CMDI 1.2: Improvements in the CLARIN Component Metadata Infrastructure

Twan Goosen[1] **Menzo Windhouwer**[2] **Oddrun Ohren**[3]
Axel Herold[4] **Thomas Eckart**[5] **Matej Ďurčo**[6]
Oliver Schonefeld[7]

[1]The Language Archive, [2]The Language Archive - Meertens Institute, [3]National Library of Norway, [4]Berlin-Brandenburg Academy of Sciences and Humanities, [5]Leipzig University, [6]Institute for Corpus Linguistics and Text Technology, [7]Institute for the German Language

twan@clarin.eu, menzo.windhouwer@meertens.knaw.nl, oddrun.ohren@nb.no, herold@bbaw.de, teckart@informatik.uni-leipzig.de, matej.durco@oeaw.ac.at, schonefeld@ids-mannheim.de

Abstract

This article reports about the on-going work on a new version of the metadata framework Component Metadata Infrastructure (CMDI), central to the CLARIN infrastructure. Version 1.2 introduces a number of important changes based on the experience gathered in the last five years of intensive use of CMDI by the digital humanities community, addressing problems encountered, but also introducing new functionality. Next to the consolidation of the structure of the model and schema sanity, new means for lifecycle management have been introduced aimed at combatting the observed proliferation of components, new mechanism for use of external vocabularies will contribute to more consistent use of controlled values and cues for tools will allow improved presentation of the metadata records to the human users. The feature set has been frozen and approved, and the infrastructure is now entering a transition phase, in which all the tools and data need to be migrated to the new version.

1 Introduction

Component Metadata Infrastructure (CMDI) has been one of the core pillars of CLARIN since the beginnings of this initiative (for an overview, see Broeder et al., 2012).

It established means for flexible resource descriptions for the domain of language resources with sound provisions for semantic interoperability weaved deeply into the data model and the infrastructure to overcome, in a great extent, the rule of metadata schism it set out to combat. Based on this solid grounding, the infrastructure accommodates a growing collection of metadata records. The development of the joint metadata domain both in number of records and diversity of profiles is proof for the success of the model and the infrastructure as such. Currently, at version 1.1 of the CMDI specification, there are 170 public profiles and over 1,000 public components defined. The CLARIN OAI-PMH harvester[1] periodically collects records from some 60 providers in more than 80 different profiles, almost 1 million as of March 2015.

However, in the first five years of its intensive usage by the CLARIN community naturally a number of design issues have arisen that need further attention. Therefore a dedicated task force consisting of developers and metadata experts from multiple CLARIN centres was established to work towards a successor to CMDI 1.1 based on the existing paradigm. After careful analysis, the task force worked out a proposal for a number of small but important changes and additions to the CMDI model leading to CMDI version 1.2. In April 2014, the Standing Committee for CLARIN Technical Centres approved the proposal, which meant that work on the implementation could begin.

The changes address the following aspects: lifecycle management, structure of the model and schema sanity (namespace issues, consistency of the meta model, attributes, mandatory/optional elements),

[1] http://catalog.clarin.eu/oai-harvester/

use of external vocabularies and cues for tools. They are described in detail in sections 2 and 3. The work on the model is accompanied by a comprehensive transition plan covering the conversion of existing data and adaptation of existing tools using CMDI data, described in section 4. Finally, section 5 details still open issues and further plans for the CMDI model and joint metadata domain.

1.1 Short description of the Component Metadata Infrastructure

It is important to understand that CMDI is not "yet another" static metadata format, but rather a meta-model, a framework allowing for the creation and use of custom schemas. It relies on a modular model of so-called metadata components (Broeder et al., 2010), which can be assembled together, to foster reuse, interoperability and cooperation among metadata modellers. Components are used to group elements and attributes, which can take values, and also other components. They are stored and maintained in the Component Registry.[2] A metadata modeller selects or creates components and combines them into a profile targeted at a specific resource type, a collection of resources or a project, tool or service. A profile serves as blueprint for a schema for metadata records. CLARIN centres offer CMD records, describing their resources, to the joint metadata domain.

Due to the flexibility of this model, the metadata structures can be very specific to an organization, project or resource type. Although structures can thus vary considerably they are still within the domain of metadata for linguistic resources and thus share many key semantics. To establish these shared semantics CMD components, elements and values can be annotated with links to concepts defined in external concept registries.[3] This allows generic tools that operate on all the CMD records in this domain, like the metadata catalogue Virtual Language Observatory (VLO),[4] to overcome differences in terminology as well as structure by operating on this shared semantics layer.

1.2 Related approaches

To position the work on CMDI in the broader landscape and to allow for comparison to the approach adopted by CLARIN we will briefly review a number of alternative approaches taken by similar or related initiatives. In the sister initiative DARIAH[5] no one common solution for resource description and discovery has been adopted yet, however a candidate solution developed within DARIAH-DE (Heinrich & Gradl, 2013) pursues an approach not too different to that of CLARIN: Repositories or collections are registered in the Collections Registry (roughly corresponds to CLARIN's Centre Registry), subsequently harvested via OAI-PMH into the Generic Search, a faceted search engine (corresponds to VLO). The schemas exposed by individual repositories are recorded in the Schema Registry where a mapping (crosswalks) can be defined. The mapping information is used for on-the-fly expansion of the queries. The main difference to the CLARIN approach is how crosswalks or semantic interoperability is achieved, namely via pair-wise mapping between the schemas, whereas in CMDI the concept links serve as pivot points, represent a separate semantic layer to ground the schemas onto, allowing for more efficient mapping (dozens of profiles share the same basic data categories). Furthermore, the DARIAH-DE approach has not yet been adopted on the European level. An alternative proposal within DARIAH is based on Semantic Web technologies: repositories provide lightweight description of their collections via RDFa[6]-annotated web pages, which are being crawled and indexed in a semantic web application.[7] This approach reflects the general tendency especially in the humanities towards adoption of semantic web technologies for resource description. Acknowledging the integrative power of the Linked Data paradigm, the CMDI developer team proposed a complete expression of CMD data (from meta model to the instances) as RDF (Ďurčo & Windhouwer, 2014), which was implemented by CLARIN-NL.[8]

[2] http://catalog.clarin.eu/ds/ComponentRegistry/

[3] The primary registry used until recently, the data category registry ISOcat, has been replaced with the CLARIN Concept Registry in December 2014.

[4] http://www.clarin.eu/vlo

[5] https://www.dariah.eu/

[6] Resource Description Framework in Attributes, see http://www.w3.org/TR/xhtml-rdfa-primer/

[7] http://rechercheisidore.fr

[8] https://catalog.clarin.eu/ds/cmd2rdf and https://github.com/TheLanguageArchive/CMD2RDF

The META-SHARE initiative,[9] on the other extreme, imposes one large schema for all resource descriptions with many optional parts and some specialization for the main resource types. Nevertheless it also adopts the basic idea of component-based modelling and concept-based semantic mapping. The principal compatibility has been demonstrated by expressing the META-SHARE schema as *resourceInfo* profiles[10] within the Component Registry.

