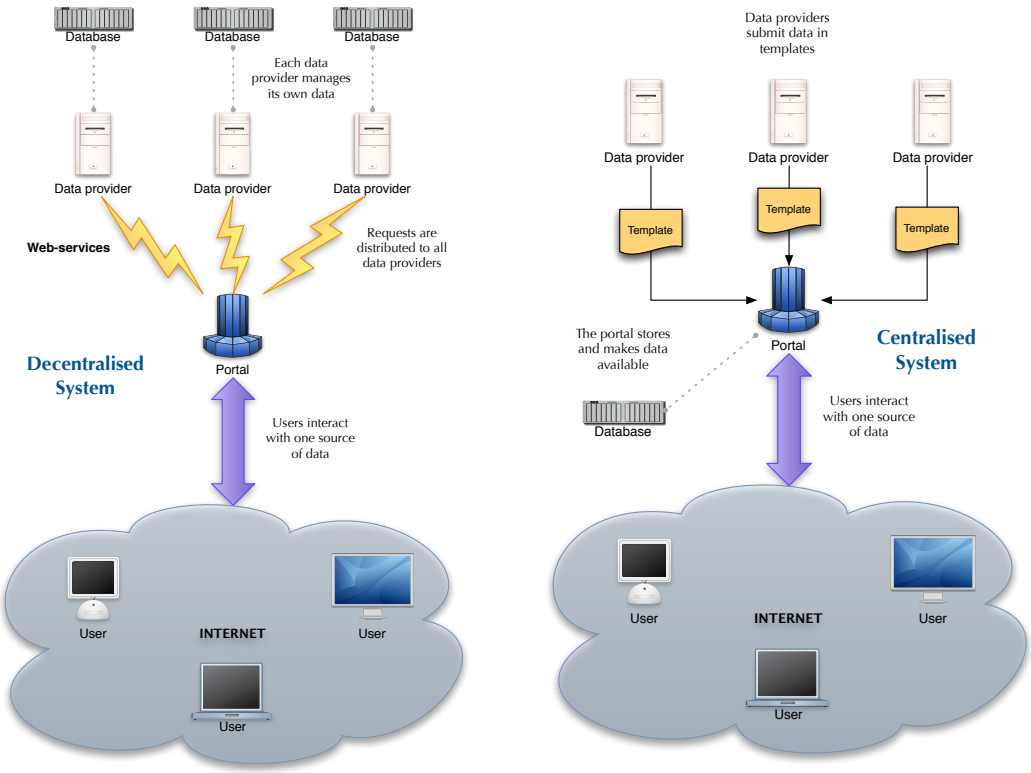
Trait Information Portal Ontology

The main objectives 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 and that can be used for inference.
- 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.

There are countless examples of solutions developed to solve the problems related to transferring data, from totally de-centralised systems to centralised systems, from web-service transport protocols to file upload systems, but there are less examples dealing with implementation and management of data standards.



Metadata is an essential part of the actual data, it is required by humans to understand what the data is measuring and how the measurement was done and it is required by machines to correctly and efficiently store the data and retrieve it according the very diverse kinds of user requests.

The process of standardisation is key to two main features of data portals: *quality* and *availability*.

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

If you want very high data quality you cannot expect to have 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 difficult task.

Another issue with standards is *participation* and *discovery*. Standards depend on best practices and scientific research, this means that scientific research influences data standards, but also the methodology and practice of data acquisition shapes the standards, meaning that a lot of people should get *involved*; unless there is a *discovery* mechanism that makes standards available for consultation and discussion, these may stagnate and not all the people involved might be able to participate.

How is it done now?

A 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* data is taken and *what* the data measures. There are no formal rules or structure for building descriptors, just the common sense to provide enough information for people to apply them in the field, which, ultimately, is their goal.

Descriptors are mostly published on paper, they are being made available now on the Internet and are sometimes compiled in database tables, but there is no formal identification and discovery mechanism that allows finding specific traits an easy task. 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 an URL or a code, this means that one must rely on the name to identify a descriptor and Google to find it, which may introduce a lot of confusion due to homonyms.

