CMDI and granularity

Identifier	CLARIND-AP3-007		
AP	3		
Authors	Dieter Van Uytvanck, Twan Goosen, Menzo Windhouwer		
Responsible	Dieter Van Uytvanck		
Reference(s)			
Version	**Date**	**Changes by**	**State**
1	2011-01-24	Dieter Van Uytvanck	Discussion document
2	2011-01-25	Menzo Windhouwer	Small fixes
3	2011-01-27	Dieter Van Uytvanck	Changes to introduction
4	2012-02-15	Dieter Van Uytvanck	Minor corrections, sections 6 and 7 added

1 Introduction

As this document tries to give some guidelines to those who want to setup a repository in a CLARIN center, it makes some presuppositions that might not be necessarily valid in other contexts:

- A distributed setup of tens of centers, where most have an own repository.
- The resources are web-accessible.
- Metadata for the resources is stored in the CMDI format, where each center might use its own resource-specific profile(s).
- All metadata is harvested via OAI-PMH and afterwards included in:
  - search engines
  - portals (like the VLO)
- The creation of the metadata is fully under control of the center, once harvested the metadata files should be fully expanded. (E.g. no XIncludes need to be resolved anymore at the side of the harvesting party.)
- This is not an academic thought exercise but the backbone for a research infrastructure that needs
  - to be implemented on relatively short term
  - to be robust
  - to scale well
  - to be usable with good user interfaces by non-technical users

2 Order of magnitude

How many metadata records should a center make, at most?

Theoretically there are no upper limits. In practice there are 2 main factors restricting the number of metadata descriptions:

- Technical ones, like the maximum number of supported entries by the repository system (e.g. for Fedora: 100 million)

- Cognitive efficiency for the end-user. Factors that nevertheless justify a higher amount of metadata instances are:

    - A high informational value of each individual record (low overlap of the descriptive fields)
    - An elaborated structure (a deep hierarchy of subsets).
    - Citability: a high probability that someone cites this metadata entry in an academic publication

In practice, **the number of metadata records should never exceed 1 million per center**. Even browsing efficiently through "only" 100.000 records already requires fine-grained metadata descriptions and a deep and well-structured hierarchy with many nodes.

Some counts for large collections[1] hosted at CLARIN centers indicate that on average 10.000 to 30.000 records should be enough to describe the resources in a detailed yet useful way:

Collection name	Number of metadata records
CHILDES corpus	28595
TüBa-DDC	19059
DoBeS collection	18490
TalkBank	14243
Corpus Spoken Dutch	12768
Bavarian Archive for Speech Signals	7227
Oxford Text Archive	4419

Even centers that host multiple collections should not exceed 1 million records. At the time of writing the largest number of metadata records per center (TLA repository) is about 125.000.

3 Metadata versus (annotation) data

The border between what is metadata and what is (annotation) data is often vague. In general metadata is believed to be valid and stable for the whole resource it describes. Some cues that information is rather annotation data, which should not be handled as metadata, are:

- The data benefits from being displayed in a tailored application (e.g. a concordance viewer).

[1] Source: Virtual Language Observatory, http://catalog.clarin.eu/ds/vlo/?wicket:bookmarkablePage=:eu.clarin.cmdi.vlo.pages.ShowAllFacetValuesPage&selectedFacet=collection

- The data contains the most elementary linguistic information (text fragments for a text corpus, transcriptions for a speech corpus) or clear annotation information (tags, translations, prosody, etc.).
- The data could be included verbatim in an academic publication (metadata generally is not a part of the research subject).

