Practical Ontology Design Principles in the VIVO context

This page attempts to define high-level principles, requirements, and components for VIVO ontologies.

Principles

- define top-level ontologies that will remain stable, organized by logical groupings influenced by existing accepted ontologies
    - One template may be the Knowledgeweb and its modular organization
- Sometimes we will reuse existing ontologies directly
    - For example, we have imported the Bibliographic Ontology (Namespace Bibo) to serve as our ontology of publications and to facility data transfer in and out of VIVO
    - We may drop certain classes or properties (such as those for legal documents), but we will use the Bibontology namespace for VIVO data
- Sometimes we will use selected classes and/or properties from well-known ontologies
    - E.g., foaf:Person
- Sometimes we will just establish mappings from VIVO classes and properties to another ontology

Management

- Let individual implementations add more specific sub-classes and sub-properties as needed, but not modify the consensus core intended for use in aggregating content at the national level

Design choices

Object property or data property?

Very often we will have to decide whether to use an object property or a data property for information such as awards – and there is no right or wrong answer. If we know we want to manage awards by name and track multiple recipients of the same award, then it would make sense to create the award as its own object in the system (e.g., Nobel Peace Prize) and to use an object property to link the award with recipients.

This approach has 2 problems. The first is that there is additional relevant information – namely the year of the award – that needs to be tracked, and an object property statement is just a subject-predicate-object triple that cannot carry additional information such as the year the award was given. This has pushed us to develop the Context nodes approach for relationships that carry additional attributes.

The alternative is to use a text data property, and list awards in human readable form but with only the possibility of using string comparison to detect whether multiple people have received the same award. In the case of awards, many of which are in fact not directly comparable, this approach seems preferable.

Single or multiple valued data properties?

There remains the choice, however, of using a single data property containing (potentially) multiple values (typically a block of text listing several awards, often in bulleted format), or to enter each award as a separate data property statement. The former is convenient for retrieval (one per person), and allows the user to control sort order and formatting of the content within the block of text. The latter makes analysis easier because we can separate one award's text from another. However, it has the major limitation that without the use of Context nodes, there is only 1 way to control the order in which a person's awards will appear when listed as part of their profile – using an alphanumeric sort order that can be applied without regard to user preference. This works well for something like keywords, but not so well for something like awards, where a person might prefer to have the listings sorted in reverse chronological order.

What are context nodes?

Context nodes are described separately in more detail below, but it's important to discuss their application here as well, using common examples.

First is the matter of terminology – the term context node came from the MESUR project and we have continued to use it in part to reference that project. A node, however, in this case means the same as what we refer to on the VIVO project as "individual" or "entity" or "object" – an individual, independent object in the VIVO system, belonging to 1 or more types or classes, and capable of being the subject (or "domain") of object and data properties and the object (or "range") of object properties.

Sometimes we use a context node as a "stub individual" or "stub entity" – an individual not intended to be displayed by itself, out of the context of its single related individual. The Educational Background class is a good example of this use of context nodes – we want to track distinct attribute fields for a person's educational background, including the year of graduation, the degree received, the name of the institution, and the person's primary field of study. For reasons described above, we don't just want to list all this information as text within a data property, but want to be able to sort a person's degrees by year or by degree received. We also want to be able to use the institution name (and potentially even an object relationship) in finding connections among people, as part of their personal provenance. We don't, however, typically want to display lists of Educational Background individuals all by themselves – hence the term "stub", as a secondary vs. a primary or 1st class of individual.

More complex are the classes under the Relationship class – Authorship and Employment. These are again intended to be hidden, but have equally important object relationships to both the author and the publication, or the employee and the employer. They exist to carry explicit additional information beyond the fact of authorship or employment per se – the author's rank among the authors of a publication, or the start and end dates of employment. The extra layer of the context node makes life more complicated but is the only way to handle requirements for ordering and/or temporal ranges on relationships.

Context nodes

Context nodes or stub entities, have been developed as a way to supplement the information that can't be stored using direct relationships between primary entities, which are limited to the subject-predicate-object triple of an RDF statement.

The prime example is to keep track of author order in a publication – if each author has only a direct author relationship to the publication, it's not possible to control the order in which authors would be listed as linked to the publication. Introducing an Authorship node (a subclass of Relationship in the

```
[Current ontology sandbox|https://confluence.cornell.edu/display/ennsrd/Current+ontology+sandbox])
```

provides the ability to include an author order data property in connecting an author with the publication.

from the MESUR paper
The principles espoused by the OntologyX ontology are inspiring. OntologyX uses context classes as the "glue" for relating other classes, an approach that was adopted for the MESUR ontology. For instance, the MESUR ontology does not have a direct relationship between an article and its publishing journal. Instead, there exists a publishing context that serves as an N-ary operator uniting a journal, the article, its publication date, its authors, and auxiliary information such as the source of the bibliographic data. The context construct is intuitive and allows for future extensions to the ontology. OntologyX also helped to determine the primary abstract classes for the MESUR ontology. Unfortunately, OntologyX is a proprietary ontology for which very limited public information is available, making direct adoption unfeasible for MESUR. As a matter of fact, all in- spiration was derived from a single PowerPoint presentation from the 2005 FBRB Workshop (G. Rust. Ontologyx. In Functional Requirements for Bibliographic Records Workshop Proceedings, Dublin, Ohio, May 2005).

MESUR: A Practical Ontology for the Large-Scale Modeling of Scholarly Artifacts and their Usage, Marko Rodriguez, Johan Bollen, and Hervert Van de Sompel