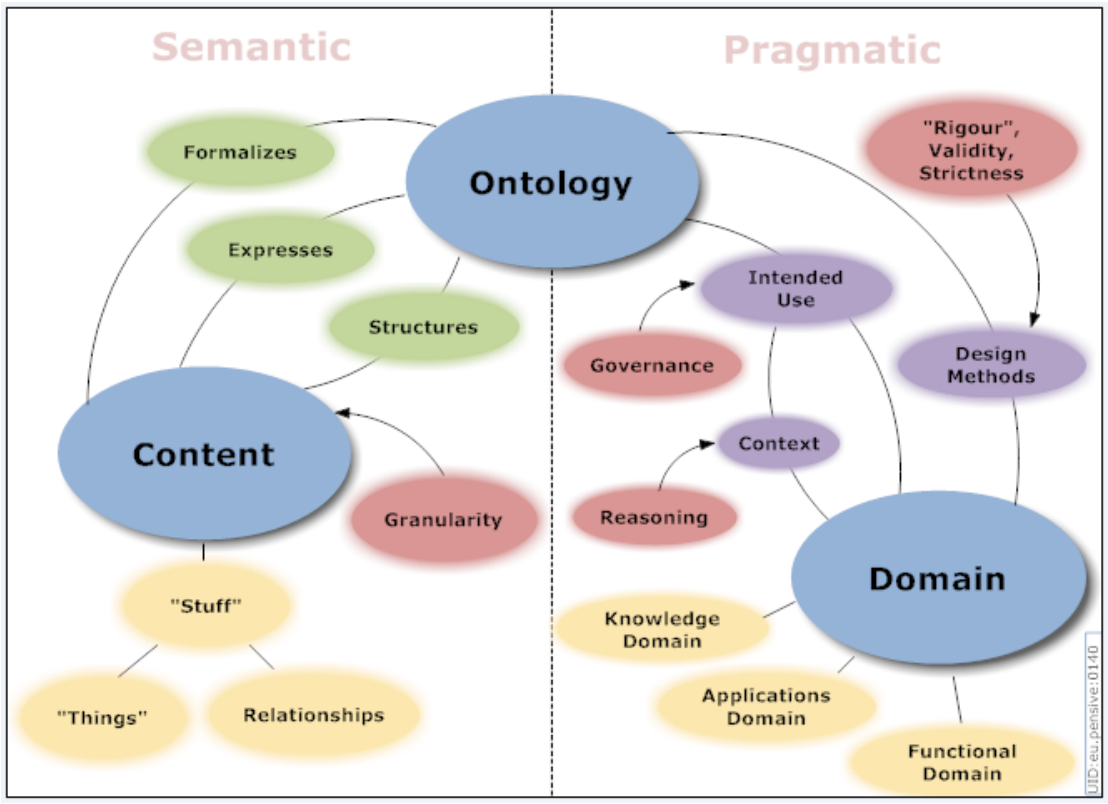
Nevertheless, this is how the majority of data aggregation portals currently work: a group agrees on a set of standards, it compiles a list of descriptors and fills data in templates annotated by these descriptors. This is an efficient way of working, the nature and structure of the received data is known, making the task of storing and serving it relatively easy.

The problem with this approach is that it may solve the vertical aspect, that is, the *quantity* of data, but the horizontal aspect cannot be handled easily: the *quality* or

nature of the data remains fixed, unless great effort is put into implementing a dynamic system.

