDF Schema.

3. Every class must be a subclass of a class. To be a subclass of a class the subclass must qualify as a member of the father class.

4. A class if fully specified when you know its definition and you know it's ancestor class.

5. Properties describe data. If something has a specific datatype that includes numerics, it is almost always a property.

Creating instances of classes

The purpose of a class is to support the creation of subclasses and class instances. If we have a Report class, we might also have a Surgical_Pathology_Report class which is a subClassOf Report. Elsewhere_General_S06_4352 may be one unique instance of the the class Surgical_Pathology_Report. As an instance of the class, the data in the report can be described using the properties specified in an RDF schema to have the Surgical_Pathology_Report domain.

The way to create an instance of a class in an RDF document is with the RDF "type" primitive.

```
<rdf:description
rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
  <rdf:type resource= "http://www.ldip.org/ldip_sch#Image">
</rdf:description>
```

Whenever we wish to create an instance of a class belonging to any RDF Schema we choose, we will add an RDF statement much like the one shown above. We begin by specifying the subject of the statement (with the "about" declaration), then we will indicate that "type" of the subject is the class listed in an RDF Schema. An object may be an instance of more than one class, and a proper RDF statement may list numerous type/class pairs, but we caution that doing so adds complexity to your document.

Preserving namespaces for Classes and Properties

One of the most important features of RDF Schemas is that you can mix and match different elements (classes and properties) from different schemas in a single document. This is done using a simple namespace notation that is common to all XML documents.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
 <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:dc="http://purl.org/dc/elements/1.1/#"
    xmlns:schem1="http://www.someplace.org/#"
    xmlns:schem2="http://www.someplace_else.com/#">

   <rdf:Description
rdf:about="http://www.gwmoore.org/ldip/ldip2201.jpg">
    <dc:creator>Bill Moore</dc:creator>
    <schem1:camera>yes</schem1:camera>
    <schem2:camera>Olympus</schem2:camera>
    <schem1:format>jpeg</schem1:format>
    <schem2:format>jpg</schem2:format>
   </rdf:Description>
 </rdf:RDF>
```

Notice that the elements camera and format appear twice, but on each occasion, they are prefixed by different namespaces (schem1 and schem2). The namespaces preserve metadata individuality.

Unique subject identifiers

Once you have made an instance of a class, you need to identify the instance uniquely. Failing this, the metadata-data pairs associated with a class instance has no meaning.

The typical unique subject identifier in an RDF triple is a URL specifying a unique web location for a data object.

Failing this, any unique identifier that permanently, unmistakably and uniquely links an object to a character string will suffice.

There are a number of registry services that provide identifiers for data objects in their domains. Examples are:

DOI, Digital object identifier.
PMID, PubMed identification number.
LSID (Life Science Identifier).
HL7 OID (Health Level 7 Object Identifier).
DICOM (Digital Imaging and Communications in Medicine) identifiers.
ISSN (International Standard Serial Numbers).
Social Security Numbers (for U.S. population).
NPI, National Provider Identifier, for physicians.
Clinical Trials Protocol Registration System.
Office of Human Research Protections FederalWide Assurance number.
Data Universal Numbering System (DUNS) number
DNS, Domain Name Service.

In the life sciences, the lsid number has achieved some popularity. The lsid resolution protocol has five parts:

Network Identifier (NID)
root DNS name of the issuing authority
namespace chosen by the issuing authority
object id unique in that namespace and assigned locally
revision id for storing versioning information (optional)

LSIDs can be used as URN's that uniquely identify items in RDF statements.

LSID Examples:

urn:lsid:pdb.org:1AFT:1
This is the first version of the 1AFT protein in the Protein Data Bank.

urn:lsid:ncbi.nlm.nih.gov:pubmed:12571434
This references a PubMed article.

urn:lsid:ncbi.nlm.nig.gov:GenBank:T48601:2
This refers to the second version of an entry in GenBank

HL7 also provides unique identifiers. An enterprise can obtain an OID at:
http://www.iana.org/cgi-bin/enterprise.pl

For example, the University of Michigan OID is:

1.3.6.1.4.1.250

The enterprise OID serves as a prefix for unique data objects within an institution.

Unique identifiers are used to uniquely specify the subject of a triple (i.e. to specify what a triple is about).

Example:

```
<rdf:description
rdf:about="urn:lsid:ncbi.nlm.nih.gov:pubmed:8718907">
<dc:creator>Bill Moore</dc:creator>
</rdf:description>
```

Here we have a unique data object specified with an lsid for a PubMed citation. The number 8718907 is the unique pubmed citation number. We add a property/value pair consisting of the Dublin Core creator element and the data value, "Bill Moore". Once we have a unique subject, we can instantiate the element for an appropriate class.

```
<rdf:description
rdf:about="urn:lsid:ncbi.nlm.nih.gov:pubmed:8718907">
<rdf:type resource= "http://www.ldip.org/ldip_sch#Document"/>
<dc:creator>Bill Moore</dc:creator>
  </rdf:description>.
```

Here we have a unique data object, instantiated as a member of the Document class. The Document class is defined in an RDF Schema referenced to a URL.

To summarize, the subject of a triple needs to be identified. The subject of a triple can be in the form of a URL (complete web address) or a URN (Unique Resource Name). URLs and URNs are both both of which are forms of URIs (Unique Resource Identifiers).

You can create your own uniquely specified data object by appending a unique number to a URN prefix. For instance, a surgical pathology report or a patient name or an image file can be the subject of a triple if it is identified by the following

urn:www.ldip.org:ldip:4Ib30fk6J3Y9gWpwMV27

Here, the prefix is "urn:www.dlip.org:ldip:" An alphanumeric suffix, "4Ib30fk6J3Y9gWpwMV27" is a 20 character random string that we have chosen for the object.

There are many ways of providing identifiers for subjects. Once a subject is identified, triples containing the identifier can be merged from multiple RDF documents appearing anywhere on the internet.

Use-case: the Laboratory Digital Imaging Project

The Laboratory Digital Imaging Project (LDIP) is sponsored by the Association for Pathology Informatics and co-chaired by Jules J. Berman and Ulysses Balis. LDIP was developed in 2004 under the direction of the Association for Pathology Informatics (API), a division of the American Society for Investigative Pathology (ASIP) to develop and promote the use of pathology images to improve patient care and improve the quality of pathology research.

Currently, LDIP's efforts are focused on establishing a free, open access, voluntary specification that will permit images generated in pathology and in clinical laboratories to be widely shared across different applications, databases, and operating

systems for the purpose of enhancing image annotation, integration, archiving, publication, and analysis. The specification is being written as an RDF (Resource Description Framework) schema that can be incorporated into XML documents and that contains sufficient self-description to facilitate data integration.

The LDIP committee is composed of members of API and corporate sponsors. All of the documents produced by the LDIP committee are freely available to the public and can be downloaded from the LDIP web site (www.ldip.org)

Format for LDIP CDEs

In a prior section, we looked at the general format for CDEs in LDIP. The (unofficial) current list of ldip CDEs is at:

http://www.gwmoore.org/ldip/scderdfh.htm

For space considerations, we won't show you the entire LDIP CDE file. Some of the excerpted CDEs are:

```
Identifier:ldip:Person
Class Label:Person
versionInfo: 0.1
Registration Authority: Association for Pathology Informatics
Language:en
Datatype: Literal
comment: All persons involved in the patient's care, including
  the patient him/herself and all providers and collections of providers.
subClassOf:Class
Contributor:Bill Moore
Date_of_contribution:05-30-2006


Identifier:ldip:Reagent
Class Label:Reagent
VersionInfo: 0.1
Registration Authority: Association for Pathology Informatics
Language:en
Datatype: Literal
Comment: Histologic_stain_reagents, tissue_fixation_reagents, and
other chemicals  employed in the laboratory.  For example:
distilled_water, ethanol,  hematoxylin, aluminum_sulphate, and so on.
SubClassOf:Class
Contributor:Bill Moore
Date_of_contribution:05-30-2006

Identifier:ldip:Instrument
Class Label:Instrument
VersionInfo: 0.1
Registration Authority: Association for Pathology Informatics
Language:en
Datatype: Literal
Comment: All the instruments used in preparing, viewing,
  and imaging a specimen. Includes: microscope, camera.
SubClassOf:Class
Contributor:Bill Moore
Date_of_contribution:05-30-2006
```

Identifier:ldip:stain
Property Label:stain
versionInfo: 0.1
Registration Authority (required): Association for Pathology Informatics
Language:en
Datatype (required, can be regex, URL, or literal): literal
comment: For example, Unstained, Hematoxylin and eosin, Congo Red, Masson, Periodic acid Schiff,....
domain:Reagent, Image
range (required): Literal
subPropertyOf:
Contributor:Bill Moore
Date_of_contribution:05-30-2006

LDIP RDF Schema

An RDF Schema can be automatically generated from the CDE. The LDIP schema has 7 top-level classes:

1. Reagent (e.g., stains, monoclonal antibodies)
2. Person (e.g., patient, pathologist, nurse, physical therapist)
3. Data_Object (e.g., report, image, protocol, electronic medical record)
4. Specimen (e.g., body fluid, biopsy)
5. Instrument (e.g., microscope, camera)
6. Term (e.g., diagnosis, stage, prognosis)
7. Event (e.g., birth, death, clinic_visit)

Every class in the LDIP RDF Schema is either one of these seven classes or a descendant of one of these seven classes.

A mini-RDF schema composed from some unofficial LDP CDEs is shown:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
<rdf:Class rdf:about="http://www.ldip.org/ldip_sch#Image">
  <rdfs:label>Image</rdfs:label>
    <rdfs:comment>
      A digital image file, such as a jpeg, tiff or DICOM file
    </rdfs:comment>
    <rdfs:subClassOf rdfs:resource="http://www.w3.org/2000/01/rdf-
schema#Class"/>
</rdf:Class>
```

```
<rdf:Property rdf:about="http://www.ldip.org/ldip_sch#fileName">
  <rdfs:label>fileName</rdfs:label>
  <rdfs:comment>
      A file name, can include a directory path in DOS or
      unix notation.
  </rdfs:comment>
  <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#report"/>
  <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#image"/>
  <rdfs:range rdf:resource="http://www.ldip.org/ldip_xsd.xsd#filepath"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.ldip.org/ldip_sch#lsid">
    <rdfs:label>lsid</rdfs:label>
    <rdfs:comment>
    The life sciences identifier (lsid)
    </rdfs:comment>
  <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#report"/>
   <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#image"/>
    <rdfs:range rdf:resource="http://www.ldip.org/ldip_xsd.xsd#lsid"/>
</rdf:Property>


<rdf:Property rdf:about="http://www.ldip.org/ldip_sch#objective">
    <rdfs:label >objective</rdfs:label>
    <rdfs:comment>
      A description of the microscope's objective lens.
      Required elements include the lens numerical aperture, and the
magnification, both of which a floating point (real) numbers.
    </rdfs:comment>
    <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#Microscope"/>
    <rdfs:range
rdf:resource="http://www.ldip.org/ldip_xsd.xsd#EnumerationObjectives"/>
</rdf:Property>


<rdf:Property rdf:about="http://www.ldip.org/ldip_sch#model">
    <rdfs:label>model</rdfs:label>
    <rdfs:comment>
        the manufacturer's model name for an instrument
    </rdfs:comment>
    <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#Instrument"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.ldip.org/ldip_sch#serialNumber">
    <rdfs:label>serial_number</rdfs:label>
    <rdfs:comment>
        the manufacturer's unique serial number for an instrument
    </rdfs:comment>
    <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#Instrument"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2000/01/ldip_xsd.xsd#serialNumber"/>
</rdf:Property>
</rdf:RDF>
```

This RDF Schema lists one class (Image) and six properties (fileName, lsid, ldipIdentifier, objective, model and serialNumber. We will be using some of these in our use-case example.

Let us look at one of the properties, fileName

```
<rdf:Property rdf:about="http://www.ldip.org/ldip_sch#fileName">
  <rdfs:label>fileName</rdfs:label>
  <rdfs:comment>
      A file name, can include a directory path in DOS or
      unix notation.
  </rdfs:comment>
  <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#report"/>
  <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#image"/>
  <rdfs:range rdf:resource="http://www.ldip.org/ldip_xsd.xsd#filepath"/>
</rdf:Property>
```

Notice that there are two domains for the fileName property: Report and Image. This is allowable under the rules of RDF Schema. But is it necessary or wise two use two domains when one will suffice? Remember, both Report and Image are subclasses of the father class, Data_Object. The easiest way to organize the RDF Schema is to list Data_Object as the single domain for fileName.

```
<rdf:Property rdf:about="http://www.ldip.org/ldip_sch#fileName">
  <rdfs:label>fileName</rdfs:label>
  <rdfs:comment>
      A file name, can include a directory path in DOS or
      unix notation.
  </rdfs:comment>
  <rdfs:domain rdf:resource="http://www.ldip.org/ldip_sch#Data_Object"/>
<rdfs:range rdf:resource="http://www.ldip.org/ldip_xsd.xsd#filepath"/>
</rdf:Property>
```

There are a few things about the RDF Schema file worth rememberin

1. All the information needed to prepare the RDF Schema file was found in the ISO-11179 conformant CDE list.

2. RDF Schema, like all RDF documents, is an XML file

3. RDF Schema employs RDF syntax. It starts with the rdf description element and lists the namespaces used in the document. Then it lists triples for classes and properties. Each triple begins with an resource identifier for the subject of the triple (the name of the class or the name of the property) and is followed by simple metadata/data pairs

4. A property can have more than one domain

5. A property range can be another resource. We have seen that when the property range is an xsd file, we can write standard xsd datatypes for the expected value of the range.