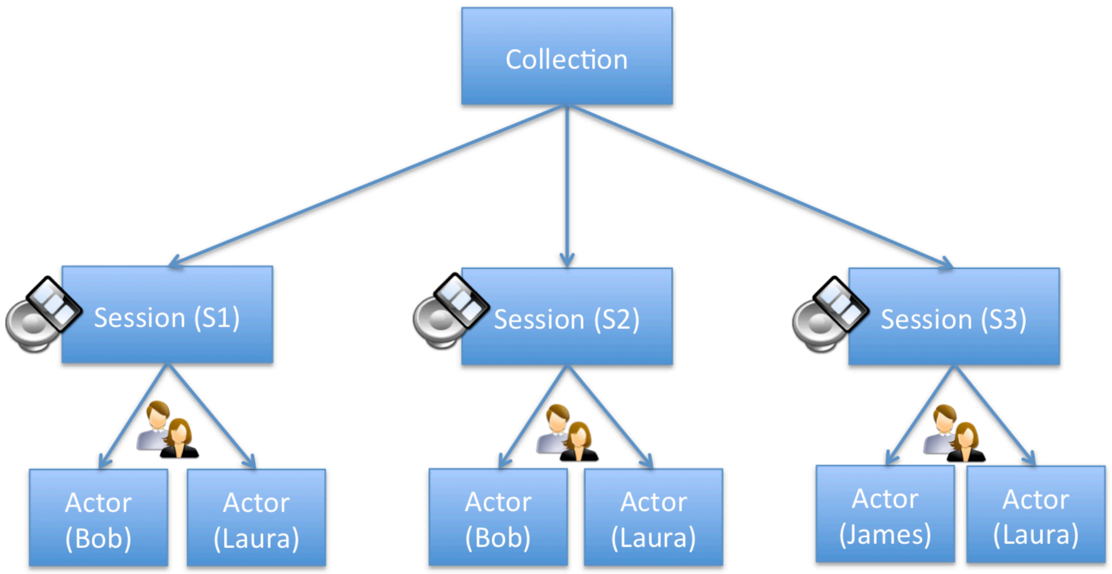
An **example** of a granularity level that goes rather to the **annotation** side of the spectrum is **information about a single sentence in a text corpus** (number of types/tokens, annotations, ...)

4 Cyclic hierarchies

With the degree of freedom that ResourceProxy links provide, it is not so difficult to create cyclic links: a newspaper instance could link to an article instance and the article could in turn (with a ResourceProxy of the type metadata) link to the newspaper. The uplinks however do not add any real information: they could also be derived automatically while traversing the tree. In general, pointer cycles are an annoying phenomenon to deal with. So while acknowledging that we cannot be 100% sure that they will not occur, **we strongly recommend against any cyclic linking with ResourceProxy links**. CMDI metadata is not intended to be used as a graph storage format.

5 Normalization

The diagram below illustrates a typical case of re-occurring entities in a data model. A *collection* (e.g. a speech corpus) contains items (called *sessions* in this example), which in turn contain links to the resources (the sound files) and to information about the persons who were recorded (*actors*).



In CMDI, it is clear that the links from the sessions to the sound recordings will be established via a ResourceProxy (with the ResourceType "Resource"). Likewise, the collection will use a ResourceProxy (with the ResourceType "Metadata") to point to each of the subordinate Sessions.

However, how to deal with the actor information? Here the main issue is related to updating information of an Actor (name, birth date, formant features, etc.): if a field is updated, the change needs to be reflected in all places where there is a reference to that actor.

In relational databases this problem is dealt with by normalizing the information: each actor is stored only once in a specialized table and all sessions are referring to that single instance's primary key with a foreign key. However when modeling this information in the CMDI/XML world, another approach is needed. **Below we list some possible strategies and argue why the last one mentioned (section 5.4) is the best approach.**

5.1 One monolithic XML document, normalizing using ref's and id's.

In this scenario all information, from the collection down to the actors is stored as one large XML document. Actors get a (for that document) unique identifier and are referenced in the session fragments with an XML reference. This works well but threatens the scalability: in the case of a large corpus (tens of thousands of sessions) the XML file becomes huge.

5.2 Modular XML documents, normalization using ResourceProxies.

Here each information unit (collection, session, actor) is represented as a singular XML document. For each actor (Bob, Laura, James) 1 CMDI instance is created. Each session points to the relevant actor CMDI documents via a ResourceProxy (of the type Metadata). Additionally this ResourceProxy could be linked to by a component for an actor (with a ref attribute).

This approach works well at the metadata creator side but poses some serious problems at the exploitation/search side (everything happening after the metadata files were harvested):

- How to deal with arbitrary depth levels of nesting? An actor could again contain a pointer to a deeper level, etc. This seriously complicates all data processing (the pointers need to be dereferenced) and the creation of search interfaces.
- Higher risk of pointer cycles.
- A higher risk towards breaking referential integrity, as the link between the session and the actor might break (using local URLs is not optimal, using PIDs in an optimal way would require constant resolution when processing the CMDI files).
- Full validation also becomes problematic as there is no straightforward way to validate that the ResourceProxy points to an instance of the expected CMDI profile. (Actually this is also true between Collection and Session, but the archive backbone is much less variable then a fully normalized approach.)

5.3 Modular XML documents, no normalization.

With this approach all reoccurring metadata is reduplicated: Session 1, 2 and 3 will all include the full information about the Actors in each session. This pushes the full responsibility for the update issues towards the metadata creator side.

If the backend used by the hosting center is a relational database, there is no problem, as serializing the metadata records to XML is trivial and the updates that took place to a single record in the database are automatically spread all over the CMDI files.