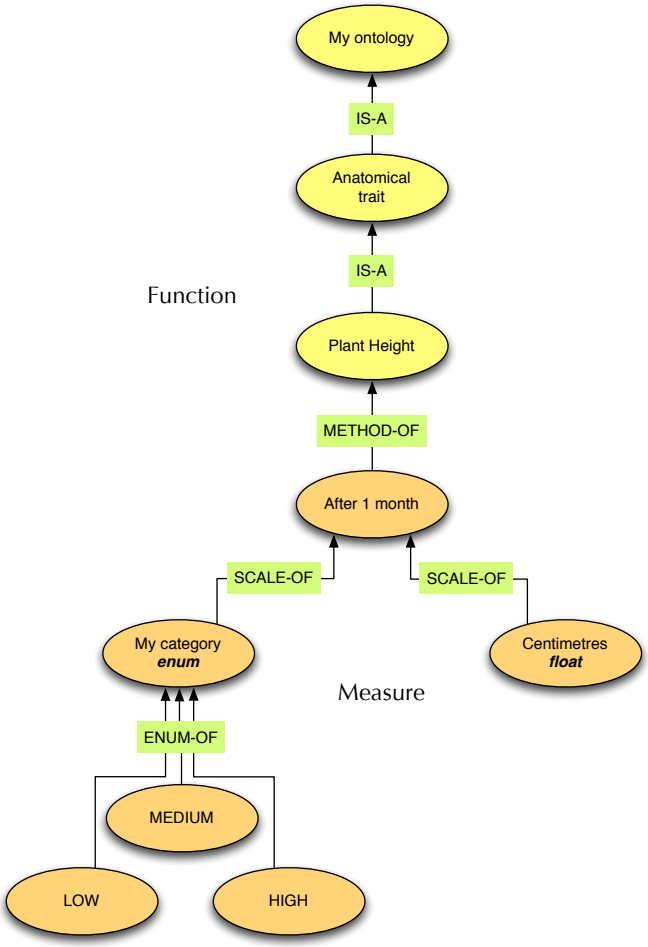
How do we want to do it?

Creating a dynamic system is what we are trying to do with the trait information portal ontology, to allow the system to grow horizontally with the same ease as it can grow vertically. In addition, we want to add a *functional* aspect to metadata, by including in the metadata the functional relationships between the measured traits. This can be done by using the ontology approach, in which in lieu of flat lists of descriptors we have a directed graph of descriptors and functional components connected by predicates.



Committing data definitions to a common ontology may not be a complete solution, but it should 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.

One part of the graph will specify how components interact among each other, this part of the ontology can be created and managed by scientists that specialise in this area, the leaf nodes of this section will be the traits that are measured in datasets; everything below these nodes should describe the different data representations that are captured in the field and this part of the graph should be managed by those generating these datasets.



The ability to record the functional relationships of objects will allow searching by inference, the ability to describe the data types will allow recording different representations of the same trait and the separation of these two areas will hopefully prevent conflicts of interest.

At the core of this data collection system is the ontology, which will be used both to document and tag traits as well as the structure of the hosted data.

The ontology

Four main objects constitute the ontology: the *term*, the *node*, the *edge* and the *tag*.

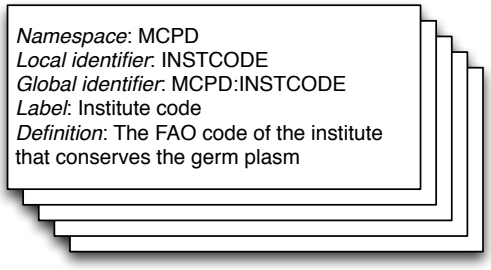
The starting point of the ontology is the *term*, which can be equated to a descriptor. Terms are objects that feature these main attributes:

- *Namespace*. The namespace is a reference to another term that represents a container or logical grouping for a set of identifiers, allowing for homonym *local identifiers*.
- *Local identifier*: The local identifier is a string that uniquely identifies the term among all the other terms that share the same namespace.
- *Global identifier*. The global identifier is the concatenation of the namespace and the local identifier, it represents the unique identifier of the term among all terms of all namespaces; this attribute is computed.

- *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 to refer to a term.
- *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.