The European DASISH project[11] delivered a metadata catalogue[12] collecting resource descriptions from the three research infrastructures CLARIN, DARIAH and CESSDA.[13] Given the great disparity of the encountered formats and the goal being a catalogue with a broad coverage but only a small fixed set of facets, individual fields in the schemas were manually mapped to the facets. This work was also strongly inspired by the CMDI approach using concept links for mapping where possible.

2 New CMDI functionality

2.1 Lifecycle Management

There is no definite metadata representation for any given language resource in terms of a single fixed CMDI component or profile. Instead, metadata modellers often encounter situations that make it necessary to adapt or amend existing metadata models. Typically, such situations are caused by needs of data providers that supply more detailed metadata than any of the existing components cater for. To ensure formal and semantic persistence of referenced metadata components, typical applications of CMDI will disallow changes of those components once they are made publically available.

Within the current version of CMDI, there is no possibility to denote the lifecycle status of components, e.g. by marking a component as deprecated and/or superseded by another component. CMDI 1.2 will provide lifecycle management support for components based on four additional header elements: *Status*, *StatusComment*, *Successor* and *DerivedFrom*. These elements appear as direct children of */CMD_ComponentSpec/Header/*.

The mandatory *Status* field is used to record the current lifecycle phase of a component. Allowed values comprise “development”, “production”, and “deprecated”. Infrastructures exploiting the CMDI framework need to ensure that only transitions from “development” to “production” but not vice versa are allowed on the grounds that components should not leave a state that denotes immutability (“production”, “deprecated”) once they reached it.

Each component can optionally be annotated with a *StatusComment*. This field can be used to record the reasons for status changes, reasons for the derivation of a new component from an existing one or other useful information regarding the component’s status in human-readable form.

The optional *Successor* element can be used on deprecated components to specify, if applicable, the URI of the component that should be used instead. Often this will be an updated or improved version of the original component. It is not necessarily a derivative in a technical sense: the successor can be a component created from scratch or another already existing component that represents a different metadata scheme which is meant to replace the scheme in the original component. As the *Successor* field holds exactly one URI, only the direct successor of a component can be specified. Note however, that succession is a transitive relation. Therefore it is possible to construct a complete chain of succession by traversing components via their *Successor* fields.

The URI specified in the optional *DerivedFrom* field allows for the reconstruction of a component’s genesis in relation to other components. Derivation in the context of CMDI is considered in a purely technical sense of copying a component and modifying it independently from the original component. As component editors are free to modify components without restrictions (as long as they are in the “development” state), the *DerivedFrom* relation does neither imply any strict structural or semantic inheritance relation among the components nor is it the inverse relation of succession. Nevertheless, we expect the typical use case for derivation to be the copying of existing components in order to improve them. This is illustrated in Figure **1**. From an existing component C a derived component C’ can

[9] http://www.meta-share.eu/

[10] Altogether 4 resourceInfo profiles were created representing different resource types, reusing most of the components.

[11] http://dasish.eu/

[12] http://ckan.dasish.eu/

[13] http://www.cessda.net/

be forked at any point in time of the original component's lifecycle. After the necessary amendments, C' will eventually enter the "production" state, possibly alongside C for some time. Finally, when C becomes deprecated, it may explicitly instantiate a *Successor* relation with C'.

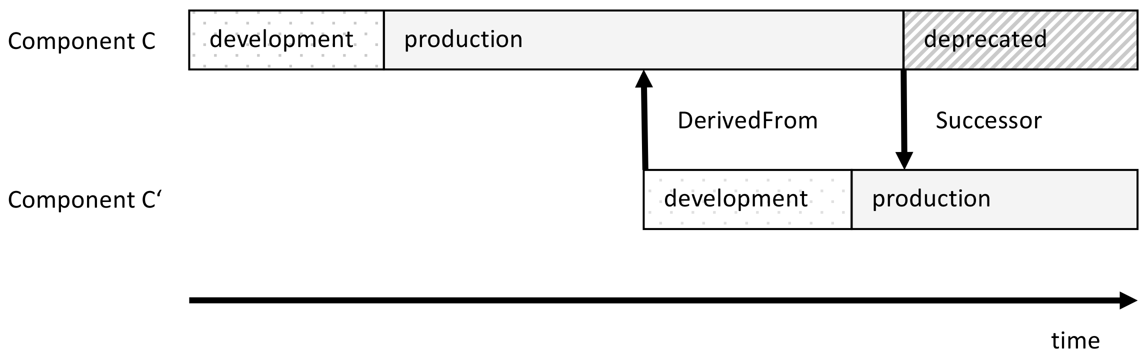

Figure 1: Lifecycle management in CMDI 1.2: lifecycle status and derivation.

It should be pointed out that value assignment to a specific lifecycle status applies only to the component in which it occurs. It is not inherited down through the component structure, nor can any inference on the value of its corresponding element in the child components be drawn safely, at least not in general. For instance, any deprecated component may include child components, which are still in production. Likewise, a deprecated component A containing a deprecated component B may have a successor A' which does not include the successor of B. However, the infrastructure might guide or pose restrictions on transitions. For example, when moving a component into the "production" state, the Component Registry might ask the user to first publish all referenced components that are still in the "development" state and warn the user if any embedded component is marked as deprecated.

The introduction of lifecycle information in components will enable a more sophisticated management of components. For example, as all published components are kept persistently within the Component Registry, the addition of improved versions of components may easily lead to proliferation. Explicit lifecycle management and especially the *Status* field can be used as filtering devices that constrain the users' and modellers' views to a more manageable subset of components. Restricting the selection of available components to those with a "production" status will help users to select the most relevant components. For special tasks such as the development of documentation or component reviews and for ensuring backwards compatibility of the Component Registry with deprecated components, all non-production components will continue to be available within the CLARIN infrastructure.

2.2 Vocabularies

The current version of CMDI requires value domains for elements and attributes to be specified locally in the components. In the cases where value domains are specified as controlled value sets this has several disadvantages. Firstly, updating any value domain is equivalent to updating the containing component. Hence, knowing that many of the value sets are work in progress this may greatly add to the proliferation of components mentioned in section 2.1. Secondly, keeping element value sets as integral parts of components inevitably hampers reuse of components. For example, consider a user looking for a component describing licences, and finding one that is perfectly adequate, except that its element representing the licence name does not list all the licences needed. In such cases, the user has no alternative but creating a new, possibly similar component, making sure that all the needed values are included in the value domain specification. On the other hand, using controlled vocabularies in metadata is in general an effective way to interconnect metadata from various origins, as long as the vocabularies are maintained as shared resources. To this end, CMDI 1.2 will support the use of external vocabularies, thereby increasing the possibility to obtain semantic interoperability across metadata.

Metadata modellers will have the opportunity to associate a vocabulary (identified by its URI) with an element or attribute in their components and profiles. The metadata creator will then be able to pick values from the specified vocabulary or (for open vocabularies) still choose to use a custom value that does not appear in the vocabulary. External vocabularies may be included in component specifications in one of two ways:

1. Vocabularies may be *imported* verbatim into CMDI components, as enumerated value domains for CMDI elements or attributes. In this case the modeller may choose to import all vocabulary items, or only a subset.
2. Vocabularies may be *referenced* by the component and be used for dynamic lookup and retrieval of values when editing metadata records. Here a non-exclusive (open) use of items from the vocabulary must be assumed.