The JPEG Images

We use the following two figures for this manuscript:

Figure 1 - Photomicrograph, normal human lung tissue, H&E, 10x, http://www.pathologyinformatics.org/Resources/ldip2103.jpg

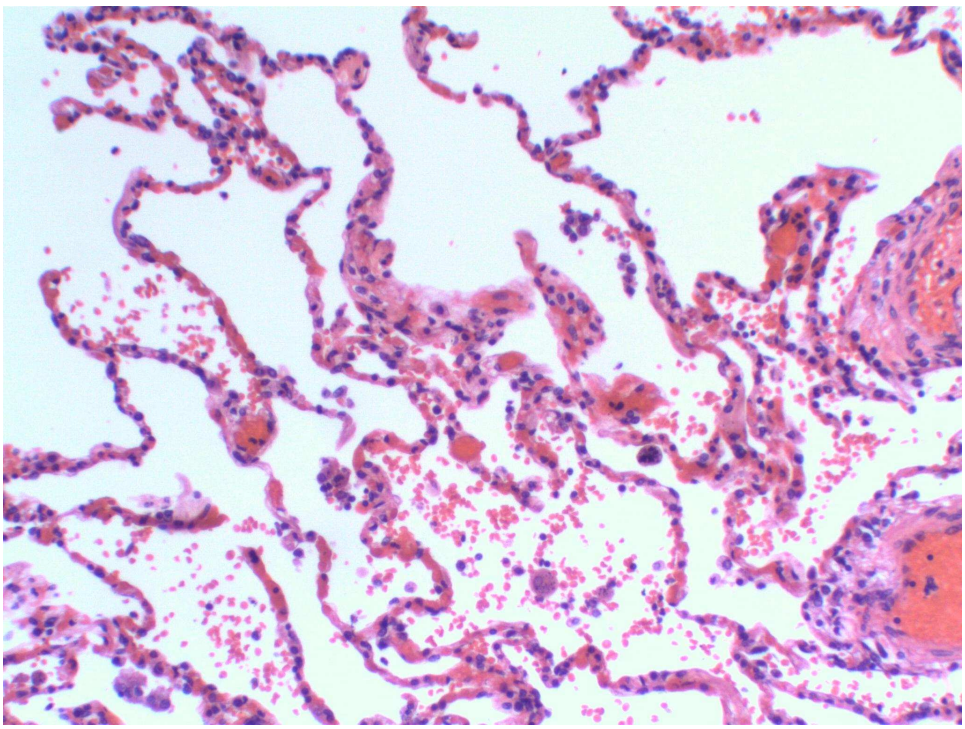
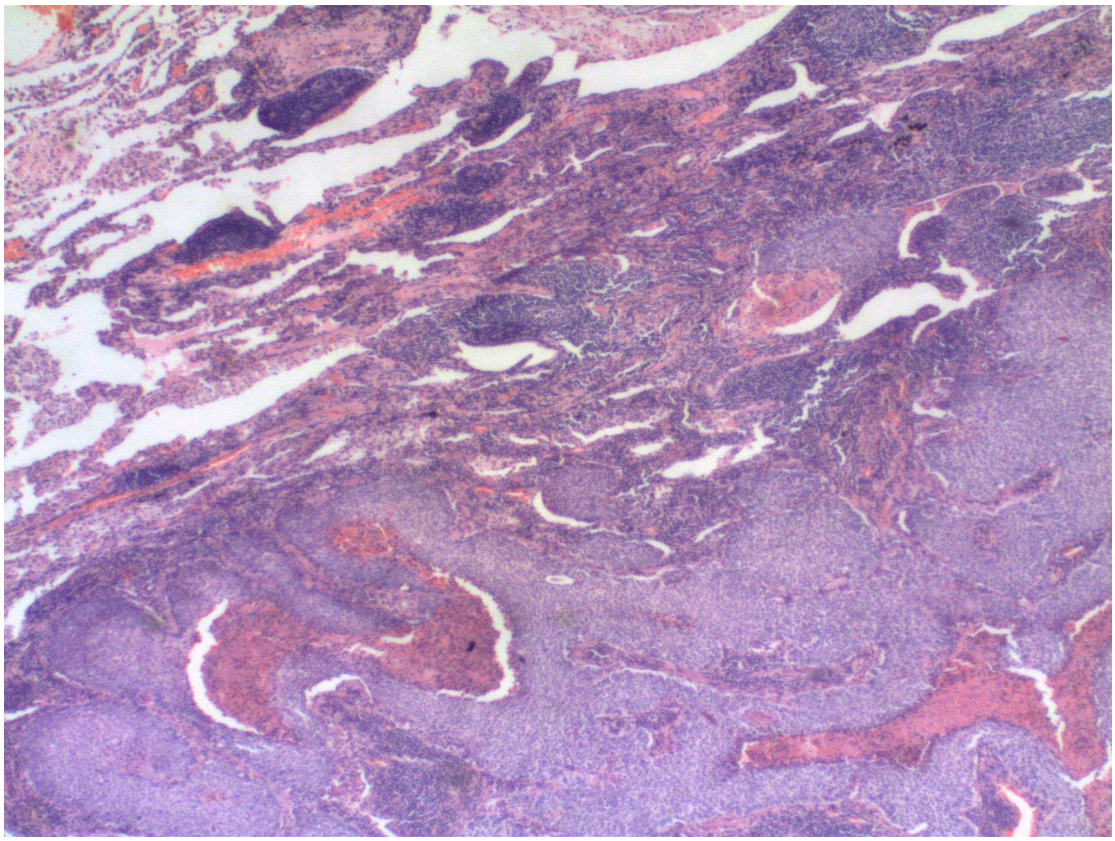


Figure 2 - Photomicrograph, squamous cell carcinoma of lung, H&E, 2.5x, http://www.pathologyinformatics.org/Resources/ldip2201.jpg



Sample textual annotation

Here is a sample annotation for a pathology photomicrograph. We will use this annotation to show how to make RDF triples from unstructured data. We will provide 5 different strategies for specifying an image object in conformance with the LDIP RDF Schema. and to an .xsd document, and five ways to use use RDF to specify an image object.

Unstructured annotation:

> On June 21, 2006, Bill Moore, M.D. took a photo of human lung tissue, using a histologic slide from an old collection of publicly distributed teaching slides of anonymous patients. He used an Olympus Model BH2 microscope, serial number 224085, under the 10x objective lens. The camera that took the digital image was an Infinity 3 CCD camera, Serial number 00169344. The produced image file is named ldip2103, is an

H&E stained photomicrograph of normal lung tissue and has the URL http://www.gwmoore.org/ldip/ldip2103.jpg.