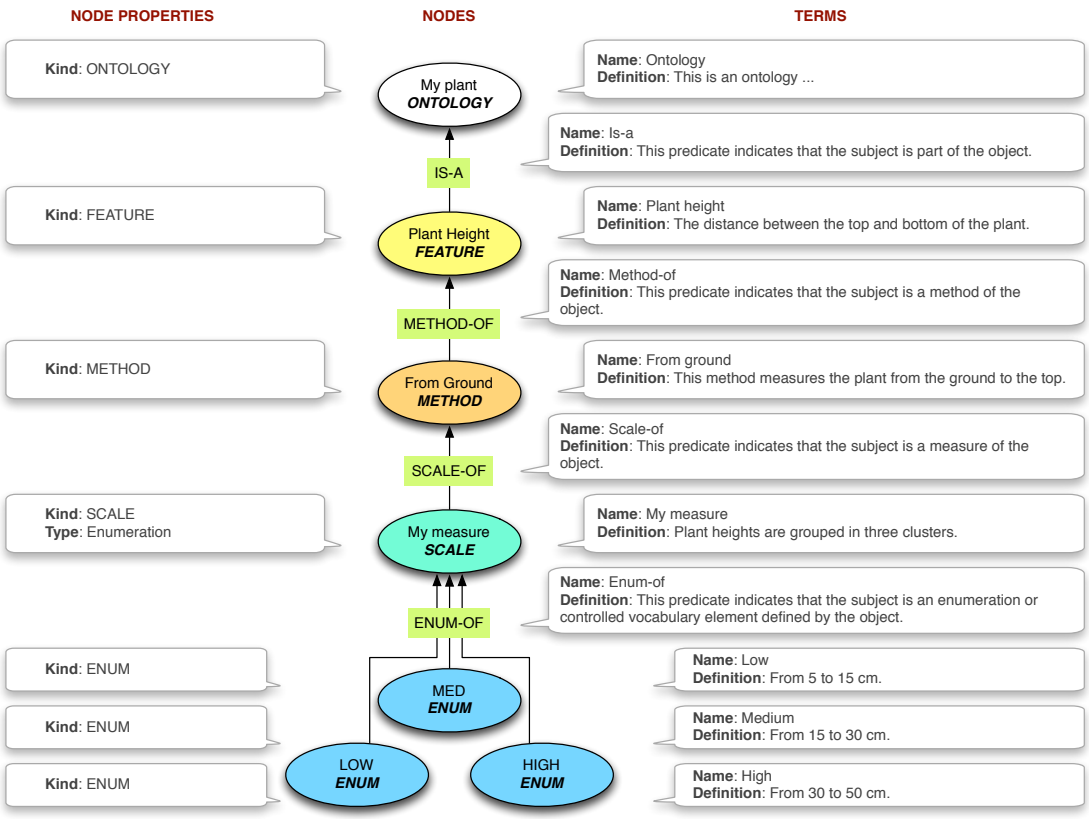
Besides the above properties, terms should be prepared to host any other kind of attribute: pictures, images, links, all that is needed to specify the meaning of the term in the most complete way.

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



Descriptors and terms are very similar, it should be very easy to transfer descriptors into terms, making the preliminary step of populating the ontology a simple task and providing the material with which a complete ontology could be developed.

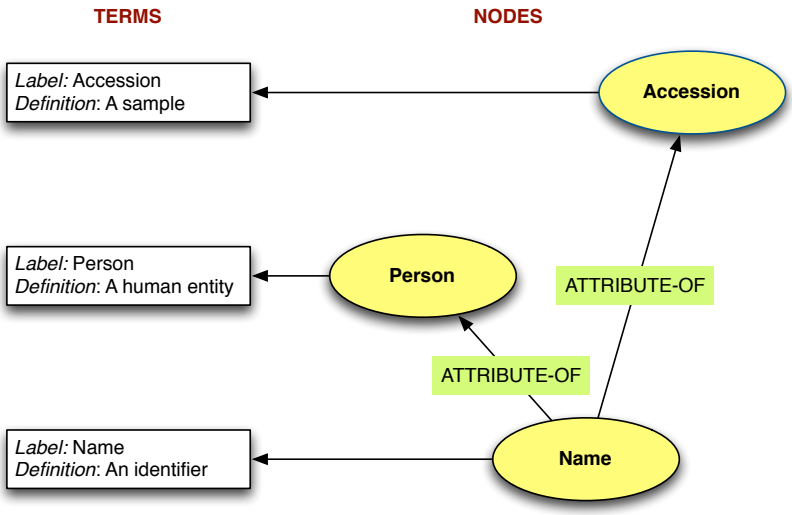
The power of ontologies, however, lies is in the ability to relate its elements. The structure of the ontology we are implementing is a *directed graph*, which is a collection of vertices each connected by a predicate in which the relationship has a direction.



