Trait Information Portal Ontology

The main reasons behind the development of the trait information portal ontology are:

* Provide a system in which one is not forced to shape data to fit a specific structure, but rather provide the means to relate data to existing standards and allow these standards to be extended.
* Provide a system in which relations between traits can be immediately visible in data.
* Provide a system in which all involved parties can discuss share ideas and jointly work on standards that are immediately made available for data annotation.

We have developed a number of data portals, ranging from centralized systems up to totally decentralized systems, using web services or file uploads. There are countless technologies and workflows for managing data transfer to and from remote locations, but these do not help the problem of data standardisation. In the end, data must be recognisable both by the humans that consult it and by the machines that catalogue it, this process of standardisation is the key to two main aspects of data portals: data *quality* and data *availability*.

The more you enforce standards, the more you minimise errors, ease data aggregation and provide a clear context for the data, providing higher *quality*. But this standardisation process requires a lot of work, which hinders data *availability* in those systems.



As with all things in life, there is no free ride, if you want very high data quality you cannot expect to have a very high data availability, except in cases where data provision is a requirement; likewise, if you accommodate a large and loose range of standards, you might make life easier for data providers, but aggregating data will become a more difficult task.

Another issue with implementing standards is *participation* and *discovery*. Standards depend on best practices and scientific research, this means that scientific research shapes data and data shapes standards. This also means that there is a great number of people that should get *involved* and unless there is a *discovery* mechanism and one place for everybody to share ideas, propose and access these standards, a lot of actors and ideas might be left out. This brings the need to create a place where metadata is consultable, compiled and available for exchange.

How do we do it now?

The most popular way to record data standards is descriptor lists. A descriptor is an object that has a name, a definition and a series of attributes that illustrate *the way* a measure is taken and *what* the measure measures. There are no formal rules or structure for building descriptors, just the common sense to provide enough information for people to use it in the field, which, ultimately, is their goal.

Descriptors are mostly published on paper, they are being made available on the Internet and are often compiled in database tables, but there is no formal discovery mechanism that makes it easy to find specific traits. Descriptors are not formally related to other descriptors, so it becomes difficult to spot duplications. Also, there is no formalised mechanism to uniquely identify a descriptor, such as a URL or code, this means that if I refer to “plant height” I might be referring to a dozen of “plant height” descriptors, each with their specific characteristics.

Nevertheless, this is how the majority of data aggregation portals or systems work: a group agrees on a set of standards, it compiles a list of descriptors and fills data in templates annotated by these descriptors. This way of working functions well, since the data structure can be determined and tuned beforehand and the nature of the data received is known. However, this represents an island in which there is no way to know or compare with other systems that collect the same kind of data. This problem has been in the heart of what Bioversity has done in the last years: SINGER, EURISCO, EUFGIS, GENESYS and others, these are all systems that collect data from heterogeneous providers to aggregate it into a single source.

The problem of *how* to transfer data, using distributed or centralised systems, has been the focus during these years. With SINGER we have tried all possible solutions, from totally decentralised systems based on web-services (demo in Australia) to the manual aggregation of data in the last years, with EURISCO we developed an automatic centralised solution that has been working for over ten years. All these solutions had their advantages and disadvantages, but little has changed in the way standards were managed and implemented.

As long as the nature of the data is known and its structure is fixed, the matter of storing and retrieving it becomes relatively easy, performance can be fine-tuned by adding resources and providing specific views by tweaking the structure, all this can be easily handled by the traditional relational structures. But when the nature of what the data describes changes, or when the kind of data we receive is subject to change often, the traditional relational structure shows quickly its limits. Attempts to solve the horizontal growth of data, such as in the GENESYS project, result in very complex structures that are not sustainable. This brings to a solution that must accommodate horizontal growth without compromising performance and sustainability.

How do we want to do it?