However, if the center uses CMDI files as the backend for the repository the update issue becomes an annoyance. Certain software features (like the table batch edit option in Arbil) might alleviate this. For large collections the updates to reoccurring entities remains non-trivial nevertheless.

5.4 Modular XML documents, (possibly) normalization using XIncludes.

This option brings the responsibility for the normalization, the referential integrity and the normalization to the side where it naturally belongs: that of the metadata creator. It is basically the same as the previous case, except that the reoccurring entities are included from the local XML files with the XInclude standard.

A session might contain an inclusion of an actor XML snippet:

```
…
<Actor>
        <xi:include href="actors/bob.xml" parse="xml"/>
</Actor>
…
```

Then it is up to the metadata creator to ensure that the snippet file actors/bob.xml is up to date. In any case it solves the update issue, as long as the effort is made to use XInclude instead of just copying the information.

In the case of Arbil using this approach, although still requiring some extra work, seems to go quite well with the already existing concepts:

- The inclusion could take place during the export operation. The consumer of the (exported) metadata files does not need to be aware of what has been included, nor does he need to worry about referential integrity.
- The snippets could become some kind of "dynamic" favorites: where the normal favorites make a copy of an entity, using a dynamic favorite would mean including a snippet. If the user changes something in the favorite entity later on, the change will be reflected automatically into all CMDI files that include it.

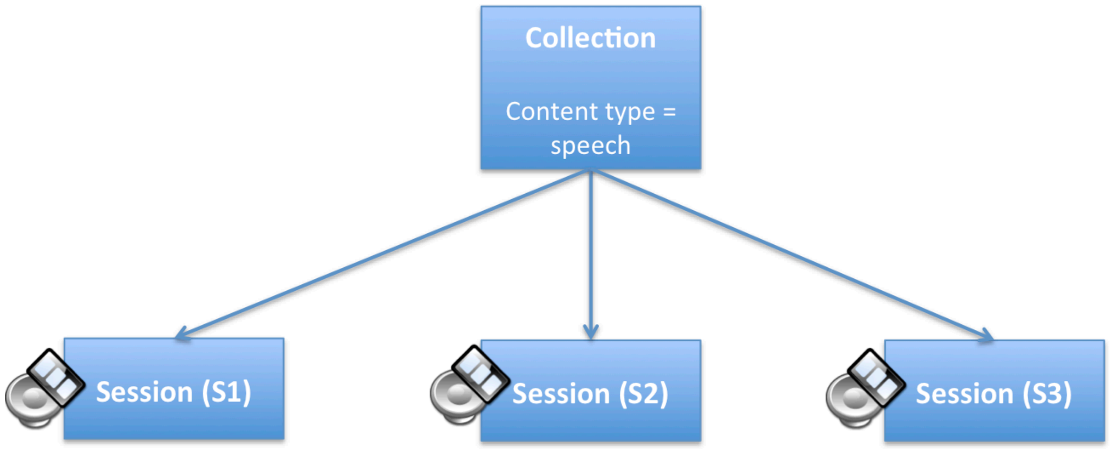
At the same time it is fully compatible with the non-normalized approach (as there is no obligation to use includes).

All in all modular XML documents (possibly using normalization by XIncludes) seems currently to be the most advisable way of modeling reoccurring information with CMDI.

However, it should not be seen as an integral part of the CMDI format – it is just a convenience at the side of the metadata creator. Only fully XSD-compatible CMDI files (including resolved XIncludes) should be offered for harvesting.

6 Metadata inheritance[2]

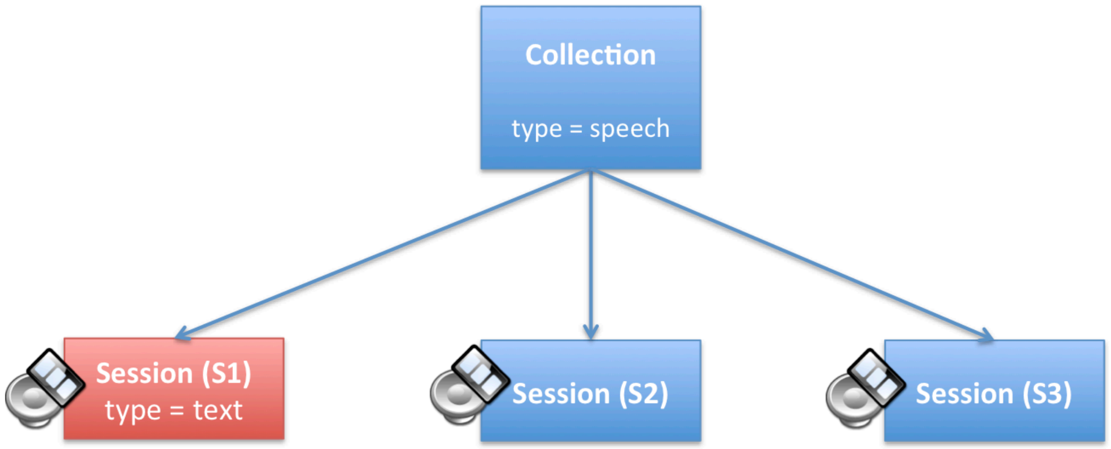
Creating hierarchies in CMDI can be done, as explained before, through the use of *ResourceProxy* elements. Especially when modeling collections the question arises in how elements that pertain to the whole collection should be expressed. Consider the following example:



A specific field, in this case that the collection exists of speech data, is applicable to all lower-situated CMDI instances. Does that mean that it is not necessary or advisable to repeat this element at the lower levels (S1, S2, S3)? In theory one could use inheritance as a model for metadata hierarchies: all metadata fields do also apply for the lower levels. However, there are some cases where such inheritance would lead to problems:

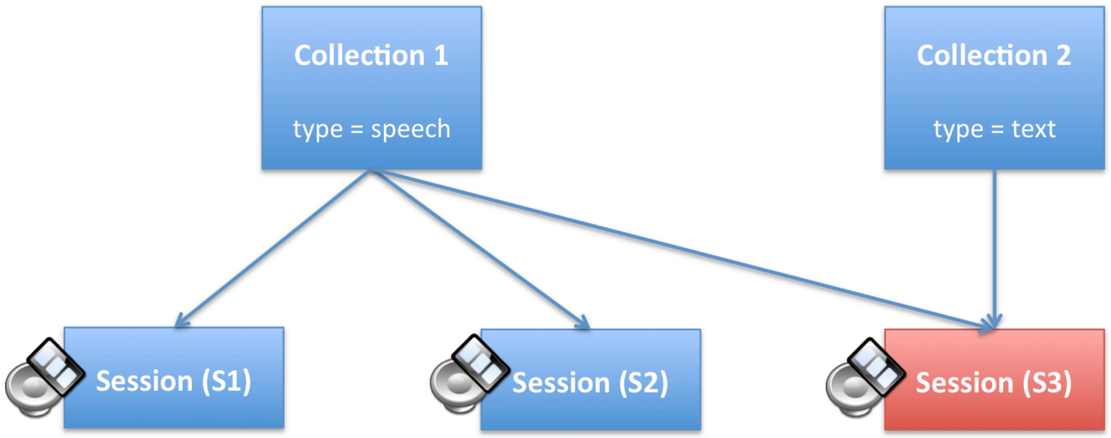
[2] It should be noted that, as is the case for the rest of this document, this section is about inheritance within CMDI *instances*. It does not cover CMDI *components* or *profiles*.

6.1 Simple inheritance: conflicting elements



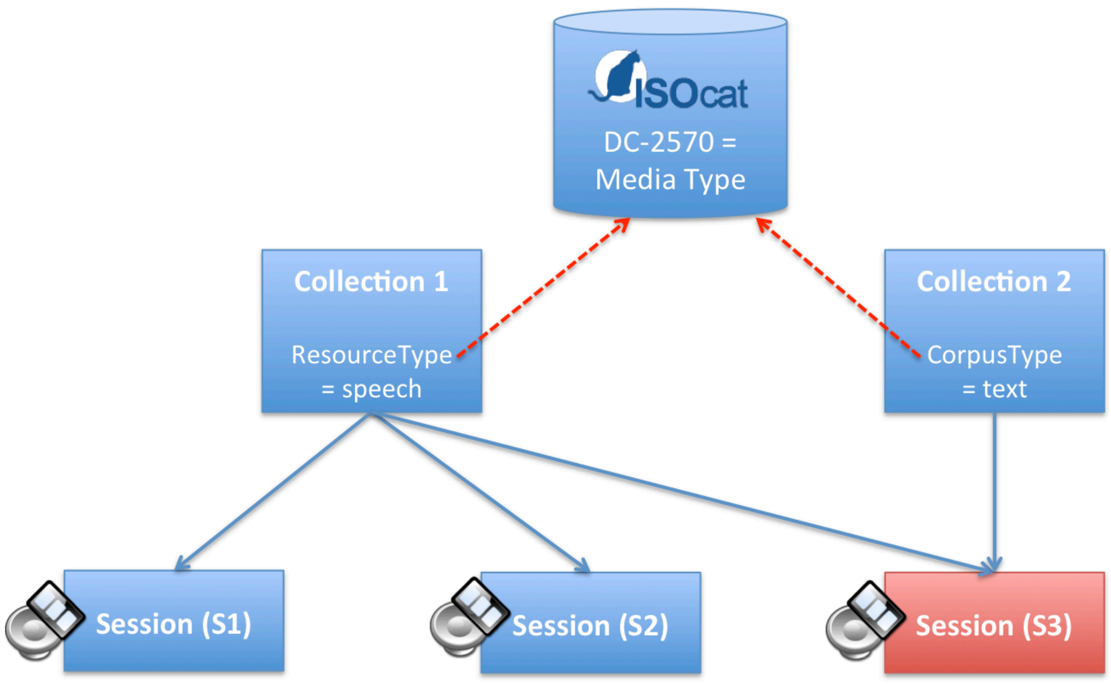
Here, the type of S1 conflicts with the inherited type from Collection. Stating that the inherited element always “loses” from the own element can clearly solve this case.