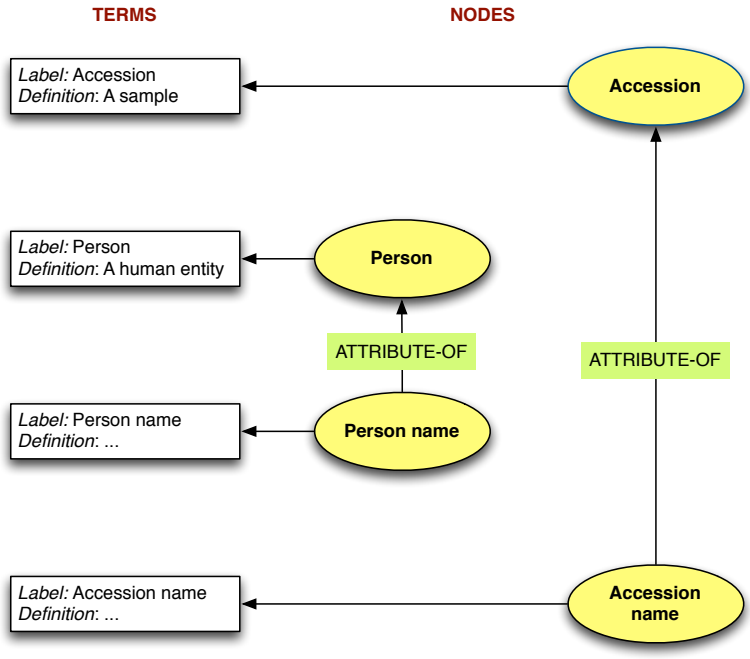
The vertices of such graphs are represented in this system by the *node*, which is a subclass of the term, in that it must reference a term. Nodes are essentially *terms in context*, or terms instantiated in a context. Nodes do not have required attributes except for the reference to the term they represent.

The interaction between nodes and terms is key in understanding how this ontology can be used and in deciding what kind of design to use.

Suppose that “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 then 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 properties 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 in language, in this case the node is an instance of a term.



The other strategy is to have only one node that can reference the same term, in this 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 in the term, since the term implies a lot about its context and the nodes would only exists to be connected, here the node and the term are the same entity.

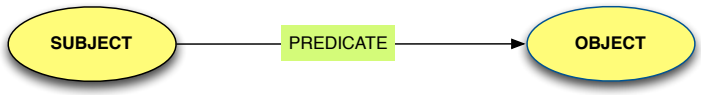


These two examples show the main principles that drive the design of this ontology: provide the ability to build semantic constructs, or tree structures.

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.

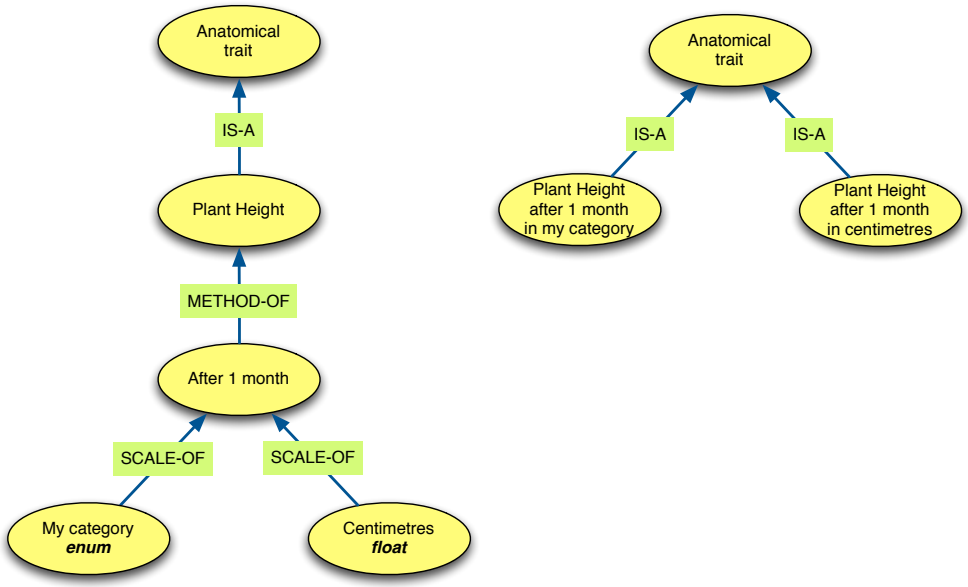
- *Object*. The reference to the *node* that represents the object or destination of the relationship.



