Versioning of Language Resources in Terms of CMD Document Instances

Kerstin Eckart & Jens Stegmann, IMS Stuttgart

Preliminaries: The question of how to address versioning in terms of CMD documents relates to all kinds of language resources that IMS Stuttgart provides (tools, web services, lexical resources, corpora, treebanks, databases, language models, parameter files etc.).In other words: although we only speak of 'resources' here, the term needs to be understood in a very broad sense that incorporates all of the above.

Furthermore, if possible we would like to see a solution that is applicable not only to versioning, but also to the representation of other structural phenomena of interest like part-whole relationships (e.g., wrt to parts of corpora or collections). Of course, what is common to both kinds of phenomena is the problem of expressing relations in terms of CMDI means. A 'unified' solution, if one can be found and agreed upon, would be much more straightforwardly exploitable at the level of applications that build upon CMD metadata records (-> think 'Ranking' of resources in the VLO, for example).

CMDI is such a flexible approach and many of the problems that people have seem to be related to the fact that there are many options for doing the same thing (here: representing relations) in terms of CMD documents. The list of options for our problem at hand includes at least the following: (i) particular elements as defined in specific profiles/components, (ii) generic elements like IsPartOf (we do not have IsVersionOf yet, but that would at least be possible), (iii) general linking mechanisms like ResourceProxy lists. While some of those mechanisms work rather bottom-up (cf. (ii)), others are rather top-down-oriented (cf. (iii)) or they could work either or both ways (cf. (i)). Finally, these mechanisms can also be combined to create all kinds of hybrid solutions.

In the following, we will frame our discussion along another aspect of interest and sketch two options of implementing versioning in terms of CMD-related technicalities along a (iii)-approach in the above sense.

Strategy I: All versions of a resource are described via one monolithic MD instance.

Possible implementation: Each version has its own `ResourceProxy` with a pertinent `id` value in the `Resources` section. The `ResourceRef` may reference 'primary' data (`ResourceType` would be `Resource`, not `Metadata` ; other options include landing pages etc.) via a URLified handle PID. In the `Components` branch of the CMD document, `xml:ref` attributes will specify the pertinent version(s) by referencing the `ResourceProxy id` values as appropriate, such that version-specific information can be expressed within the scope of respective components or elements.

Advantages: This option **(i)** does not introduce references between different MD instances, and **(ii)** its relation semantics for version relation could already be properly described via the `ResourceRelationList` mechanism as of CMDI 1.2. With regard to storage and harvesting this option is **(iii)** analogously representable in the structures of (Fedora-based) repositories in terms of digital objects, pertinent data streams and the OAI-PHM protocol. **(iv)** There will be little redundancy in the information describing the different versions, and **(v)** when one version is found in the VLO the user automatically finds all versions.

Disadvantages: (i) All components, maybe even additional elements, can appear for different versions, i.e. the cardinality semantics of the profiles change. So the profiles in use need to be revised; restrictions to cardinality 1 for components are no longer possible. **(ii)** For each new version, the MD instance is changed. If an error is introduced, MD information which has been correct and therefore available before might become corrupted und unavailable. **(iii)** An XML file describing many versions can be rather confusing, moreover as information about a specific version will often be scattered over the entire components branch of the MD instance.

Open questions: (i) One version may consist of more than one `ResourceProxy`, so which id will be referenced in the components? (Use `ResourceRelationList` to identify the 'main' `ResourceProxy`

for this version? Reference lists of ids?) **(ii)** How does the VLO represent version-specific MD? This applies to all VLO representation layers, because a resource might even have version-specific `ResourceNames`. It might also be the case that only one version of the resource fits the users search criteria. Do the users have to find out themselves which version does?

Strategy II: Each version of a resource is described via its own MD instance.

Possible implementation: The different MD instances for particular versions will now need to be related to each other, i.e. linked and bundled. One possibility: by means of an additional 'meta' MD instance representing the whole resource in an abstract sense. The latter could have `ResourceProxy` elements with `ResourceType` being `Metadata` and `ResourceRef` values that reference the MD instances describing the particular versions (again, best via URLified handle PIDs). Only the latter might make reference to primary data, landing pages or relevant parts of a particular version (via ResourceProxy and the ResourceType being Resource, for example), but 'intermediate levels' are possible, too.

Advantages: (i) Changes would rather be made to the headers, not so much to the components of existing records. **(ii)** Particular MD instances would be more lucid as compared to the 'one monolitic MD instance' approach above. **(iii)** It may be easier for users to find information relating to particular version of a resource of interest. **(iv)** Multiple ResourceProxies per version would be rather unproblematic (bundled via version-specific MD instance).

Disadvantages: (i) The setup is more complicated. For example, we might need some kind of inheritance mechanism in the sense of MD instances for versions inheriting the information already represented at the abstract resource level. **(ii)** There may well be changes over time regarding what kind of information is version-specific vs. what is true for all versions of a resource (and hence representable on the meta MD instance level vs. particular version level). **(iii)** We might end up with a lot of redundancy in our MD instances. This would make maintenance work and corrections very difficult and error-prone. **(iv)** No information on branching version threads representable (?), e.g. dependency and constituence versions of a treebank. **(v)** The mapping/treatment in terms of the structure of a Fedora-based repository is much less straightforward.

Open questions: (i) How would the VLO make the relation semantics explicit? Forward links to specific versions from the abstract resource representation, backward links from specific versions? **(ii)** Given the state of CMDI 1.2, the semantics can not be handled adequately via RelationsList as the moment. **(iii)** Which options do we have regarding the mapping to Fedora Commons Digital Objects and Data Streams? This has to work with the OAI-PMH web service. It seems that we would need to have seperate digital objects for particular versions, which is rather cumbersome. **(iv)** Ranking in VLO might be a problem; but possible solution via PageRank. **(v)** May still require changes to the profiles that the centers use at the moment.

Summary: We think that both strategies have their relative pros and cons and that it is difficult, if not impossible, to qualify one as being superior to the other. There seem to be relative advantages and disadvantages to both approaches, as well as possible difficulties in implementing either approach. Indeed, while discussing the issues adressed here, our own intuitions have shifted quite a bit. Personally, we think that we could bring about both strategies as well as implementations that would deviate from what we have sketched here. (There are definitely alternative ways of going about both options; hybrid solutions are possible, too. We have not even touched upon all aspects of interest. For example,, assignment of PIDs has played no role in our discussion here.)

The most important suggestion from our point of view: that the CLARIN-D consortium agrees on a uniform way of treating the phenomena under discussion. If possible: uniform across the different centers as well as across the different domains of application (versioning and part-whole relationsships). Otherwise, we may end up in a situation where it will be very difficult to build useful applications that exploit the wealth of CMDI metadata records provided and the elaborate structural descriptions that they may contain.