The above will be facilitated by introducing a new element *Vocabulary* in *ValueScheme* elements, with an optional *enumeration* element for imported, closed vocabularies. Examples are given in Code example 1 and Code example 2 below. At the instance level, an attribute *ValueConceptLink* (in the CMDI namespace) will be allowed on fields that have a vocabulary linked to hold the URI of the selected value, see Code example 3.

```
<Element name="Language" CardinalityMax="1" CardinalityMin="1">
      <ValueScheme>
            <Vocabulary URI="http://openskos.meertens.knaw.nl/iso-639-3"
      ValueProperty="skos:prefLabel" ValueLanguage="en">
                  <enumeration>
                        <item ConceptLink="http://cdb.iso.org/lg/CDB-
                  00138580-001">Dutch</item>
                        <item ConceptLink="http://cdb.iso.org/lg/CDB-
                  00138512-001">French</item>
                        ...
                  </enumeration>
            </Vocabulary>
      </ValueScheme>
</Element>
```

Code example 1: An element in a component specification with a closed external vocabulary

```
<Element name="Institution" CardinalityMax="1" CardinalityMin="1">
      <ValueScheme>
            <Vocabulary
      URI="http://openskos.meertens.knaw.nl/Organisations"
      ValueProperty="skos:notation"> </Vocabulary>
      </ValueScheme>
</Element>
```

Code example 2: An element in a component specification with an open external vocabulary

```
<cmdp:Institution
cmd:ValueConceptLink="http://openskos.meertens.knaw.nl/Organisations/dc0
2b3ea-00d9-433f-a540-9baf94a14be0">Sound and Vision</cmdp:Institution>
```

Code example 3: An element in a metadata record (CMDI instance) with a vocabulary item specified

Note that the two modes of using external vocabularies in CMDI 1.2 have quite distinct implications on the component life cycle as well as metadata management. Importing the vocabulary as enumeration into the component allows for strict schema validation of the values in the instance data, but does not automatically reflect changes in the vocabulary. Updating the local copy will typically be done by deriving a new component from the old one and importing the current version of the external vocabulary into the new component.

On the other hand, referencing a vocabulary allows keeping the list of possible values dynamically up to date, but standard XML validation tools will not be able to handle element values obtained this way. The modeller has to decide based on the expected completeness and change rate of the vocabulary which mode to apply. It is assumed that such decisions will be informed by future usage and experience with the vocabulary service, through which guidance and best practice will emerge. As a general rule, large and dynamic vocabularies (e.g. an institution or person registry) are typical candidates for referencing, whereas small and stable vocabularies (e.g. small lists of formats or units) might be imported.

The usage of external vocabularies has some impact on the infrastructure. At the model level, the vocabulary facilities are specified to be generic, in the sense that no assumption about specific services is made. On the operational level – as initially supported by the core CMDI infrastructure – it will be designed to support specifically the OpenSKOS-based CLAVAS vocabulary service (Brugman, 2012), through which vocabularies of languages, organisations and value sets extracted from ISOcat are already available. To make the new functionality available for metadata modellers and creators, both Component Registry and existing metadata editors must be updated accordingly. Dedicated validation tools for handling references to external vocabularies would be useful and feasible, but seeing such tools more as part of the vocabulary service than of CMDI as such, there is at this point no plan for providing such tools as part of the upgrade mechanism supplied by the CMDI task force.

2.3 Cues for Tools

Some of the applications in the context of CMDI, especially those directly used by human users, require information that goes beyond formal specification and validation aspects. This includes documentation of meaning and purpose of all content-related elements and hints for improved visualisation of metadata content. Furthermore CMDI 1.2 will provide the basis for a powerful feature that allows automatic derivation of element content.

2.3.1 Improved documentation of CMDI elements

Documentation especially of content-related elements is essential for both metadata creators and human interpreters. CMDI 1.1 already provides an option to document the usage of CMDI elements but lacks this functionality for attributes or components. Therefore CMDI 1.2 expands the existing approach to all kinds of metadata entities. This allows schema creators to document their profiles in all necessary detail. Furthermore, CMDI 1.2 will permit multiple documentation values for different languages, which can be the basis for localised user interfaces. Code example 4 shows the specification of a component that contains an element, which in turn contains an attribute. It has documentation in both English and Dutch for the first two levels.

```
<CMD_Component name="Actor" CardinalityMin="0"
CardinalityMax="unbounded" ComponentId="ex_compid_actor">
  <Documentation xml:lang="en">
   This is a person or entity that plays a role in the resource
  </Documentation>
  <Documentation xml:lang="nl">
   Dit is een persoon of entiteit die een rol speelt in de bron
  </Documentation>
  <CMD_Element name="firstName" ValueScheme="string"
  DisplayPriority="0" CardinalityMax="1">
    <Documentation xml:lang="en">
     This is the given name of a person
    </Documentation>
    <Documentation xml:lang="nl">
     Dit is de voornaam van een persoon
    </Documentation>
    <AttributeList>
      <Attribute name="nickname" Type="string">
        <Documentation xml:lang="nl">
         Bijnaam van een persoon
        </Documentation>
      </Attribute>
    </AttributeList>
  </CMD_Element>
</CMD_Component>
```