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

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*.

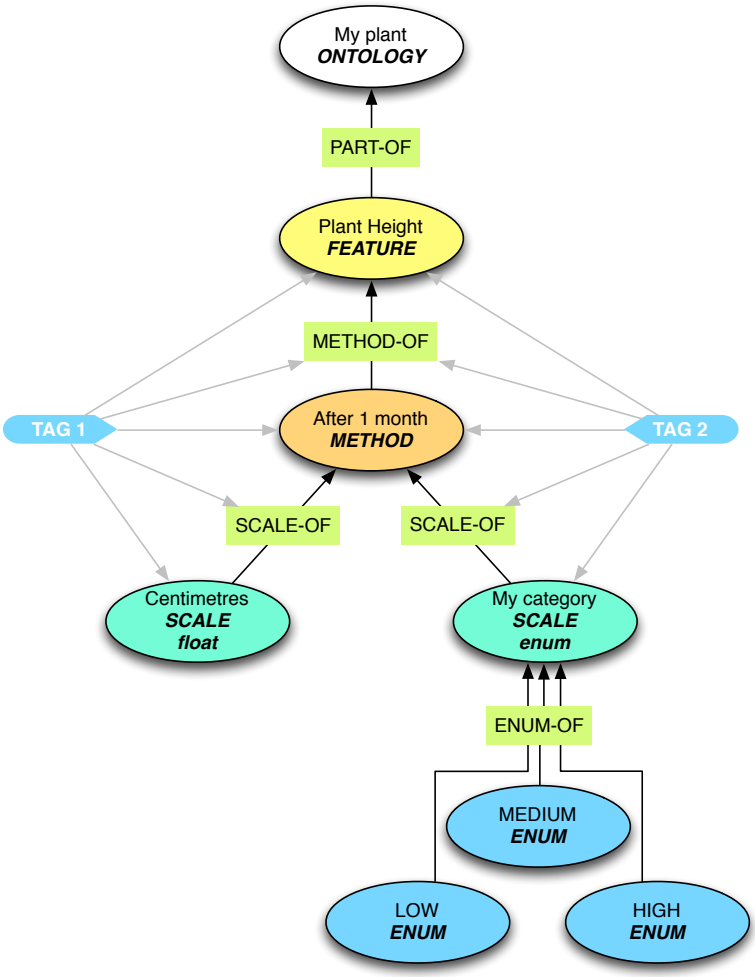
Ideally there should be a single node that describes a data element, however, data comes in many sizes and colours, "plant height" may be expressed in many ways and often it is not possible nor desirable to convert between these different formats.



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 similar "plant height" traits differing in little 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 each branch in the second will be stored as a different piece of data, but in the second case the structure will allow for a greater freedom when relating elements on the ontology.

Since a data element can be defined by a sequence of more than one vertex, 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.