We can create a structured annotation composed of a collection of metadata-value pairs describing the image object identified by its unique URL:

http://www.gwmoore.org/ldip/ldip2103.jpg

creator Bill Moore
date created June 21, 2006
microscope make Olympus
microscope model BH2
microscope serial number 224085
image type photomicrograph
camera make Infinity
camera model 3
camera type CCD
camera serial number 00169344
histologic stain H&E
organism human
tissue normal lung

Let us look at the annotation. Dr. Moore has used an anonymous slide with no provided clinical information. By omitting patient-related information, human subjects protection issues do not apply, and this manuscript can be distributed freely. Had there been clinical information included in the annotation, the process of creating an RDF specification for the image object would be much the same as the process as we shall describe. The purpose of this exercise is to show a minimal implementation of an RDF specification.

Let us look at the metadata value pairs. The metadata elements that we will need are: creator, date created, make (referring to microscope and to camera), model (referring to microscope and to camera), serial number (referring to microscope and to camera), type of image, histologic stain, organism and tissue. We should have an RDF Schema document that includes the name and definition for each of these metadata properties.

In the next sections, we will take the structured annotation (above) and transform it into an RDF specification for the image object. There are actually 5 ways we can do this.

Five ways of providing a photomicrograph with annotations

1. RDF document with pointer to image file
2. RDF document containing image binary converted to base64 ascii
3. Image file with RDF document inserted into image file header
4. Multiple RDF documents pointing to an image
5. Multiple RDF documents pointing to multiple images or multiple parts of a single image file

RDF document with pointer to image file

The simplest way to create an LDIP image specification is to prepare an RDF file containing triples that use LDIP Classes and Properties to annotate the image and to include one triple that points to the URL of the image binary.

The following example provides pathologists with the only method that most will ever use.

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ldip="http://www.ldip.org/ldip_sch#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">

 <rdf:Description rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
   <rdf:type rdf:resource= "http://www.ldip.org/ldip_sch#Image"/>
   <dc:title>Normal Lung</dc:title>
   <dc:creator>Bill Moore</dc:creator>
   <dc:date>2006-06-21</dc:date>
   <ldip:instrument_id
rdf:resource="urn:www.ldip.org:ldip:Olympus_BH2_224085"/>
   <ldip:instrument_id
rdf:resource="urn:www.ldip.org:ldip:Infinity_3_00169344"/>
   <ldip:imageType>photomicrograph</ldip:imageType>
   <ldip:stain>H and E</ldip:stain>
   <ldip:tissue>lung</ldip:tissue>
   <ldip:organism>human</ldip:organism>
   <ldip:objective>10x</ldip:objective>
   <ldip:diagnosis>normal</ldip:diagnosis>
</rdf:Description>

<rdf:Description rdf:about="urn:www.ldip.org:ldip:Olympus_BH2_224085">
<rdf:type rdf:resource="http://www.ldip.org/ldip_sch#Instrument"/>
<ldip:instrumentType>Microscope</ldip:instrumentType>
   <ldip:make>Olympus</ldip:make>
   <ldip:model>BH2</ldip:model>
   <ldip:serialNumber>224085</ldip:serialNumber>
</rdf:Description>

<rdf:Description rdf:about="urn:www.ldip.org:ldip:Infinity_3_00169344">
  <rdf:type resource= "http://www.ldip.org/ldip_sch#Instrument"/>
  <ldip:instrumentType>Camera</ldip:instrumentType>
  <ldip:make>Infinity</ldip:make>
  <ldip:model>3</ldip:model>
  <ldip:serialNumber>00169344</ldip:serialNumber>
</rdf:Description>
</rdf:RDF>
```

RDF document containing image binary converted to base64 ascii

Though we distinguish text files from binary files, all files are actually binary files. Sequential bytes of 8 bits are converted to ascii equivalents, and if the ascii equivalents are alphanumerics, we call the file a text file. If the ascii values of 8-bit sequential file chunks are non-alphanumeric, we call the files binary files.

Standard format image files are always binary files. Because RDF syntax is a pure ascii file format, image binaries cannot be directly pasted into an RDF document. However, binary files can be interconverted to an from ascii format, using a simple software utility. This simple Perl script, using the MIME::Base64::Perl module is all that is necessary to interconvert binary files to Base64.

```
#!/usr/bin/perl
use MIME::Base64::Perl;
open (TEXT,"c\:\\ftp\\ldip\\ldip2103\.jpg")||die"cannot"; #path to sample file
binmode TEXT;
$/ = undef;
$string = <TEXT>;
close TEXT;
$encoded = encode_base64($string);
open(OUT,">2103.txt");
print OUT $encoded;
close OUT;

#$decoded = decode_base64($encoded);
#open(OUT,">binary.jpg");
#binmode OUT;
#print OUT $decoded;
exit;
```

Here is an example of the same RDF document shown in the prior use-case. The only difference is that in addition to pointing to the URL that identifies the image, this document contains the image file converted to base64 ascii.

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ldip="http://www.ldip.org/ldip_sch#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">

 <rdf:Description rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
   <rdf:type rdf:resource= "http://www.ldip.org/ldip_sch#Image"/>
   <dc:title>Normal Lung</dc:title>
   <dc:creator>Bill Moore</dc:creator>
   <dc:date>2006-06-21</dc:date>
   <ldip:instrument_id
rdf:resource="urn:www.ldip.org:ldip:Olympus_BH2_224085"/>
   <ldip:instrument_id
rdf:resource="urn:www.ldip.org:ldip:Infinity_3_00169344"/>
   <ldip:imageType>photomicrograph</ldip:imageType>
   <ldip:stain>H and E</ldip:stain>
   <ldip:tissue>lung</ldip:tissue>
   <ldip:organism>human</ldip:organism>
   <ldip:objective>10x</ldip:objective>
   <ldip:diagnosis>normal</ldip:diagnosis>
   <ldip:base64File>
/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsL
DBkSEw8UHRofHh0aHBwgJC4nICIsIxwcKDcpLDAxNDQ0Hyc5PTgyPC4zNDL/2w
BDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyM
jIyMjIyMjIyMjIyMjL/wAARCAYACAADASIAAhEBAxEB/8QAHwAAAQUBAQEB
AQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQA
.
.
.
```

```
D1phhQnOMVSxL6nRHEaWZz8tpvGBWdPpMpbO08967AW8YOcU4xIRgqKr6yl0
N449x0S0OFOkv6Yq1a6W6HJHFdb9miznbStAhB4AzVfWl0KqY3nsjjNSuEtosE9B
XC6lrf7whT39a7jxLp0xjbah6V5leaZchySh6104PC+196R1zrU6cFZo/9k=
</ldip:base64File>
</rdf:Description>

<rdf:Description rdf:about="urn:www.ldip.org:ldip:Olympus_BH2_224085">
<rdf:type rdf:resource="http://www.ldip.org/ldip_sch#Instrument"/>
<ldip:instrumentType>Microscope</ldip:instrumentType>
   <ldip:make>Olympus</ldip:make>
   <ldip:model>BH2</ldip:model>
   <ldip:serialNumber>224085</ldip:serialNumber>
</rdf:Description>

<rdf:Description rdf:about="urn:www.ldip.org:ldip:Infinity_3_00169344">
  <rdf:type resource= "http://www.ldip.org/ldip_sch#Instrument"/>
  <ldip:instrumentType>Camera</ldip:instrumentType>
  <ldip:make>Infinity</ldip:make>
  <ldip:model>3</ldip:model>
  <ldip:serialNumber>00169344</ldip:serialNumber>
</rdf:Description>
</rdf:RDF>
```