Code example 4: New means for documentation of CMD entities

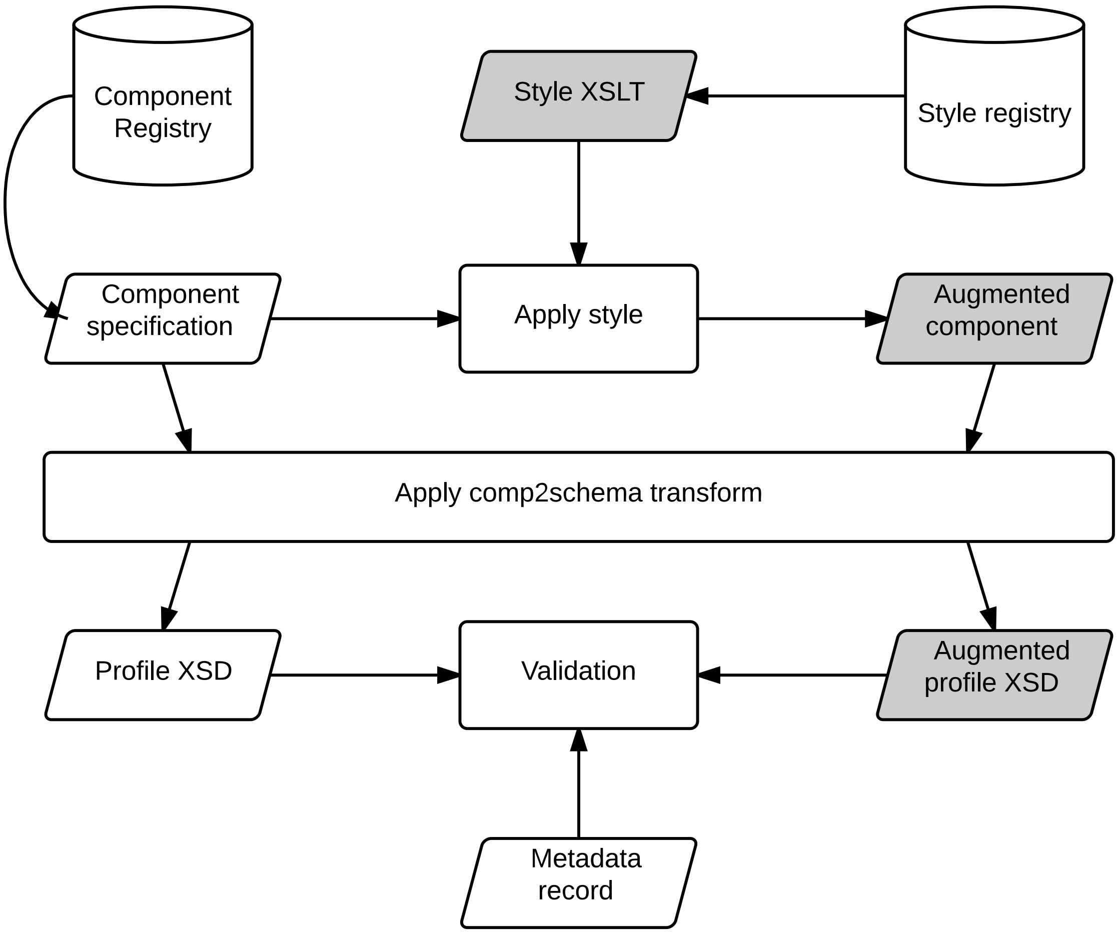

Figure 2: Workflow for visual hints. Shaded components relate to the augmentation of profile schema documents with styling information.

2.3.2 Support for visual hints

Also in the context of user-friendly interfaces extensive changes are introduced to augment metadata profiles with information about how the metadata content should be presented to the user. CMDI 1.1 only provides a very simple approach to specify display priorities for elements. Experiences of recent years showed that this functionality is hardly used and in most cases not even understood by many metadata schema creators. Therefore, this approach is superseded by a new namespace *http://www.clarin.eu/cmdi/cues/display/1.0* for all kinds of display cues. By using an open namespace CMDI 1.2 does not prescribe a closed set of functionality but is completely open for any future extensions that are deemed necessary.

Visual hints that may be useful include:

- Grouping information to allow visual merging of components. This would be especially useful in cases where the content of two components that both contain information about the same issue can be merged and the underlying CMDI structure is not relevant for the end user.
- Selection of elements as representatives of their component. In many cases components contain very extensive information that is relevant in specific contexts but has only minor importance for most users. This could be the case when a component contains very detailed information about a book but only its author and title should be displayed to the user.
- Information about the relevance of a specific element for the whole component that is used as an indication for metadata creators what fields are recommended, optional or even deprecated.

- Explicit visual hints about how an element should be displayed to the end user. This may include suggestions about colour, font size, usage of frames to emphasize specific elements, or the usage of italic or bold letters.

The original component specification is only augmented if necessary. This can be done by means of XSLT transformations according to the workflow as laid out in Figure 2. The current workflow uses CMDI profile specifications that are stored in the Component Registry and converts them via XSLT to XML schemata. These can be used to validate specific metadata record files. This new, supplemental approach extends this workflow by applying XSL transformations provided by a new style registry to enrich the component specification files with additional style attributes. These are also included in the following transformation from component specification to XML schemata. A metadata record that includes style attributes can then be validated against an enriched version of the component specification thus allowing a flexible and expendable workflow without losing the ability to validate a record file against a formal schema. As a consequence of the chosen approach, a tool or a user can decide if display hints are needed at all or may select between different sets of display cues if available.

2.3.3 Value Derivation

A further extension in CMDI 1.2 is the specification of value derivation cues. The experience with CMDI in the last years revealed that a lot of metadata could be automatically derived from other values. The systematic usage of this feature avoids redundancy, helps metadata creators build consistent metadata and allows an explicit definition of relations between elements. Useful applications of this feature may include:

- Definition of duration as the difference of two timestamps.
- Specification of language or country names based on already stated ISO codes.
- Support of keywords like "FileSize" or “CreationDate” that are automatically replaced with their actual value by editor tools.
- Inference of values based on simple regular expressions like the extraction of initials based on already specified first and last name, or the content for a field ‘publication year’ based on a more specific date information.

Similar to the support of visual hints there is no fixed set of allowed rules and keywords. Instead a general framework is specified where most information about relations is defined externally, and the actual derivation is regarded as an optional functionality of applications. Hence it is up to the community what rules, formulas and keywords will establish themselves in the future and what formal structure they will have. Consequently, it is also expected that different tools may support different value calculation methods as there won’t be a central authority that governs a set of allowed values.

In Code example 5, an element holding the age of a file is defined. Its value can be derived from a sibling field *CreationDate*. It assumes a syntax in which a keyword ‘CurrentDate’ exists, as well as a function ‘date’ that in this example takes as its value the path to its sibling element evaluated to its value.

```
<CMD_Element name="AgeOfFile"
        AutoValue="$CurrentDate - date({../CreationDate})"/>
```

Code example 5: Definition of derived values (with hypothetical syntax)

2.4 Attributes in instances

In CMDI 1.1 attributes on instance elements were always optional. The schema for component specifications does not offer a way of expressing the cardinality of an attribute, nor does the Component Registry provide a way of marking an attribute as mandatory. Because of the lack of such an option, it is not possible to closely mimic the constraints of some existing models; the TEI Header (TEI Consortium, 2014), for example, has mandatory attributes. It also poses a needless restriction. In CMDI 1.2, an element 'required' is added to the attribute definition in component specifications in CMDI 1.2 to allow for both optional and mandatory attributes.

For example, a mandatory attribute could be defined inside an element definition as shown in Code example 6. The profile schema generated from this example would render instances of the *firstName* element without a *nickname* attribute invalid.

```
<Element name="firstName" ValueScheme="string"
Documentation="This is the firstname of a person"
DisplayPriority="0" CardinalityMax="1">
    <!-- provide a nickname attribute for this element -->
    <AttributeList>
          <!-- example of an attribute using a simple type -->
          <Attribute name="nickname" Type="string"
                required="true"/>
    </AttributeList>
</Element>
```

Code example 6: Definition of derived values

3 Fixed CMD functionality

3.1 CMD Namespaces

In CMDI 1.1 a CMD namespace, i.e. *http://www.clarin.eu/cmd/*, was introduced. All CMDI records use this namespace, regardless of the profile, and thus XML Schema. This approach, although simple, has led to problems with the basic assumptions about XML, namespaces and schemas made by tools and standards outside of CLARIN. For example, the metadata harvesting OAI-PMH protocol (Lagoze et al, 2002), which is used by CLARIN but specified by the Open Archive Initiative, demands that only one schema is associated with a metadata prefix. But CMDI metadata comes with many schemas, a different one for each profile. Also tools, such as Xerces2-J,[14] that perform XML Schema validation, assume (backed by the XML Schema recommendation (Thompson et al, 2004)) that a namespace is associated with a unique schema and base their caching strategy on this. In CMDI 1.2 therefore, a general namespace for the CMDI Envelope, and profile specific namespaces for the payload are added. (Code example 7 illustrates the use of these two namespaces.) This allows binding of the CMDI Envelope schema to the OAI-PMH CMDI metadata prefix and also supports caching of profiles specific schemas. In principle this change touches every resource and tool in the infrastructure. Fortunately many of these tools can use various approaches, e.g. wildcards, to ignore the profile specific namespaces when they access arbitrary CMDI records.