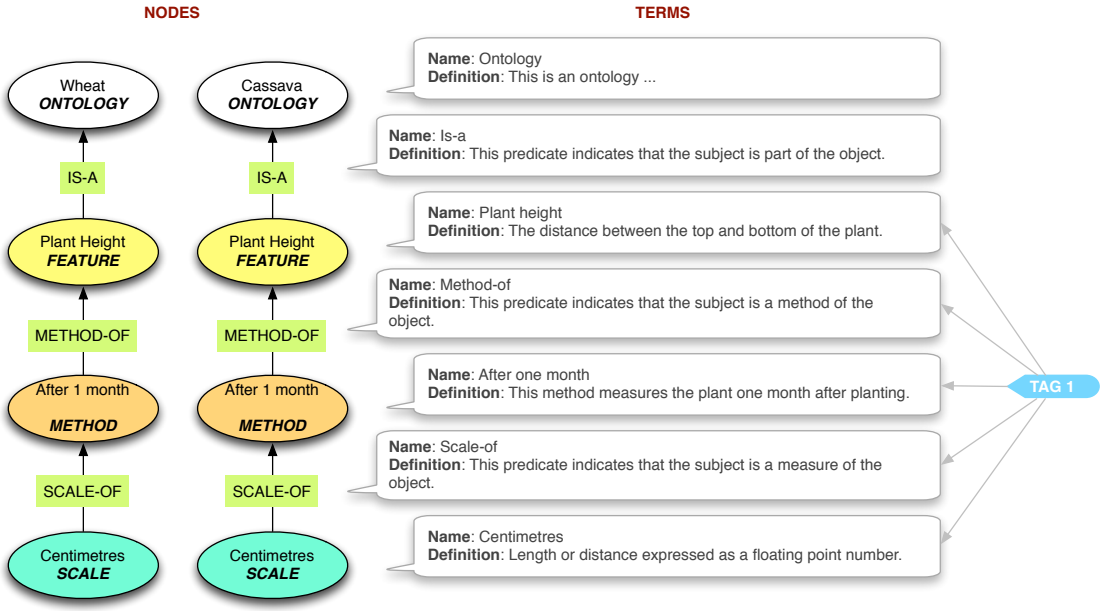
For instance, a dataset may have one set which represents the *plant height* measured *after one month* in *centimetres*, this would be represented in the ontology by a vertex which defines the *plant height* trait, a vertex that defines the *method* used to take the measurement, in this case after one 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 provided.



Tag 1: **Plant height/method of/after 1 month/scale of/centimetres**
Tag 2: **Plant height/method of/after 1 month/scale of/my category**

The advantage of the tag is that it can generate short identifier, leaving the freedom to use long URN identifiers in terms without impacting in the size 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 the same data field.

