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 the entry point can be a trait, regardless of the kind of object it describes.
* Provide a system in which all involved parties can discuss share ideas and jointly work on standards which 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 the goal of a trait.

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: we agree on a set of standards, we compile a list of descriptors and aggregate data filled in templates annotated with 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 have a clear limit in sight. 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 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.

Term

The starting point of the ontology is the *term*, which can be equated to the descriptor. Terms are objects that require 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 a term, it represents the descriptor’s name.
* *Definition*. As the label, this is a suggested attribute which 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 references to other terms, 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, is in the way its elements can be related in its graph structure that allows navigation in unexpected ways. The structure of the ontology we are implementing is a *directed graph*, which is a collection of vertices each connected by a predicate.

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 will have the same global identifier, but will mean different things. It will be likely that the two nodes will need an additional attribute 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 vocabularies form language.

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. These directed graphs are used to describe traits and their relationship to the different aspects and functions of a plant, as well as to provide the specification of the type and units used to measure such traits. The rules that determine which elements can be used for annotating data are implemented by the tag object.

Tags

Ideally, leaf graph vertices should represent the nodes that can be used to describe data, so, again ideally, the term identifier related to such nodes could be used as a field name; in practice, this cannot be done if we want to create a system that is both robust and flexible. *Tags* are objects that contain a *sequence* of graph vertices, in other words, they contain the path between one node which represents the *feature* of the tag, a series of nodes that represent the *methods* of the tag and a leaf node which represents the *scale* of the tag.

The feature represents the trait, “plant height” could be considered a feature. The method