**Versioning, temporal modelling and concept evolution**

We will discuss here how to track changes to the conceptual model and how to represent temporal information and versioning. The running example is based on how classes and relationships changed over the past year in our conceptual model.

Consider the following fragment (or view) of the model:



Figure 1: version 1 at instant 0

Let’s consider this to be our baseline at the moment we start tracking changes, i.e. instant 0. This is version 1 of the model fragment.

At some point in time (instant T1) the notion of Sentinel Value Domains where introduced, both Enumerated and Described. The following figure shows the model at T1 after the update (for simplicity, we do not represent here the Substantive Value Domains):



Figure 2: version 2 at instant T1

Note that there is a new version of Instance Variable because the modeller decided that the new relationship to Universe granted a new version. This is version 2 of the model fragment.

Later on (at instant T2) the notion of Universe was revised and the former concept was split into a new version of Universe and the new notion of Population. At T2, the measures relationship in Instance Variable was changed from Universe to Population, and Represented Variable was associated to Universe. The next figure shows the model at T2:



Figure 3: version 3 at instant T2

This time the modeller decided that there is no need for new versions of Represented Variable and Instance Variable, only the relationships changed. (With the current Drupal implementation this is not an option since relationships are part of the source classes and thus a relationship change would trigger a new version of the class that owns it). This is version 3 of the model.

So far all three versions are completely independent of each other. There is no link between the two versions of Instance Variable or Universe, for instance, and the versions of the model fragment have to be used as a whole. Moreover, there is no way to know by just looking at the models that both Universe 2 and Population comes from the former Universe class.

The definition of version and the granularity at which changes are tracked have a heavy impact on the model as a whole. For instance, if the new relationship from Represented Variable to Universe required the creation of a new version of Represented Variable (which is in fact the case in the current Drupal), that change would have a cascade effect: we would need a new version of Instance Variable, because it would be extending the new version of Represented Variable, together with new versions of Enumerated and Described Sentinel Value Domains, because they would be associated to the new version of Instance Variable. The cascade effect could be quite large in a model with a high degree of connectivity between classes like ours.

In contrast, if we consider relationships (and even properties) as separate entities whose changes can be tracked separately from the classes they relate, then a change in a relationship won’t have an impact beyond itself. To support that we will need to have *intervals of validity* associated to both classes and relationships (and perhaps properties as well). A temporal model for our running example would be the following:



Figure 4: temporal model

Figure 4 contains the whole history of the changes described above. Each interval is given by a pair of time points. The initial time point in the model is 0 and the current time point is represented with the distinguished word *Now*. At instant T1 the new version of Instance Variable was created with interval [T1,Now] and the old version was changed from [0,Now] to [0,T1). Note that relationships have their own intervals, e.g. relationship has from Represented Variable to Universe is valid in [T2,Now]. Time intervals need to satisfy some simple consistency conditions [to be completed].

Note that all versions can be extracted from this model, e.g. version 2 would be the snapshot of the model at time T1, which can be constructed by keeping only classes and relationships whose intervals of validity contain T1.

On top of that model it’s possible to add specialized relationships to track the evolution of classes over time. For instance, we could use the *becomes* relationship to link successive versions of the same class or relationships between classes that have evolved from another one. The following is the resulting evolution model:



Figure 5: evolution model

In Figure 5 it is easy to see that both Universe 2 and Population have evolved from the former Universe class and that Instance Variable 2 has originated in Instance Variable. By traversing *becomes* relationships we can find semantic connections over time, i.e. across versions.

It’s also possible to define a *canonical* view of the temporal and evolution models. In a canonical model a new version is created only when the definition of a class changes – changes to relationships and properties are tracked separately and do not impact on the classes themselves. A canonical evolution model for our running example is the following:



Figure 6: canonical evolution model

In Figure 6, the two Instance Variable versions were merged into one because the only difference between them was the fact that the latest version has a relationship to Universe and later to Population. Now that information is represented in the relationship itself.

**Summary of Proposal**

The temporal model can be supported in the Drupal by the addition of two fields to each property and relationship — a *start date* and an *end date*. In fact this is in line with best practices both at StatCan (IMDB and Census) and in the NCI CDE Browser which is used widely across NIH in the United States. The values of those properties can be automatically generated in most cases based on the current interval (for existing classes) and the instant in which the update happened. (We’ll provide references to the relevant related work for these algorithms).

We might extend Drupal to present the temporal model visually. We might do this at the package level.

Versions at any granularity can be computed on the fly from the temporal model. Alternatively, we could have releases at the class granularity instead of packages or view releases. All of this would be reflected in the XMI.

Next we would apply rules to the XMI that we might use to version old views and create new ones. Each version of a view would have an expiration date. Those rules are not fixed. They might change over time.

Next we would create an RDF representation of the whole XMI. Using Protégé or a similar tool and the RDF as input, we might create first an evolution model and then a canonical evolution model. The evolution model can be used, among other things, to infer relationships between seemingly unrelated classes that have evolved over time. Both of these models might accompany each new publication. They would be part of the user’s guide for that release. These models could guide DDI users as they decide whether or not to “trade up” views from one release / publication to the next.