We cut the bulk of the image file from the RDF document. A complete base64 image file can occupy hundreds or even thousands of pages of text. The base64 rendition of ldip2103.jpg has a length of 508,750. It is somewhat ironic that the only way of making an image file readable within an RDF document is to create a file that is too long to actually read.

Image file with RDF document inserted into image file header

Almost all popular image formats contain "header" sections that are not part of the actual image binary. The header sections contain information that is used by image viewing software to properly display the image. Robust imaging software applications are written with subroutines that parse through the different headers of images and extract information such as the height, width, pixel number, pixel size, pixel color, color map index, and so on. Some headers are extensible, allowing sofware to insert blocks of text into the header without changing the image binary.

It is easy to insert an RDF document into the header of a jpeg image file, and it is just as easy to extract the RDF triples. Here's how you do it:

1. Prepare your RDF document just as you would do in the earler use-case examples.
2. Use software that adds comments to the header of a jpeg file.
3. Use the new jpeg file (now with RDF comments) to display the image or to send to colleagues. When displayed, it will look exactly like the file before the contents of the RDF document were added.
4. Use software to extract the comments from the header of the jpeg file, as needed.

The fillowing Perl script will take the jpeg image ldip2103.jpg and add the RDF document from the first use-case to its header. Once created, the program extracts and displays the contents of the RDF file.

```
#!/usr/local/bin/perl
use Image::MetaData::JPEG;
```

```
my $filename = "ldip2103.jpg"; #comment:your filename here
my $file = new Image::MetaData::JPEG($filename);
die 'Error: ' . Image::MetaData::JPEG::Error() unless $file;
print "Description of JPEG file\n";
print $file->get_description();
print "\n\nRDF Annotations to JPEG file\n\n";
open (TEXT, "rdf_desc.xml")||die"cannot"; #comment:rdf document
$line = " ";
while ($line ne "")
  {
  $line = <TEXT>;
  $file->add_comment($line);
  }
unlink $filename;
$file->save($filename);
my $file = new Image::MetaData::JPEG($filename);
my @comments = $file->get_comments();
print join("",@comments);
exit;
```

This Perl script requires the freely available open source module Image::MetaData::JPEG. You can download this module from CPAN (Comprehensive Perl Archive Network, www.cpan.org).

Multiple RDF documents pointing to image file

There are time when the the structural data (i.e., the non-binary data) for a data object's specification must be distributed in multiple files.

This is one of the most important reasons for using a data specification, rather than a data standard. The specification permits you to create a dynamic object composed of informational pieces that can be updated, so that the content and value of a specified image object increases over time. A data standard obligates you to compose a static data file. If the standard data file contains information that cannot be shared (due to human subject risks, or to intellectual property encumbrances), the standard file usually cannot be distributed. A specification may consist of multiple files connected by URL pointers. If component files contain privileged information, the data object's specification can be distributed with access restricted to specified files.

RDF file 1 (Describes an image)

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ldip="http://www.ldip.org/ldip_sch#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">

 <rdf:Description rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
   <rdf:type rdf:resource= "http://www.ldip.org/ldip_sch#Image"/>
   <dc:title>Normal Lung</dc:title>
   <dc:creator>Bill Moore</dc:creator>
   <dc:date>2006-06-21</dc:date>
   <ldip:instrument_id
rdf:resource="urn:www.ldip.org:ldip:Olympus_BH2_224085"/>
   <ldip:linkedFile rdf:resource="http://www.ldip.org/file2">
```

```
   <ldip:instrument_id
rdf:resource="urn:www.ldip.org:ldip:Infinity_3_00169344"/>
   <ldip:linkedFile rdf:resource="http://www.ldip.org/file3">
   <ldip:imageType>photomicrograph</ldip:imageType>
   <ldip:stain>H and E</ldip:stain>
   <ldip:tissue>lung</ldip:tissue>
   <ldip:organism>human</ldip:organism>
   <ldip:objective>10x</ldip:objective>
   <ldip:diagnosis>normal</ldip:diagnosis>
</rdf:Description>
</rdf:RDF>
```

RDF File 2 (Describes a microscope referenced by File 1)

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ldip="http://www.ldip.org/ldip_sch#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="urn:www.ldip.org:ldip:Olympus_BH2_224085">
<rdf:type rdf:resource="http://www.ldip.org/ldip_sch#Instrument"/>
<ldip:instrumentType>Microscope</ldip:instrumentType>
   <ldip:make>Olympus</ldip:make>
   <ldip:model>BH2</ldip:model>
   <ldip:serialNumber>224085</ldip:serialNumber>
</rdf:Description>
</rdf:RDF>
```

RDF File 3 (Describes a camera referenced by File 1)