6.2 Multiple inheritance: conflicting elements



Here, contradictory elements are inherited from multiple parent nodes. Rules that solve this could be thought of, but with deep hierarchies things can become very complex.

6.3 Semantic inheritance conflict



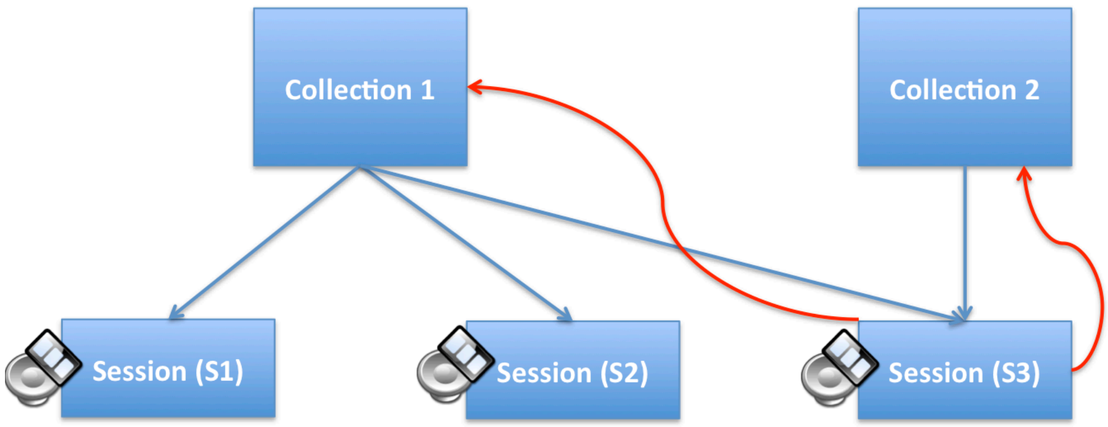
Same case as above but with one more complication: syntactically S3 could have both ResourceType = speech and CorpusType = text but semantically these elements result in a conflict. Here too, a set of conflict-resolving rules could be made, but even more than in the previous case it will result in an extremely complex system.

6.4 Conclusion

Although one could think of all kinds of inheritance systems, dealing with it in a good and transparent way would make the CMDI search and interpreting software too complex. **Hence, for now no metadata inheritance will be supported in the CLARIN-provided CMDI exploitation stack.**

7 Bottom-up hierarchy links ("IsPartOf")

The CMDI header part features an IsPartOfList element that makes it possible to provide uplinks to nodes higher in the hierarchy. In the following example S3 points back to its parents (red arrow), Collection 1 and Collection 2:



In CMDI this is expressed as follows with the IsPartOf elements:

```
...
<Resources>
      <ResourceProxyList/>
      <JournalFileProxyList/>
      <ResourceRelationList/>
      <IsPartOfList>
         <IsPartOf>http://foo.eu/Collection1.cmdi</IsPartOf>
         <IsPartOf>http://foo.eu/Collection2.cmdi</IsPartOf>
      </IsPartOfList>
</Resources>
...
```

This functionality was originally thought of as a mechanism to provide explicit inheritance (if A IsPartOf B it also inherits all elements from B), but because the inheritance mechanism turned out to be utterly troublesome (see the previous section), this idea was abandoned.

This means in practice that anyone can freely use the IsPartOfList element but that there is no guarantee[3] that the CLARIN CMDI exploitation software will be able to use or interpret it (e.g. for providing backlinks or inheritance of metadata elements).

[3] At the time of writing there is no exploitation software at all that supports the IsPartOfList.