From the above discussion it becomes clear that descriptor lists and relational structures are not powerful enough to provide a sustainable solution to vertical and horizontal growth, to dynamic structures and cannot be the base of a more *intelligent* way of discovering data and traits.

Descriptor lists should be replaced by ontologies in which concepts are related between each other forming a series of graphs. Committing data definitions to a common ontology may not provide a complete solution, but it will guarantee consistency. The main goal is to create a specification of a series of concepts that range from the high level categories, through the specification of measurable traits down to the specification of the units used to measure these traits.

In addition, using relationships, data specified by specific elements of one ontology may be discoverable trough elements of other ontologies, providing the semantic rules that inference engines will be able to use in order to search data in a more advanced and integrated way.

The ontology we are building for the trait information portal is constituted by four main objects: the *term*, the *node*, the *edge* and the *tag*.

Term

The starting point of the ontology is the *term*, which can be equated to the descriptor. Terms are objects that feature the following attributes:

* *Global identifier*. All terms *must* have an identifier in the form of a string that will uniquely identify it among all others. This identifier is composed of two elements:
  + The *namespace*, which is a reference to another term that represents a container or logical grouping for a set of identifiers, this allows more than one term to share the same *local identifier*.
  + The *local identifier*, a string, uniquely identifies the term among all the other terms that share the same namespace.
* *Label*. Although not strictly required, this attribute is strongly suggested: it represents the name or short description of the term expressed in several languages. This is what a human would use when referring to it.
* *Definition*. As the label, this is a suggested attribute that represents the description of the term. Whenever the label is not self-explanatory, or whenever there is the need for further specification, this attribute can be used to provide such information and, as the label, this information should be provided in several languages.

Besides these attributes, terms should be prepared to host any kind of attribute, pictures, images, links and all that is needed to specify the meaning of the term in the most complete way.

Terms are not formally related to each other, they do not form a specific structure; they simply represent a general concept out of a specific context. Terms should be organised in dictionaries where they are searchable by label, code, synonyms and other attributes that are relevant.

In that sense, descriptor lists could be easily transferred to term dictionaries, making the preliminary step of populating the ontology relatively easy.

Node

The power of ontologies, however, lies is in the ability to relate its elements. The structure of the ontology we are implementing is a *directed graph*, which is a collection of vertices each connected by a predicate in which the relationship has a direction.

The vertices of such graphs are represented in this system by the *node*, which is a subclass of the term, in that it must reference a term. Nodes do not have other required attributes, they represent *a term in context*, and feature any attribute necessary to illustrate or clarify that context.

The interaction between nodes and terms is key in understanding how this ontology can be used and in deciding how you want to shape it.

“Name” is a *term* that defines a string used for identification, the term is generic and does not have a specific context. However, if we instantiate a *node* that references the “name” term connecting it to a person, we have an instance of a person name; likewise if we connect it to an accession we will have an accession name. These two nodes share the same term; they have the same global identifier, but mean different things. It will be likely that the two nodes will feature additional to describe the function of the name in the person context and in the accession context. This is an example of how one could build an ontology following a parallel on how words are used to express concepts.

Another strategy is to have only one node that can reference the same term, in that case you would have “person name” and “accession name”, these will be two terms and there will be one node per term. In this case most of the information can be stored on the term side, since the term implies a lot about its context and the nodes would only exists to be connected.

These two examples show the main principles that drive the design of this ontology: provide the ability to build semantic constructs, or tree structures.

Edge

Nodes are related among each other by *edges*, these objects contain the following attributes:

* *Subject*. The reference to the *node* that represents the subject or origin of the relationship.
* *Predicate*. The reference to the *term* that represents the predicate of the relationship. The predicate can be equated to the type of relationship.
* *Object*. The reference to the *node* that represents the object or destination of the relationship.

Besides these attributes, any other property may be added which can be used as a weight or measurable attribute of the relationship.

Edges provide the glue that connects all the graph vertices and provide a direction to the relationship; the object that is used for annotating data is the tag.

