A simple metadata ontology (and how it could evolve)

Chris Patton, 31 May 2013

To start off, I think it would be useful to describe our project as I understand it. Our goal is to create a crowd-sourced, open platform ontology for metadata which could be applicable to just about any field or subject matter (intellectual property in general). The key innovation is that the set of terms and their relationships will evolve to peoples' needs in a social ecosystem. We will provide a service that (a) allows users to input metadata with all required terms (according to our ontology) and export it in some controlled format (e.g. XML), and (b) to propose new terms when they're needed. Stable ontological terms will emerge in two ways:

(1) *Vernacular*: users propose new terms and definitions for those terms. These will be discussed and voted on in the stack overflow manner.

(2) *Canonical*: we should allow an expert to declare a term in the ontology.

As the ontology will be entirely user-driven, there are a number of pathologies that could develop. *Redundancy:* two terms exist in the ontology with the same semantic meaning. *Irrelevance:* a particular term is not applicable to the particular subject matter, but is required by the ontology. [Examples would be good.] What is needed is a way of controlling the evolution of the ontology in an automated way. In this document, I propose simple semantics that helps to avoid these issues and maintain a sound ontology as the registry evolves. Part of my approach prescribes how new terms are proposed.

First, here's the essential specification of what our system should accomplish. From now on, *term* is meant to describe classes of metadata, e.g. Type, Format, Creator, etc. *Instance* will be used for particular values described by a term—e.g., JPEG, plain_text, or PDF are instances of Format. We want a function, call it *ExportMetadata*() returns a set of only those (term, instance) pairs that pertain to the subject matter of the users data, according to our ontology. Part of this interface is a way to propose new terms in a systematic way.

MetadataExporter

I start off by describing the front end user experience. (For now, this will be a simple console-based Python driver program. I imagine it's possible to make this beautiful on the web.) Essentially, the goal is to unite the process of getting metadata and proposing new metadata terms into the same interface. The user inputs relevant metadata terms and the program outputs the responses formatted in a standard way, such as XML. The user is first prompted for basic terms such as Name, Creator, Subject, Publisher, Contributor(s), Date, Language, Rights, etc[1]. At this point, we could generate a unique identifier (Id) with EZID. These are terms with simple domains; terms with more complex meanings require a richer ontology. When the user is prompted for Type, for example, he or she is given a list of possible types, e.g.: document, table, video, audio, image, etc. Particular types often have attributes that are only relevant to them. For example, Dimension is relevant to an image, but not to a document. *The program proceeds such that the user is only prompted for metadata features relevant to his or her data/document/etc*.

1 These terms and their meanings come from the Dublin-core RFC.

To continue, let's look at some use cases. Alice is a botanist. Say Alice wants to export metadata for her dataset about grapes. Her table has three properties: time, temperature, and the expression level of the protein she's studying. She types in the Name, Creator, Publisher, Contributor(s), Date, and Rights and is ready to specify the Type. We'll call the possible values for non-primitive terms instances of that term. The program gives her the following instances of Type:

document,
table,
video,
audio,
image, or
[propose new instance]

Let's say Alice is new to our system wants to propose the “dataset” as a new Type. This initiates a debate in the stackoverflow manner about the meaning of dataset. Alice insists that a dataset is a fundamental type. Finally, someone persuades her that, in fact, her grapes are more generically a table. Alice agrees and chooses table from the above list.

In addition to possible values for terms, the program also provides a list of terms relevant only to that term. After Alice chooses table, the program provides no additional options and is ready to export the metadata. However, Alice believes her metadata could be enriched by including the table's dimension (hers is 3-dimensional). She can now propose Dimension as a new term relevant to the table Type. The program tells her that Dimension already exists as a term for the image Type. She believes that the semantics are the same for the dimension of tables and images, so she goes ahead and proposes the new term. This initializes another debate, and the consensus is that Dimension is also relevant for tables. After some the vernacular term stabilizes, it is added to the core ontology by an expert user.

Alice successfully added a term and her confidence (and reputation) has gone up a notch. Next, she decides that the term Organization is not captured by the ontology. She proposes this new term, which initializes yet another debate. This time, Dave the curator reminds her that Organization is semantically prescribed by the Creator term, and would therefore introduce a redundancy. Alice sees the light and the thread closes.