[14] http://xerces.apache.org/xerces2-j

```
<cmd:CMD
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:cmd="http://www.clarin.eu/cmd/1"
xmlns:cmdp="http://www.clarin.eu/cmd/1/profiles/clarin.eu:cr1:p_13119
27752306"
CMDVersion="1.2"
xsi:schemaLocation=
"http://www.clarin.eu/cmd/1 http://www.clarin.eu/cmd/1.2/envelop.xsd
http://www.clarin.eu/cmd/1/profiles/clarin.eu:cr1:p_1311927752306
http://catalog.clarin.eu/ds/ComponentRegistry/rest/registry/profiles/
clarin.eu:cr1:p_1311927752306/1.2/xsd">
    <cmd:Header>
         …
    </cmd:Header>
    <cmd:Components>
           <cmdp:ToolService>
                 …
           </cmdp:ToolService>
    </cmd:Components>
</cmd:CMD>
```

Code example 7: Fragments of a CMDI record illustrating the use of the namespace

Another namespace related issue is the potential clash between reserved attributes, i.e. *ref* and *componentId*, and user defined attributes. In CMDI 1.2 reserved attributes are moved to the general CMD namespace, so the user has the freedom to define attributes with arbitrary names. These arbitrary names include the names which were reserved for CMDI attributes in 1.1, as shown for *ref* in Code example 8.

```
<cmdp:name cmd:ref="h42"  ref="http://viaf.org/viaf/113230702"
type="person">Douglas Adams</cmdp:name>
```

Code example 8: The separate namespace for envelope and payload allow usage of the *ref* attribute that was a reserved attribute in CMDI 1.1

3.2 Changes in the CMD Envelope

In CMDI 1.1, *IsPartOfList* with its *IsPartOf* elements can be used to link to collections that the described resources and/or metadata are part of. However, the nature of the (implicit) subject of an *IsPartOf* statement has been unclear. While its current position within the Resources element may indicate that any *IsPartOf* relation applies to *all* resources referenced in *ResourceProxyList*, its mere name 'IsPartOf' indicates a single subject.

In CMDI 1.2, this issue will be resolved by moving *IsPartOfList* to the envelope top level alongside Resources, and restricting the semantic of *IsPartOf* to express a partitive relationship between the described resource as a whole and some collection or larger resource. See Code example 9 for an illustration.

```
<CMD xmlns="http://www.clarin.eu/cmd/1">
      <Header>
            <MdProfile>clarin.eu:cr1:p_1345561703673</MdProfile> ...
      </Header>
      <Resources>...</Resources>
      <IsPartOfList>
            <IsPartOf>http://infra.clarin.eu/example/mycollection.cmdi
            </IsPartOf>
      </IsPartOfList> ...
</CMD>
```

Code example 9: Usage of *IsPartOfList* in a metadata record (CMDI instance)

Other relationships between resources than *IsPartOf* can, broadly speaking, be expressed in one of two ways in the CMDI framework; either using components and elements, or as *ResourceRelation* elements within the *Resource* section of the CMDI envelope. *ResourceRelations* in CMDI 1.1 contain simply a *RelationType* element giving a name for the relation, together with elements *Ref1* and *Ref2* pointing to the related resources.

Existing data shows that the latter method has been very little used. There seems to be a general feeling that the current *ResourceRelation* is too simplistic and underspecified to convey the intended information. Although no fundamental change will be performed in CMDI 1.2, the intention is to clarify the semantics of the current specification, all the while keeping the door open for expressivity extension at a later date.

In CMDI 1.2, *ResourceRelation* elements should always contain exactly two Resource elements (replacing *Res1* and *Res2*), explicitly constraining relationships to be binary. In these elements, a mandatory *ref* attribute (indicating a resource listed in the same CMDI record) and an optional *Role* element with an optional *ConceptLink* attribute is added. Moreover, *RelationType* is extended with an optional *ConceptLink*. The new scheme is illustrated in Code example 10, in which the (fictitious) *ConceptLinks* refer to the CLARIN Concept Registry.[15] This way, both relationship direction as well as semantic marking of both relation type and resource roles may be defined by metadata creators.

```
<ResourceRelationList>
      <ResourceRelation>
            <RelationType
      ConceptLink="http://hdl.handle.net/11459/CCR_C-2318_bfda5ab9-
      a429-c2e5-8f08-7c8dfca8245a">annotates</RelationType>
            <Resource ref="rp1">
                  <Role ConceptLink="http://hdl.handle.net/11459/CCR_C-
            346_bfda5ab9-a430-c2e6-8f08-7c8dfca8245a">annotation</Role>
            </Resource>
            <Resource ref="rp2">
                  <Role Conceptlink="http://hdl.handle.net/11459/CCR_C-
            417_bfda5ab9-a429-c2e6-8f08-7c8dfca8245a">annotated</Role>
            </Resource>
      </ResourceRelation>
</ResourceRelationList>
```

Code example 10: Example of *ResourceRelationList* in a metadata record (CMDI instance)

3.3 Component Schema Cleanup

Since the development of CMDI started, multiple developers have worked on the schema that governs how CMDI profiles and components are specified in XML. Different modelling strategies have been applied leading to a mixed bag, e.g. most properties of CMDI elements are specified via XML attributes while similar properties are specified in XML elements for CMDI attributes, as is showcased in Code example 11 (left hand side). In CMDI 1.2 (example on the right hand side) these different approaches are cleaned up by going back to the original approach of using XML attributes whenever applicable.

[15] http://www.clarin.eu/conceptregistry

```
<CMD_Element
Multilingual="true"
CardinalityMax="1"
CardinalityMin="1"
ValueScheme="string"
name="Description">
<AttributeList>
    <Attribute>
      <Name>LanguageID</Name>
      <Type>string</Type>
    </Attribute>
 </AttributeList>
</CMD_Element>
```

```
<CMD_Element
Multilingual="true"
CardinalityMax="1"
CardinalityMin="1"
ValueScheme="string"
name="Description">
<AttributeList>
    <Attribute
    name="LanguageID"
    type="string"/>
</AttributeList>
</CMD_Element>
```

Code example 11: Comparison of the element and attribute definition in CMDI 1.1. and 1.2

4 Migration from CMDI 1.1 to 1.2

Centres should upgrade their data and tools if they wish to benefit from the changes in CMDI 1.2 and good integration with the infrastructure as other centres are upgrading as well. New tools and future versions of existing tools may support CMDI 1.2 only and may not be applicable to unconverted metadata (although conversion can always be performed on the fly, either transparently by the tool or as a pre-processing step by the client).