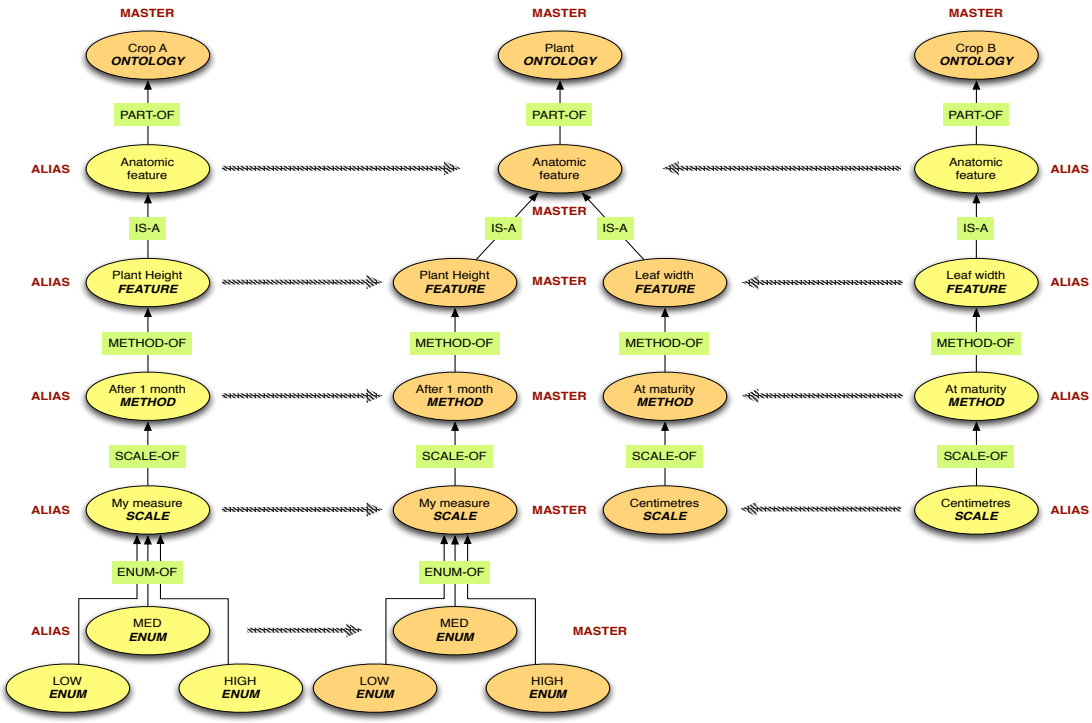


Usage

The combination and interaction of the above objects allows building a variety of different kinds of structures which can be helpful to make the user experience easier.

Nodes are not technically subclasses of terms, rather, they reference a term. This means that there may be several nodes referencing the same term: in this case one node will be considered a *master* and all the others will be considered *alias*, this feature can be used to create views and functional aggregations.

When you create an alias node a master node is automatically created and whenever you connect alias nodes, their relative master nodes will also be connected. This means that the alias structures will only feature their specific relations, while the master structure will feature the collection of relationships of all related aliases.



The first advantage of this feature is that you are able to create specific views within a larger structure: when navigating the “crop A” ontology you will only see the relationships that are relevant to that crop and, likewise, the same is true when you navigate “crop B”. This can be useful when dealing with terms of common use: for instance if you navigate to the centimetres term, without the ability to be locked in a view, you would see all the elements that are related to centimetres, which could make finding your way back rather difficult.

The other advantage of this feature is that the master structure is populated automatically, which can be interesting for checking the functional validity of the whole model. This also means that you have a choice of two strategies when building ontologies: use the master model, such as the plant ontology, as a template or guide for creating specific ontologies, such as crop ontologies; or load the specific ontologies and get the master model for free. The fact that tags reference terms rather than nodes is a guarantee that there will be data consistency between related ontologies.

Implementation

This system is not designed just to create ontologies, the main goal is to use these ontologies to annotate data, data collection is the ultimate goal of the system.

In order to accommodate graph structures and manage large quantities of data, we have chosen to move away from the traditional relational model and engine to embrace new technologies and workflows. The system is implemented using two main kinds of databases: the *document* database and the *graph* database.

Document databases store whatever data you provide them, there is one predefined identifier that is used to uniquely identify an object, but the rest of the data each unit stores is free. In this system we have selected *MongoDB* because of its performance, flexibility and scalability. The main duty of that database is to store the actual *data* and to serve as an *index* for the graph 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 storing large quantities of data and retrieving it, they are 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.

The combination of these two data storage engine types allows us to handle very large quantities of data with dynamic structures, provide extremely fast response times both for the data and the metadata and implement inference algorithms to make the system a very powerful portal.

Datasets are stored in a MongoDB collection, each element is considered a unit or record which may represent a multitude of different kinds of objects, such as an accession, landrace, crop wild relative or population. Each unit may hold any number of attributes which are identified by the *tags* we have described before. Additional data may be added to specific units, such as experiment measurements, since units do not have structure limitations.

The building principle of the whole system is that an attribute or property is identified by a *tag* which is constituted by a sequence of *terms* which define the property, and these terms may be navigated through a graph of *node* vertices that point to these terms and are connected by *edges*. This is strictly true also for the elements that comprise the ontology: all the attributes of terms, nodes, edges and tags are defined by tags. This

means that to create an attribute you must define it in the ontology, but it also means that any object in the system can have any attribute defined by the system. This guarantees a high level of consistency.

The system is functional as-is without the need of the Neo4j graph database, what this component adds is the ability to apply inference. The graph database allows using traversal algorythms that allow discovery of data through traits not directly connected to the data. A very simple example is finding data recorded by country through regions: if data has a country field I can at a later stage implement a regions ontology which relates its elements to countries, using traversal algorythms I can search for data using regions without these being directly related to the data.

The initial step is to implement the ontology and the annotation features, let users try the system and provide feedback that will be used to make the user experience as efficient and clear as possible; once the system will have been fine-tuned and the interface deemed user-friendly we can explore the possibilities inference offers us.