Tags

In an ontology that describes the traits of a crop the leaf nodes would be those that represent the traits and those which could be used to tag data sets, however, data comes in many sizes and colours, “plant height” may be expressed in many ways and often it is not possible nor desirable to convert between these different formats. This means that one has two options: incorporate the *methods* and *scales* in the vertex that defines the trait, or create child *method* and *scale* vertices connected to the trait. In the first case you will end up having a series of “plant height” traits all identical except for some small detail, in the other case you would have a single “plant height” vertex that branches into a series of specialised leaf nodes. The result is the same, each node in the first case or branch in the second will be stored as a different piece of data, but in the second case the structure will allow a greater freedom when relating elements on the ontology. This also serves the purpose of dividing the scope of the ontology: all vertices down to the trait represent the *functional* relationships of the object the ontology is attempting to describe, while everything below the trait node refers to how the function can be *measured*. This division allows two kinds of users to work on the same ontology without stepping on their toes.

Since one vertex is not enough to define a data set, we have introduced the *tag*. This object features a property, which records the path between the vertex that defines the trait and the vertex that defines the data type or scale. This attribute is a path that records the *term identifiers* of the vertices and the *predicates* that relate them. For instance a dataset may have one data set which represents the *plant height* measured *after one month* in *centimetres*, this would be represented in the ontology by a vertex which defines the *plant height* trait, a vertex that defines the *method* used to take the measurement, in this case after one month and a final vertex that defines the *centimetres* unit in which the data is expressed in; this last vertex would also indicate the data type in which the data is expressed in.

The advantage of using a separate object to tag data types is that a short identifier can be generated, leaving the freedom to use URL identifiers in terms without impacting in the creation of the data store.

The other important feature is that the tag path *does not reference the nodes that connect the trait to the scale, but the terms that the nodes reference*. This means that if you have several ontologies that share the same “plant height”, “after one month” and “centimetres” sequence, using any of these to annotate a data set will result in the same tag, thus in the same data field.

Structure notes

The combination of the four above objects create a system that is very flexible, allowing the specification of complex systems.

There may be several nodes that reference the same term, this means that one of these nodes is considered a *master* and the others are considered *aliases*, this feature can be used to create *views* in the ontology. Suppose you have a plant ontology and a series of specific crop ontologies, the *master* ontology would represent the plant ontology which records all the elements and relationships of the specific crop ontologies and the *alias* ontologies would be the crop specific ontologies: both the master and the alias element will share the same terms, but alias elements will only show their specific relationships. The reason to have aliases is to reduce the view to only what is pertinent to that view, excluding all the other relationships that are not relevant to the current view. Imagine the master node for “centimetres”, it would be related to “plant height” as well as any other attribute that can be measured in centimetres: this would render the navigation of an ontology a very confusing experience. Finally, as described before, the tag object is structured in such a way data fields are shared between alias paths: when navigating an ontology to select which elements to use for annotating a specific data set, it is irrelevant which view is used in the navigation, if the path between the trait and the scale elements is the same, this will result in the same tag.

Implementation notes

The system is implemented using two main kinds of databases: document database and graph database.

Document databases store whatever data you provide them, there is one predefined identifier that is used to uniquely identify an object and the data that can be stored is a free-form nested structure. In this system we have selected MongoDB because of the performance, flexibility and scalability. The main duty of that database is to store the actual data and to serve as an *index* for the ontology elements.

Graph databases store triplets of subject/predicate/object relationships, in this system we have selected Neo4j. These databases are generally not as powerful as document databases in terms of indexing and not as scalable in terms of locality (data cannot be sharded as easily as in document databases, since you cannot predict relationships), but they are extremely powerful and fast when *traversing* graph structures is required.

The combination of these two data storage engine types provides the ability to implement inference algorithms at the level of the ontology and fast retrieval at the level of data.