CMDI 1.1 will be phased out in the future, but initially the core infrastructure components will support both version 1.1 and 1.2, allowing centres to migrate at their own pace. Centres may choose to keep supporting both versions after upgrading, for example by performing on the fly transformations. Migrating to CMDI 1.2 is an active migration process requiring varying degrees of effort from the centres depending on the specifics of the repository and/or tools maintained by the centre involved. The CMDI task force will supply a ready-to-use upgrade mechanism, based on Extensible Stylesheet Language Transformations (XSLT) stylesheets, that will allow centres to convert their metadata records from CMDI 1.1 to 1.2, either one time statically (individually or in batch) or dynamically on the fly. Figure 3 shows a schematic overview of the various aspects of the migration from CMDI 1.1 to CMDI 1.2.

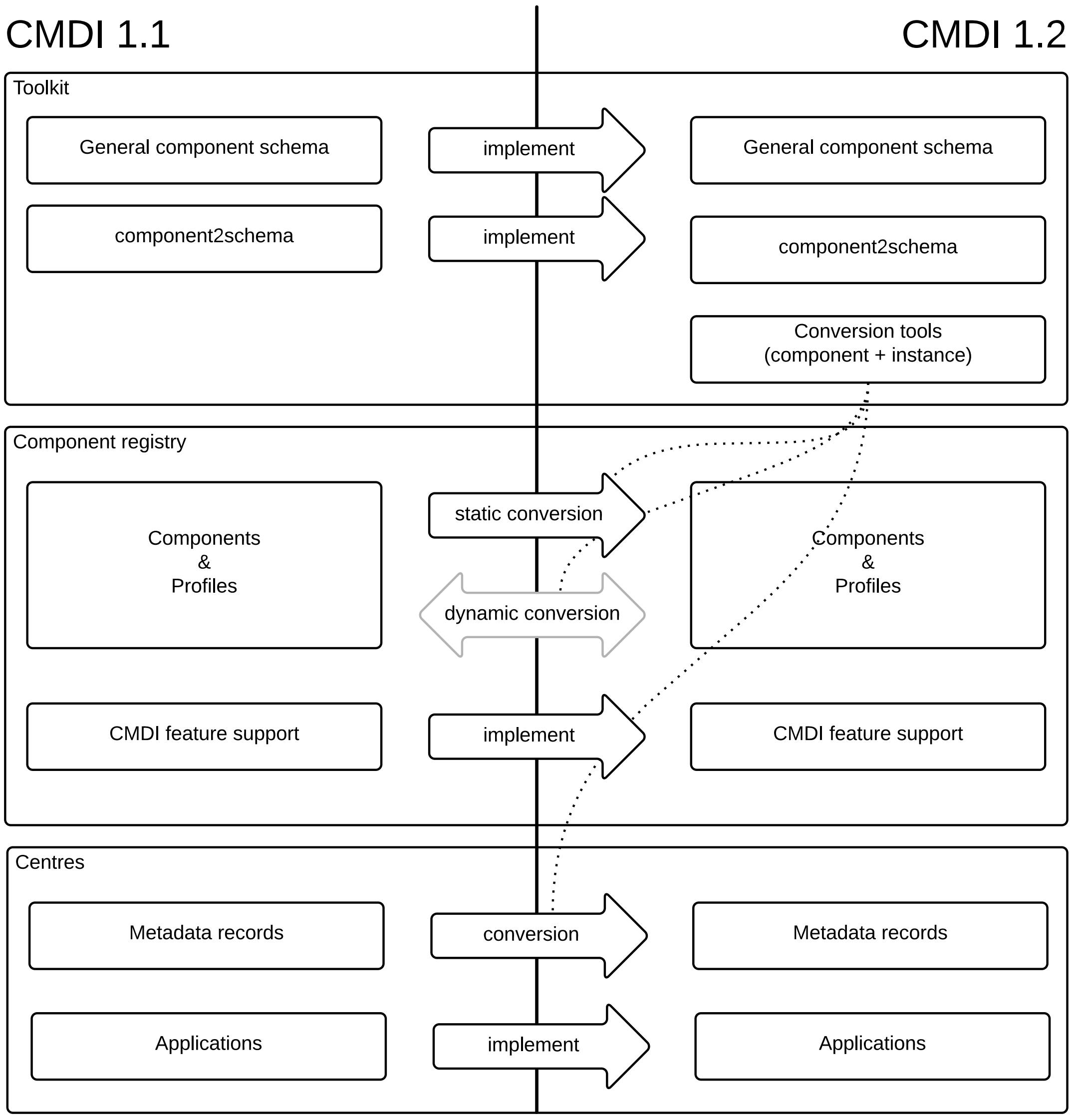

Figure 3: Conversion between CMDI 1.1 and CMDI 1.2: general overview of the toolkit (top) upgrade and the necessary upgrade steps in subsequent parts (below) of the infrastructure.

4.1 CMDI Toolkit and Component Registry

The CMDI toolkit comprises the definitions (in the form of XML Schema Definition (XSD) and XSLT documents) that define the language for the specification of metadata components and profiles as well as the structure of metadata instances in relation to profiles. The task force will produce a new version of this toolkit, which then provides the essential components for creating CMDI 1.2 metadata.

The Component Registry is built on top of this toolkit and will be the first infrastructure component to be adapted to support CMDI 1.2. All existing components and profiles stored in the Component Registry will be statically converted to CMDI 1.2 using an XSLT stylesheet that is part of the toolkit. These components and profiles will become available at a new location in the Component Registry's web service. CMDI 1.1 versions of all components and profiles will be generated on-the-fly by applying a downgrade XSLT and can be requested by tools and users at their current locations. Therefore, the Component Registry will remain compatible with existing infrastructure components. An analysis has shown out that converting existing components and profiles (i.e. those that were present in the registry before the conversion to 1.2) back to CMDI 1.1 after the upgrade can be carried out losslessly, therefore the validity of existing metadata instances is not affected.

Components and profiles that will be created after CMDI 1.2 support has been added to the Compo-

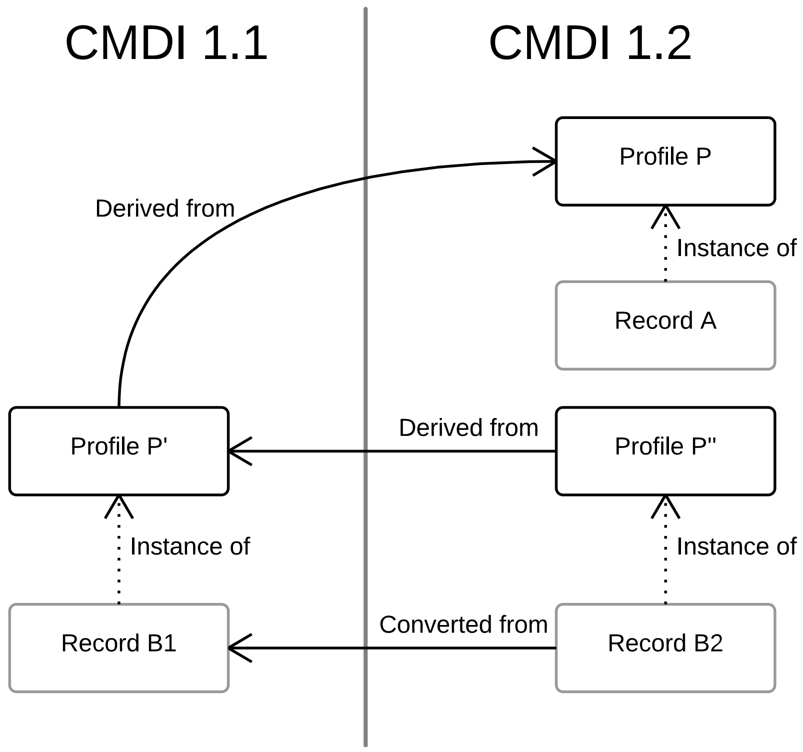

Figure 4: Workflow for a CMDI 1.1 record that is created on basis of a CMDI 1.2 profile and later converted to a CMDI 1.2 instance

nent Registry cannot be lossless converted to 1.1 in all cases, as they may make use of one or more of the newly added features. This poses no problem, as there are no pre-existing instances based on these specifications.