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ldip="http://www.ldip.org/ldip_sch#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="urn:www.ldip.org:ldip:Infinity_3_00169344">
  <rdf:type resource= "http://www.ldip.org/ldip_sch#Instrument"/>
  <ldip:instrumentType>Camera</ldip:instrumentType>
  <ldip:make>Infinity</ldip:make>
  <ldip:model>3</ldip:model>
  <ldip:serialNumber>00169344</ldip:serialNumber>
</rdf:Description>
</rdf:RDF>
```

Multiple RDF documents pointing to multiple images or multiple parts of a single image file

Suppose, as in the previous example, that the triples relevant to your image lie in multiple RDF files. Suppose, further, that your image is just one of a set of images that were all obtained during the same session and that all the images apply to the same patient. This situation is routine for radiologic images, wherein dozens of images transecting the brain, or the abdomen may form part of the same report.

How might you annotate this complex set of data files and image binaries? Simply include an RDF assertion for each image.

RDF File 1 (Describes 2 images)

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ldip="http://www.ldip.org/ldip_sch#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">
```

```
 <rdf:Description rdf:about="http://www.gwmoore.org/ldip/ldip2103.jpg">
    <rdf:type rdf:resource= "http://www.ldip.org/ldip_sch#Image"/>
    <dc:title>Normal Lung</dc:title>
    <dc:creator>Bill Moore</dc:creator>
    <dc:date>2006-06-21</dc:date>
    <ldip:instrument_id
rdf:resource="urn:www.ldip.org:ldip:Olympus_BH2_224085"/>
<ldip:linkedFile rdf:resource="http://www.ldip.org/file2">
    <ldip:instrument_id
rdf:resource="urn:www.ldip.org:ldip:Infinity_3_00169344"/>
<ldip:linkedFile rdf:resource="http://www.ldip.org/file3">
    <ldip:imageType>photomicrograph</ldip:imageType>
    <ldip:stain>H and E</ldip:stain>
    <ldip:tissue>lung</ldip:tissue>
    <ldip:organism>human</ldip:organism>
    <ldip:objective>10x</ldip:objective>
    <ldip:diagnosis>normal</ldip:diagnosis>
</rdf:Description>
<rdf:Description rdf:about="http://www.gwmoore.org/ldip/ldip2201.jpg">
    <rdf:type rdf:resource= "http://www.ldip.org/ldip_sch#Image"/>
    <dc:title>Normal Lung</dc:title>
    <dc:creator>Bill Moore</dc:creator>
    <dc:date>2006-06-21</dc:date>
    <ldip:instrument_id
rdf:resource="urn:www.ldip.org:ldip:Olympus_BH2_224085"/>
<ldip:linkedFile rdf:resource="http://www.ldip.org/file2">
    <ldip:instrument_id
rdf:resource="urn:www.ldip.org:ldip:Infinity_3_00169344"/>
<ldip:linkedFile rdf:resource="http://www.ldip.org/file3">
    <ldip:imageType>photomicrograph</ldip:imageType>
    <ldip:stain>H and E</ldip:stain>
    <ldip:tissue>lung</ldip:tissue>
    <ldip:organism>human</ldip:organism>
    <ldip:objective>2.5x</ldip:objective>
    <ldip:diagnosis>squamous cell carcinoma</ldip:diagnosis>
</rdf:Description>
</rdf:RDF>
```

RDF File 2 (Describes a microscope referenced by File 1)

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ldip="http://www.ldip.org/ldip_sch#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="urn:www.ldip.org:ldip:Olympus_BH2_224085">
<rdf:type rdf:resource="http://www.ldip.org/ldip_sch#Instrument"/>
<ldip:instrumentType>Microscope</ldip:instrumentType>
    <ldip:make>Olympus</ldip:make>
    <ldip:model>BH2</ldip:model>
    <ldip:serialNumber>224085</ldip:serialNumber>
</rdf:Description>
</rdf:RDF>
```

RDF File 3(Describes a camera referenced by File 1)

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ldip="http://www.ldip.org/ldip_sch#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="urn:www.ldip.org:ldip:Infinity_3_00169344">
   <rdf:type resource= "http://www.ldip.org/ldip_sch#Instrument"/>
   <ldip:instrumentType>Camera</ldip:instrumentType>
```

```
    <ldip:make>Infinity</ldip:make>
    <ldip:model>3</ldip:model>
    <ldip:serialNumber>00169344</ldip:serialNumber>
  </rdf:Description>
  </rdf:RDF>
```

The same approach can be used to reference multiple images within a single image file. The rdf:about attribute can point to any file block or file element that contains the part of the image that is the intended subject of the triples (e.g., region of interest, thumbnail, tile, waveform, color map, and so on).

The Perl scripts

It would not be useful to have RDF documents if they could not be easily constructed, modified, transformed, merged, and analyzed by software agents. Furthermore, much of the power of RDF comes from its generalizability. A software agent that interrogates one RDF document should be able to interrogate any RDF document. Once a generalized software agent is built, it becomes easy to make the agent autonomous (i.e., able to traverse multiple RDF documents and to build inferences based on the meaningful assertions contained in those documents.)

Toward this end, Perl scripts will be available as supplements to this manuscript. Please visit the API resources site for updates.
http://www.pathologyinformatics.org/informatics_r.htm

These Perl scripts can be used as prototypical utilities with some of the functionality that will be needed in software agents.

1. Parses any RDF file into a list of triple (rdf2trip.pl)
2. Converts LDIP CDEs to RDF schema (cde2rdfs.pl)
3. Converts n3 notation to RDF (rdfn3.pl)
4. Extracts n3 triples from RDF (rdfparse.pl)
5. Inserts RDF triples into a jpeg header (imagecom.pl)
6. Inserts jpeg binary as Base64 ascii inside RDF (base64en.pl)
7. Validates LDIP RDF-Schema (valschem.pl)
8. Validates the LDIP CDE list (val_cde.pl)
9. Validates the XSD datatypes for property ranges (val_xsd.pl)
10. Validates LDIP-compliant RDF files (val_ldip.pl)

Discussion

The purpose of the Discussion section is to provide a rationale for limiting our dependence on traditional data standards by choosing to develop RDF data specifications. We will also explain why we think that XML Schemas (a completely different technology from RDF Schemas) are an inadequate method for specifying data objects.

XML Schemas are meaningless and are a barrier to data integration

XML represents an enormous advance over traditional methods for organizing data (e.g., databases and unstructured text). The attachment of metadata to data values provides a easy way of annotating data values. XML data can be structured with XML schemas that list the metadata elements that must be included in conforming documents, along with their relative locations in the document. The inclusion of datatyping within the XML Schema language provides a way of ensuring that the data

contained in an XML document will satisfy specified requirements in the XML Schema.

The problem with XML schema is that it functions within the realm of data structure, not data meaning. Data meaning is achieved when a metadata-data pair is bound to a specific object. XML schemas simply provide the structure for the occurrences of metadata-data pairs. It is impossible to create autonomous software agents that can interrogate XML documents that do not bind metadata and data to a specific subject. Simply put, a logical inference requires a subject to which the inference applies. If you need to have documents that contain meaningful assertions, you must move beyond XML and into the world of RDF.

Data Specifications contrasted with Data Standards

An RDF Schema is a dictionary of classes and properties. An RDF data specification consists of an RDF Schema that describes the classes and properties in its specific knowledge domain plus an xsd document that defines the datatypes associated with a proprety's range. An RDF document that specifies a data object contains instances of classes listed in an RDF Schema that are bound to data described by the properties listed in the RDF Schema.

RDF data specifications have a number of properties that data standards lack.

1. RDF data specifications are optional. The purpose of a data specification is to provide an opportunity for people to describe their data objects. Data standards, unlike data specifications, are often imposed requirements.

2. RDF data specifications are self-describing and contain all the information needed to interpret the contained information. Data files that conform to standards are often inscrutable and not intended to be read by humans.

3. RDF data specifications can be interrogated by general autonomous software agents. Competently written general software agents can parse and understand any RDF document. This is the underlying premise of the semantic web. Files that conform to most data standards can only be parsed by software specifically written to accommodate the data standard.