To conclude, Alice has proposed three modifications to the metadata ontology:

(1) instance “dataset” of term Type,
(2) term Dimension of instance “table”, and
(3) term Organization of term Base[2].

As far as I can conclude as of this writing, these are the three basic changes that may be proposed. The next section describes the semantics of an ontology that would allow our system to capture all of these.

---

2 Terms have a hierarchal structure with Base as the root. This architecture is described in detail in the next section.

Term domains and semantics

The ontology is comprised of a pool of *terms*, *instances* (types) of those terms, and *relations* between them. There are a number of primitives with simple domains. We won't worry about the mechanics of these domains too much, but here they are:

- *String*
- *Int* : integer greater or equal to 0, the natural numbers
- *Time* : Representation of date/time with arbitrary precision

Dublin-core prescribes the following (likely) primitives:

- Name of type *String*,
- Creator of type *String*,
- Description of type *String*,
- Publisher of type *String* (potentially non-primitive),
- Contributors of type *String list*
- Subject of type *String*,
- Language of type *String*,
- Date of type *Time*,
- Rights of type *String, and*
- Id of type *String* (necessarily unique).

There are seven more terms in the set which we'll consider to be non-primitive for our purposes. These are:

- Type,
- Format,
- Source,
- Relation, and
- Coverage.

Each term has an *instance set:* e.g. Type = {document, table, video, image, audio, … }, Format = {PDF, XML, plain_text, JPEG, mp3, … }, Name is a subset of all strings, Id is a subset of unique strings. It's also possible to leave terms unspecified. In this case, we'll call it *nil*. Therefore each instance set is the union of {*nil*} and all possible instances.

The ontology is organized in a hierarchy, the root of which is the term Base. This is analogous to class hierarchies in object oriented programming, except that instances of our terms can have attributes unique to that instance. There are three possible relationships:

(1) Term A → instance a : a is a type of A, e.g. Format → JPEG.
(2) Term A → Term B : B is a characteristic of A, e.g. Base → Format.
(3) Instance a of A→ Term B : B is a characteristic of the particular instance a of A.
(4) ~~instance a → instance b~~ : this relationship is excluded, as it's not sensible.

I claim that this specification precisely defines all possible ontological relationships between metadata terms. If this is the case, then all possible changes to the ontology (through some crowd-sourcing mechanism) are well-defined. This makes it possible for us to control the evolution of the ontology in a systematic manner.

Architecture
In addition, this semantics gives us a specification for the software system.

- A *Term* is a metadata term and its corresponding instance set. If it's a primitive, its type is noted and the instance set is empty. A term also includes a definition.

- An *Instance* is an instance (type) of a term paired with a reference to its *Term.*

- The *TermPool* contains the set of all *Terms* (dictionary in Python, hash table in C).

- The *InstancePool* contains the set of all *Instances*.

- *Ontology* is a graph representation of term-term, term-instance, and instance-term relations. The graph is rooted—it's not always a tree, but I do believe it's a DAG (not important)—at the term *Base*. Base has some primitive attribute terms, such as *Name*, *Creator*, *Date*, etc.

To export metadata (standard query), we simply traverse the *Ontology* graph breadth-first starting at *Base*. First, for primitives terms, we input the correct value. For non-primitives, we either select an instance from the instance set or propose a new instance. At the same time, it should be possible to propose any new term for a given instance or term. Finally, for *ExportMetadata*(A), for all terms B such that A → B is a relation in *Ontology,* do *ExportMetadata*(B).

Automatic redundancy resolution via NLP
A stated ambition of this project is to use NLP techniques to analyze definitions of proposed terms. The proposed ontology semantics allows to resolve the relationships between terms in order to determine computationally if there is a collision. Of course, I don't think there's anyway to do this without human input, but it provides a way to flag obvious redundancies.

Let's think about Alice as an example. You may recall that she proposed *Base* → *Organization*. Let's say her proposed definition was: “entity or group for which the material was prepared”. Say the definition for the term *Creator* is “person, group, or entity for which the material was prepared”. *Creator* is more general than *Organization* (this is a trivial case in NLP), and both are at the hierarchal level. *Organization* is therefore an obvious redundancy.