A scenario that needs to be supported by the infrastructure is depicted schematically in Figure **4**. In this scenario, CMDI 1.1 metadata gets created based on profiles that are based on 'native' CMDI 1.2 specifications. If such metadata eventually gets converted to CMDI 1.2 it will not necessarily be valid to the original CMDI 1.2 specification. For example, a CMDI 1.2 profile schema (*P*) might define a mandatory attribute, an option not available in CMDI 1.1. Therefore, the 'dumbed down' profile schema (*P'*) will allow omission of this attribute in instance records (such as *B1* in the diagram). To allow for such a scenario without rendering the metadata invalid when upgrading (yielding *B2*), the Component Registry will also provide a 'dumbed down' CMDI 1.2 version (*P")* of each profile, which in fact will be the result of applying the specification upgrade script to the result of the specification downgrade script applied to the original specification. This version of the profile schema will be available through a separate call, which will perform the chained conversion on the fly. When upgrading a CMDI 1.1 metadata record, its schema location reference should be set to this version of the schema in case the profile is based on a 'native' CMDI 1.2 specification; in other cases, the original CMDI 1.2 version of the schema should be referenced, allowing usage of new CMDI 1.2 features in the instance. This is not an issue if the output of the conversion is either transient or not subject to change.

5 Conversion of CMD Records

The task force will provide an XSLT stylesheet for upgrading metadata records from CMDI 1.1 to CMDI 1.2. Upgrading a record entails transforming the schema reference into a reference to the schema based on the CMDI 1.2 version of its profile (in some cases this should be the 'dumbed down' version, see above) and applying all required changes to make the document compliant with the CMDI 1.2 specification (see sections 2 and 3). No information will get lost in the upgrade process, and the component structure will not change.

In some exceptional cases, an automated transformation cannot be carried out. Specifically, if no profile reference is present in the original record or multiple *ref* attribute values are found on a single element (both of which are schema valid in CMDI 1.1). If such a case is encountered during transformation, the stylesheet will yield an error and the owner of the record will have to adapt the record manually.

A method for converting (downgrading) CMDI 1.2 records to CMDI 1.1 will not be provided by the task force, as there is no generally applicable way of doing so without potentially losing information. In cases where centres or individuals do wish to perform such a conversion, a conversion targeting specific profiles should in generally be quite straightforward. A reason for doing so could be the desire to apply a tool that only supports CMDI 1.1 to a native CMDI 1.2 record.

5.1 Tools, Services and Repositories

Since the Component Registry will keep supporting CMDI 1.1, the need to upgrade other tools, services and repositories hosted and maintained by the centres will not be pressing immediately in most cases. Centres will probably not be inclined to permanently switch to CMDI 1.2 before the majority of relevant tools supports it. On the other hand, the development and adaptation of tools will be driven by the availability of metadata. Adding support for CMDI 1.2 to central tools and services that deal with a broad variety of metadata sources and types, such as the Virtual Language Observatory, will be most urgent. As soon as some support exists in the exploitation stack, it makes sense for repositories to start providing CMDI 1.2 metadata. In some cases this can be achieved by simply applying (additional) transformations. In other cases, however, this will depend on more thorough modifications in the metadata creation pipeline, including editors and content management systems, especially if the new features of CMDI 1.2 are to be harnessed. Centres that generate CMDI on the fly, based on a separate primary data source such as a relational database, have the choice to keep providing both CMDI 1.1 and CMDI 1.2 alongside each other.

Based on the namespace URIs OAI endpoints are able to provide different versions of the CMDI records. The *http://www.clarin.eu/cmd* namespace URI corresponds to CMDI 1.1. While any higher minor version of CMDI 1.x will use the *http://www.clarin.eu/cmd/1* and with the next major version change the *http://www.clarin.eu/cmd/2* URI will be used. This scheme does require future minor versions within a major version to be compatible with each other.

6 Roadmap

Work on the implementation has begun mid 2014, starting with the creation of a new version of the toolkit. Once this has been completed, the Component Registry software stack (REST service and front end web application) will be updated, followed by the migration of all registered components and profiles. After this, the remainder of the infrastructure can be migrated in a distributed fashion. CMDI 1.1 can be formally deprecated once a significant share of the existing records has been migrated and all relevant tools have been adapted. CMDI 1.1 will keep being supported at the core infrastructure level even after deprecation, as will CMDI 1.2 after its eventual succession.

There are a number of tasks related to CMDI 1.2, some of which are currently being worked on, and some of which are planned for after or in parallel to the implementation of CMDI 1.2. First of all, the CMDI task force has initiated the process of writing an extensive and formal specification of CMDI. Such a specification does not exist for CMDI 1.1. Members of the task force have started working on this specification and expect to finish the document in the second half of 2015. In addition to this formal description of the technical scope of CMDI, a document describing *best practices,* targeted primarily at the metadata modeller, is under development.[16]

There is on-going work - coordinated by the CLARIN Metadata Curation task force - on evaluating the quality of the metadata records in the joint metadata domain (cf. Trippel et al., 2014). The main goal is to provide a service that examines individual records or whole collections, performing a number of basic checks (schema validation, "dead links", etc.), and optionally normalisation of values based on controlled vocabularies, producing a curation report that lists encountered issues. The checks will especially also cover the specifics of the CMD versions, to support the data provider in the transition period. Once completed, this service will be integrated into the basic workflow for harvesting the

[16] At time of writing, a draft version of the CMDI best practice guide is available at http://www.clarin.eu/content/cmdi-best-practice-guide

metadata and filling the VLO.

6.1 Open issues

The CMDI task force has decided to leave a number of known shortcomings and potential improvements unaddressed in CMDI 1.2. Rather, these specific issues should be investigated further so that, if feasible, a reliable and non-controversial solution can be incorporated in a future version of CMDI. This section briefly describes four salient ones.

Metadata record versioning information

Most metadata records are subject to change over the course of their lifespan due to for example content fixes, extension, or adaptation to external circumstances. Sometimes a change is applied impromptu, so that a newer version overwrites an existing one, while in other cases a versioning policy is in place that ensures that older versions remain available and each new version gets a distinct identifier. The same applies to resources. In either case, it's often desirable to encode versioning information close to the versioned item. Ways of doing this within CMDI records can be thought of, but an investigation of use cases and ways of representing such information, ideally based on existing common practices, has to be carried out in order to derive one or more appropriate candidate solutions.