4. RDF specifications can reduce the complexity of data. All data can be described in RDF documents consisting of data triples. Data standards have no unifying principle of data description. The presence of competing standards, different versions standards, and proprietary extensions of standards have contributed to the complexity of electronic information.

4. The data in a data specification can be distributed over multiple RDF documents.

5. The assertions in RDF data specifications have meaning, and the meaning is preserved when the assertion is extracted from the data specification document. Data standards do not contain meaningful assertions. There is no general way of extracting components of a data standard and building datasets composed of meaningful assertions.

6. A data specification can comply with multiple RDF schemas at once.

7. A data specification can be written without violating intellectual property or breaching patient confidentiality.

LDIP contrasted with DICOM

DICOM is a complex and highly specialized standard was developed over several decades and that is intended to negotiate a variety of network transactions in addition to encapsulating an image binary plus a variety of image annotations, most of which are related to the technical details related to the image capture device (e.g., CT-scanner, MRI) and of the image capture. DICOM was created at a time prior to the development of XML and of the http protocol and depends of a variety of data control devices that some may find anachronistic (e.g., prescribed byte locations, DICOM-specific data transfer and negotiation protocols). Mastering the technical aspects of DICOM is a lengthy task that requres months spent analyzing the many volumes of DICOM technical reports. With few exceptions, DICOM is something that serves radiologists through proprietary software written for imaging devices that may cost millions of dollars in outlay plus maintenance.

Though DICOM has proven itself to be an excellent standard for radiology images, it is not a popular standard for pathology images. Pathologists typically use off-the-shelf cameras and software, and the raw pixel data is usually automatically provided in a popular image format such as jpeg or bmp or tiff. The special needs of the pathologist consist of conveying clinical, pathologic, stain, and instrument data along with the image binary. The annotative data would be required so that images could be distributed in a form that provides useful archival information that is bound to the image. This information could be used to search/retrieve/group images based on diagnostic features and would enhance the image when it is used in consultation with other pathologists or as an item of research interest.

The purpose of LDIP is to provide pathologists with a simple, inexpensive and easily implemented method for annotating images with useful descriptive data, and for binding these annotations to the images.

Porting between Data specifications and Data Standards

Because all the data in a specification is fully described, it is very easy to write software that will port a specification into a data standard. To have full compatibility between a data specification and data standard, the specification must contain all of the required data elements of the data standard. This usually involves:

1. Studying the data standard and writing an RDF Schema (or supplementing an existing RDF Schema) with classes and properties appropriate for the data standard.

2. Writing software that will parse the RDF document in which the data object is specified (trivial) and transform the triples into a document that conforms to the data standard.

At the current time, there is no software designed to port the LDIP data specification to DICOM or to any other data standard. Once the LDIP specification is completed, software can be written to port between LDIP-compliant documents and DICOM files.

Validating a specification

Software validation is one of the more challenging areas in computer science. Dozens of books, hundreds of manuscripts and many thousands of hours of programmer time have been devoted to this demanding subject. Fortunately, validation in the realm of data specifications can be quite easy.

Basically, data specifications consist of lists of triples. Triples are valid when the following conditions hold:

1. The property in a triple is suitable for the subject.
2. The value of the triple is suitable for the property.

Validating an RDF document (in the context of this manuscript, a document that specifies a data object or objects) comes down to this:

1. Checking that the document is well-formed XML.
2. Checking that the document is well-formed RDF?
3. Checking that the triples are valid?

Certifying compliance with an RDF specification

Certifying data compliance is perhaps the weakest (or strongest, depending on your point of view) feature of RDF specifications. Users of an RDF specification, such as LDIP, can choose the classes they wish to include in their RDF docuents. They can mix LDIP RDF Schema classes with classes obtained from other RDF schemas. Their final RDF document may contain very few classes from the LDIP specification.

It would be easy to say that data specifications are not about compliance. Data specifications are about describing data objects completely and meaningfully. Nonetheless, those who make the effort to use a specific RDF Schema may wish to indicate that their document conforms to the Schema.

At this time, we know no widely accepted way of determining compliance with an RDF data specification. We suggest that the following be included in your RDF documents:

1. In a comment, state that the document was prepared in compliance with the RDF Schema.

2. In the same comment, provide a list or count of the class instances from the current document that were derived from classes defined in the RDF Schema.

3. In the same comment, describe any tests performed to validate the document against the RDF Schema.

Owl and DAML

Owl and DAML are extensions of RDF. They extend RDF in a prescribed manner, through well-designed RDF Schemas that build on W3C's top-level RDF Schema.

Neither OWL or DAML are discussed here, but we remind you that all RDF documents may benefit from classes and properties available in publicly available RDF Schemas. We urge readers to visit these URLs for further information:

http://www.w3.org/TR/owl-ref/
http://www.daml.org/

LDIP (Laboratory Digital Imaging Project)

The Association for Pathology Informatics Laboratory Digital Imaging Project (LDIP) was developed in 2004 under the direction of the Association for Pathology Informatics (API), a division of the American Society for Investigative Pathology (ASIP) to develop and promote the use of pathology images to improve patient care and improve the quality of pathology research.

Currently, LDIP's efforts are focused on establishing a free, open access, voluntary specification that will permit images generated in pathology and in clinical laboratories to be widely shared across different applications, databases, and operating systems for the purpose of enhancing image annotation, integration, archiving, publication, and analysis. The specification is being written as an RDF (Resource Description Framework) schema.

The LDIP committee is composed of members of API and corporate sponsors. All of the documents produced by the LDIP committee are freely available to the public and can be downloaded from the LDIP web site (www.ldip.org)

General instructions for creating an RDF Schema for a knowledge domain or for a class of data objects

You can easily create an RDF Schema to specify the classes and properties relevant to a knowledge domain.

1. List the classes and properties that comprehensively describe a knowledge domain.

2. Determine a class heirarchy. Every class on your list must be a subclass of something else. The top level classes may be subclasses of Class.

3. Everything that is not a class must be a property. Determine the domain of each property (the classes to which the property applies) and the range of each property (the kind of data value associated with the property). The default value for a property's range is "Literal".

4. Prepare an xsd datatype declaration for each Property range that requires a constrained data type.

5. Prepare a CDE list that includes the basic CDE annotations recommended in ISO-11179 and the basic annotation needed to describe classes, properties and datatypes for property ranges. Use the generic LDIP CDEs for classes and properties as a default.

6. Validate the CDE list (by careful proofreading or through a software utility such as validcde.pl)

7. Convert CDEs to an RDF schema using a software utility (such as cde2rdf.pl) or "by hand" if the CDE list is short.

8. Validate the RDF schema.

9. Vet the RDF schema through the intended user community

10. Distribute the RDF schema to the public

11. Repeat steps 1-8 ad libitum, providing a new version number to each successive specification.

General instructions for specifying biomedical data using RDF Schemas

You can easily create RDF documents that fully describe a data object and that comply with one or more RDF Schemas written for a knowledge domain.