Recursive component definitions

Any component or profile specification in any existing version of CMDI, including version 1.2, can be modelled as a tree. The Component Registry does not accept component specifications that hold a reference resulting in a cycle. Therefore no CMDI schema can be derived that allows for arbitrary depth of nesting. To illustrate, one cannot model a component *A* such that it contains a component *B* which in turn specifies component *A* as a descendant. XSD does allow for such circular references, and in fact some existing metadata schemata contain them. For example, the schema for MODS defines an element 'relatedItem' within 'mods', which can hold the same child elements as a 'mods' elements, including 'relatedItem' (Gartner, 2003). However, in CLARIN it is strongly suggested best practice to use semantically explicitly specified concepts for metadata elements. This is in strong contrast to very general concepts such as 'relatedItem' where the position within the metadata tree crucially contributes to the semantics of a given metadata element. The best practice approach strongly reduces the need to exploit the structure of the metadata and therefore reduces the need for recursive use of components. Moreover, introducing the possibility of circular references in component specifications would require a number of fundamental changes in tools that process CMDI records on basis of component specifications or profile schema files.

Nillable fields

Element types in CMDI are derived from XSD types. An option on types that is available in XSD, but not adopted in CMDI, is nillability. While many of the potential use cases for *nil* values can be covered by omitting optional fields, or leaving a string element blank, there are also cases where there is no proper alternative. For example, a modeller might decide that date information should be mandatory, but also want to support cases where date is undefined. Leaving a mandatory date element empty renders an instance document invalid with respect to the schema, so that would not be a proper solution.

The need for workarounds, such as representing dates and booleans as strings or making fields optional where they should not, can be removed by allowing, on selected elements, for the XML-standard attribute *xsi:nil*[17] (which takes the value true to indicate a non-value in the record). It can be combined with a specification of the semantics of *nil* in its particular context (e.g. whether it represents 'unknown' or 'unspecified'), either in the record or in the component specification. This would also prevent metadata creators from entering bogus information to force validity. However, before such support can be added, the methods for defining the exact semantics of *nil* values need to be decided on. Furthermore, the effects on profile schema generation from the component specification need to be investigated.

[17] http://www.w3.org/TR/xmlschema-1/#xsi_nil

Resource proxy constraints

Finally, the task force has discussed but not yet designed or implemented, ways of controlling, via the component specification, the range of resource proxy types and allowed reference points to these proxies. By means of such a facility, a metadata modeller could for instance specify that an instantiation of an 'audio recording' profile should only contain resource proxies with 'audio' media types.[18] Similarly, profiles intended to be used for metadata collections could restrict resource proxies to those of the 'metadata' type.

As an example of controlling the resource reference points, a multimedia session profile might require a reference for each audio or video resource proxy from a 'technical details' section and to a text file proxy from a 'transcription' section.

Such a specification mechanism provides the modeller with some control over the resources coupled to metadata documents, which is lacking from current CMDI implementations. A level of validity (in addition to schema validity) could be derived from this aspect of the specification in relation to a metadata instance. For this reason, this proposal needs to be worked out further before it can be integrated centrally into the CMDI framework. The 'cues for tools' extension mechanism described in this paper could be used to add comparable functionality on the level of user guidance, depending on support by editors and other tools.

7 Conclusion

After 5 years of intensive usage by the community the CMDI task force has reflected the gathered experience in a new minor version of CMDI – CMDI 1.2 – with a number of fixes and improvements. The proposal being finalized and approved, the work now concentrates on a smooth transition of the infrastructure and the data. It is hoped that the CMDI community will largely and successfully adopt CMDI 1.2 and provide the support required to implement these and other enhancements in the future.

References

Broeder, D. Kemps-Snijders, M., Van Uytvanck, D., Windhouwer, M., Withers, P., Wittenburg, P., and Zinn, C (2010, May). A Data Category Registry- and Component-based Metadata Framework. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC)*, pages 43–47, Valletta, Malta

Broeder, D., Windhouwer, M., van Uytvanck, D., Goosen, T., and Trippel, T. (2012). CMDI: a Component Metadata Infrastructure. In *Describing LRs with Metadata: Towards Flexibility and Interoperability in the Documentation of LR Workshop Programme*.

Brugman, H., and Lindeman, M. (2012). Publishing and Exploiting Vocabularies using the OpenSKOS Repository Service. In *Describing LRs with Metadata: Towards Flexibility and Interoperability in the Documentation of LR Workshop Programme*.

Ďurčo, M. & Windhouwer, M. (2014). From CLARIN Component Metadata to Linked Open Data. In *Proceedings of the third Workshop on Linked Data in Linguistics: Multilingual Knowledge Resources and Natural Language Processing,* LREC 2014 Workshop.

Gartner, R. (2003). MODS: Metadata Object Description Schema. *JISC Techwatch report TSW*, 03-06.

Gavrilidou, M.; Labropoulou, P.; Desipri, E.; Giannopoulou, I.; Hamon, O. & Arranz, V. (2012). The META-SHARE Metadata Schema: Principles, Features, Implementation and Conversion from other Schemas. In *Describing LRs with Metadata: Towards Flexibility and Interoperability in the Documentation of LR Workshop Programme*. LREC 2012, Istanbul.

Henrich, A., & Gradl, T. (2013). DARIAH (-DE): Digital Research Infrastructure for the Arts and Humanities-Concepts and Perspectives. International Journal of Humanities and Arts Computing, 7(supplement), 47-58.

[18] As defined by the *mimetype* attribute in the Resource Proxy, referring to Media Types, formerly known as MIME types; see http://www.iana.org/assignments/media-types/media-types.xhtml.

Lagoze, C., Van de Sompel, H., Nelson, M., and Warner, S. (2002). *The Open Archives Initiative Protocol for Metadata Harvesting*. http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm. Accessed on 20 June 2014.

TEI Consortium, eds. (2014). *Guidelines for Electronic Text Encoding and Interchange*. 20 January 2014. http://www.tei-c.org/P5/. Accessed on 20 June 2014.

Thomson, H.S., Beech, D., Maloney, M., and Mendelsohn, N. (2004). *XML Schema Part 1: Structures Second Edition*. http://www.w3.org/TR/xmlschema-1/. Accessed on 20 June 2014.

Trippel T., Broeder, D., Durco, M. and Ohren, O. (2014) Towards automatic quality assessment of component metadata. In *Proceedings of the Ninth Conference on International Language Resources and Evaluation (LREC). Reykjavik, Iceland, 26-31 May, 2014.* Pages 3851-3856.

Windhouwer, M., Goosen, T., Schonefeld O, Ohren, O., Eckart, T., Herold, A., Misutka, J., Frankhauser P., Schiel, F., Eckart, K., et al. (2014). *CMDI 1.2 changes - executive summary*. Technical Report CE 2014-0318, CLARIN ERIC, http://www.clarin.eu/content/cmdi-12-changes-executive-summary.