1. Look at your own available data for the data object. List your data as triples sequenced as: subject - metadata - data

2. Create an RDF header that includes the URL of each of the RDF Schemas that you will use in the document. Be sure to include the Dublin Core RDF Schema. This document includes the elements that describe the file (i.e., creator, data of file, type of file, etc.) and is used by librarians to index your document.

3. Use the RDF Schema(s) appropriate for the knowledge domain of your data object. Determine which of your listed triples have subjects that are instances of classes in the RDF Schema and metadata consistent with class-appropriate properties. Type these subjects as class instances. Check that the data values conform to any data value constraints listed in the xsd datatype file associated with the RDF Schema.

4. If you have common data elements (classes or properties) that are not part of any public RDF Schema, create your own RDF Schema to accommodate these elements and use the RDF Schema as the resource for those elements wherever they appear in your RDF specification document.

5. Contact the curator of the public RDF Schema that is appropriate for the elements you created and ask if your RDF Schema can be added to the public RDF Schema

6. Validate that the document is well-defined XML, syntactically correct RDF and that all triples conform to class-property-datatype descriptions from the RDF Schema.

Problems with RDF

In the meta-reality of informatics, perfect things may be implemented imperfectly. The issues that concern us most regarding RDF are as follows:

1. Despite the hype, RDF is not widely used. The web is virtually all HTML, with a minor contribution in XML and a negligible contribution from RDF. Many internet visionaries predict a bright future for RDF. Now is a good time to master this simple but fascinating model of reality.

2. RDF is growing in complexity. The many extensions to RDF (including OWL and DAML) have made it difficult to master every aspect of the subject. In this manuscript, we have included only selected aspects of RDF that we consider essential for productivity.

3. RDF, and ontologies written for RDF, do not restrict multi-class inheritance. In our opinion, this is a huge problem that can lead to hopelessly complex systems. We strongly recommend extending Properties to multiple classes rather than extending multiple classes to data objects.

4. Though many magazine articles have been written extolling the virtues of RDF and of the semantic web, most of these articles are pitifully superficial. Not surprisingly, no two persons ever seem to ever have the same "take" on the subject of RDF data specifications. The best literature seems to come from the W3C, but these Web recommendations are technical reports and do not focus on the needs of biomedical informaticians. In this this manuscript, we have tried to provide one interpretation of the theory and implementation of RDF as it relates to specifying data.

Dangers of complexity and multiple inheritance

Software programs, unlike physical constructs, have no limits to complexity. Software programs intended to be simple may become complex when recursive subroutines are include or when subroutines call other subroutines. Data standards add to the complexity of software when the standards themselves are represented in a manner that is not common to other standards. For instance, if one standard is represented by an XML schema, and another standard is represented by bit-apportioned data records, and another standard is represented by free-text descriptions, and all the standards contain elements that have ambiguous relationships to elements in the other two standards, the harmonization of all three standards in a single software implementation becomes impossible.

Complexity has always posed limitations on the reliability of software systems. There is no way to predict or to avoid outcomes in a complex system. Since we have no current methods for coping with complexity, the best solution is to reduce the level of complexity in software systems whenever feasible.

RDF reduces complexity four ways:

1. RDF Schemas are simple documents that may contain only classes and properties.

2. RDF Schemas can be designed to avoid multi-class inheritance through the use of multi-class property domains. Avoiding multi-class inheritance reduces the complexity of drawing inferences from RDF documents.

3. The data in RDF documents exist in triples, and these triples preserve their meaning when extracted from the document in which they are contained.

4. A software agent that parses one RDF document should be able to parse any RDF document.

.

Summary

1. Meaning is achieved by binding a metadata-data pair to a specified subject, into a so-called triple. Example:
Jules J. Berman (subject) **favorite food** (metadata) **pizza** (data)

2. The subject of a triple needs to be identified as a unique data object. The metadata needs to be defined, and the data needs to have a specified structure. These are achieved with identifiers, that uniquely specify class instances; with RDF Schemas that assign classes to subjects and assign properties to metadata; and with xsd datatypes that impose structure on data values.

3. RDF documents consist of triples. RDF documents begin with a declaration of the RDF namespace in which the syntactical elements of RDF are defined. When the RDF document creates instances of classes defined in one or more external RDF Schema documents, the namespaces of the RDF Schema documents are also listed at the top of the RDF document.

4. Triples can be collected from heterogeneous RDF datasets, and the data pertaining to a specified subjects can be easily merged by RDF parsers. An RDF parser is a general utility that will work equally well for any RDF document because all RDF documents conform to the W3C's RDF syntax recommendation.

5. A specification is a document that describes a data object in a manner that can be understood by humans or by computers. An RDF Schema is a dictionary of classes and properties that can be used to completely describe a data object in its knowledge domain. There may be many different ways of specifying an object in an RDF document, but if the specification is in the form of an RDF document that uses RDF Schemas to create class instances and define metadata and uses XSD datatypes to constrain the value of data, the specification will be understood by competent general software agents written to interrogate RDF documents.

6. Data specifications have many advantages over data standards. By writing domain-specific RDF Schemas (e.g., the LDIP image specification) we can reduce our dependence on data standards and enhance our ability to integrate data collected from heterogeneous datasets.

Competing interests

The author declares that he has no competing interests.

Disclaimers

The Perl scripts are provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular

purpose and noninfringement. In no event shall the author or copyright holder be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.

The opinions expressed in the manuscript are those of the authors and do not necessarily represent the policy or opinions of the Association for Pathology Informatics or of the Laboratory Digital Imaging Project or any of their members.

This manuscript is a work of literature and has no other purpose. The statements included in the manuscript are provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the author or copyright holder be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the manuscript or the use or other dealings in the manuscript.

Intellectual property

The current version of this manuscript is at:
http://www.pathologyinformatics.org/Resources/jjb_gwm.pdf

Links to supplemental files, as they are released, will be available at the Association for Pathology Informatics resources site:
http://www.pathologyinformatics.org/informatics_r.htm.

It is strongly recommended that users download the most current version of this manuscript and delete all prior versions.

Acknowledgements

Dr. G. William Moore, M.D., Ph.D., provided all of the photomicrographs and most of the draft Common Data Elements for the Laboratory Digital Imaging Program. We thank Dr. Ulysses Balis, co-chair of the Laboratory Digital Imaging Program, for his many useful suggestions, and we thank Dr. Robert Leif, for his many helpful criticisms.

Citations to this article

Public or published documents that refer to or draw from information contained in this manuscript should include a citation to this manuscript. The recommended form of citation is:

Berman JJ, Moore GW. RDF data specifications: the simple alternative to complex data standards. http://www.pathologyinformatics.org/Resources/jjb_gwm.pdf, July, 2006.

If the URL is inactive, the e-manuscript should be available directly from Jules J. Berman at jjberman@alum.mit.edu..