Title	CMDI 1.2 changes - executive summary
Version	2
Author(s)	Menzo Windhouwer *(The Language Archive/DANS)* Twan Goosen *(CLARIN ERIC)* Oliver Schonefeld *(Institut für Deutsche Sprache)* Oddrun Pauline Ohren *(CLARINO)* Thomas Eckart *(ASV Leipzig)* Axel Herold *(Berlin-Brandenburg Academy of Sciences and Humanities)* Jozef Misutka *(LINDAT-Clarin)* Peter Fankhauser *(Institut für Deutsche Sprache)* Florian Schiel *(Bayerisches Archiv für Sprachsignale)* Kerstin Eckart *(IMS, Universität Stuttgart)* Jussi Piitulainen *(FIN-CLARIN)* Emanuel Dima *(Eberhard Karls Universität Tübingen)* Marcin Oleksy *(CLARIN-PL Language Technology Centre)* Matej Ďurčo *(CLARIN Center Vienna)* Sussi Olsen *(CLARIN-DK-UPCH)* Tõnis Nurk *(Center of Estonian Language Resources)* Uwe Reichel *(Bayerisches Archiv für Sprachsignale)* Marta Villegas *(University Pompeu Fabra)*
Date	2016-07-07
Status	Update after release of CMDI 1.2
Distribution	SCCTC
ID	CE-2014-0318

Note: This is an updated version of a document that was originally published 2014-04-07. The purpose of the original document was to present the proposed changes to CMDI 1.1 that would make up CMDI 1.2. However, at the time of publication, the work on the actual specification and implementation was still to begin. In the process of implementing the toolkit for CMDI 1.2 and writing its specification, the CMDI task force has made a number of decisions that causes deviations with respect to the details given in the original version of this document. This document retains the perspective of the original document but **reflects the actual technical state** of CMDI 1.2 at the time of its release.

More information, including a full specification of CMDI 1.2 can be found at http://www.clarin.eu/cmdi12. The toolkit is available via http://hdl.handle.net/11372/CMDI-0001.

Throughout this document, sections that have been altered or extended significantly in this version have been marked with a red line in the margin.

1 Introduction

The current version of the Component Metadata Infrastructure (CMDI) is 1.1 and was established early 2011. Since then the usage of CMDI has increased strongly, and through this usage the community has become aware of a number of potential feature additions as well as improvements with regards to its design and implementation. The CMDI task force (see below) compiled a set of achievable and non-controversial changes that can be applied to the current specification to come to a new version, which will be labelled 'CMDI 1.2'. In addition, it has worked out a migration workflow that allows for a gradual transition to the new version that can be carried out by centres on an individual basis, at any time, and without losing any information, while requiring minimal centralised effort apart from the work required on the central infrastructure components.

The proposed changes that define CMDI 1.2 are grouped into a number of separate topics. For each topics, this document describes a general description of the related issues and/or potential enhancements, followed by the proposed changes and, where applicable, an assessment of the impact on tools and centres. For the more technically inclined reader there is an Appendix (page 16 and further) containing XML representations of CMDI 1.2 component specifications, instances and parts of the general component schema.

All information in this document is based on the more elaborate descriptions that can be found on the CMDI 1.2 wiki pages[1], which in turn are the outcome of discussions held partially on the same platform and partially in various (virtual) meetings by the CMDI taskforce.

The wiki pages also contain the discussion on two topics that did not result in any changes for CMDI 1.2, i.e., adding an MdType element[2] to the header to indicating the collection status, and allowing resource proxy references on elements[3]. For the first topic the CMDI taskforce does have a *best practice suggestion*[4].

1.1 CMDI task force

The CMDI task force has the task of maintaining the CMDI specification, documentation and implementing the core infrastructure components. The work towards a CMDI 1.2 specification proposal (of which this document is part) was distributed over a number of working groups (WG's) corresponding with the topics described in the various sections of this document.

---

[1] https://trac.clarin.eu/wiki/CMDI%201.2

[2] https://trac.clarin.eu/wiki/CMDI%201.2/Header/MdType

[3] https://trac.clarin.eu/wiki/CMDI 1.2/Resource proxies/Element

[4] https://trac.clarin.eu/wiki/CMDI%201.2/Header/Summary#bps

2 Migration

Centres should migrate their metadata, tools and infrastructure to CMDI 1.2 if they want to keep up with current developments. Aspects of migration can be categorised as component, instance or tool related. Existing components and profiles should stay available as they are, while newly created components and profiles should stay accessible to the existing infrastructure components. Instances need only be upgraded from 1.1 to 1.2 on a per-repository basis. Migration has varying impact on infrastructure components and tools, which needs to be assessed.

2.1 Proposed solutions

2.1.1 Component migration

XSLT style sheets will be created that operate on the component specifications for conversion between 1.1 and 1.2 and vice versa[5]. An upgrade of all existing components will need to be carried out once, the result of which will be consolidated as the new state of the Component Registry. After this, the Component Registry can perform reverse conversion on request. These should result in functionally equivalents to the original specifications. An assessment has shown that cycles starting with original 1.1 components can be carried out without losing information. Components created newly in 1.2 cannot always be converted to 1.1 losslessly but poses no problem as there are no pre-existing instances.

A round-trip starting with 1.2 is also possible and requires some attention as this is not guaranteed to be lossless and therefore instances conforming to an original 1.2 profile schema may not be valid with respect to the schema based on the profile converted back and forth. This will have to be taken into account when extending the Component Registry with conversion functionality.

Component migration will be carried out centrally and has no direct impact on centres apart from 1.2 versions of existing profiles becoming available, allowing for instance migration.

2.1.2 Instance migration

The CMDI taskforce will facilitate instance *upgrading* only, i.e., CMDI 1.1 compatible instances will need to be upgradeable to 1.2 but no valid use case for a general conversion in the opposite direction has been raised.

An XSLT style sheet for general usage will be created that carries out the following changes to an existing 1.1 record:

- changes the CMDI version number to 1.2
- changes the schema location of the instance to the 1.2 counterpart of the original profile
- updates the namespaces (see Section 8.2)
- moves reserved attributes into the CMDI namespace (see Section 4.2)

[5] Available as part of the CMDI toolkit at http://hdl.handle.net/11372/CMDI-0001

- adds an MdProfile header if it was not present before (see Section 6.1)
- applies changes in the structure of the resource proxies section (see Section 9.1)
- performs various clean-up tasks

It should throw an error if:

- no MdProfile or schema location was provided

It should issue a warning if:

- there are multiple values in a *ref* attribute (see Section 9.1)
- multiple (different) profile IDs are present
- the xsd schema location features are used in a non-compliant way
- if a schema location was used other than one from the component registry
- there are incomplete resource relations

2.1.3 Tool migration

The upgrade to 1.2 has the biggest impact on the Component Registry. Next in line are CMDI editors, e.g., Arbil and Proforma, which need to become version aware and might need to support both 1.1 and 1.2 for a while. Other exploitation tools can cope, in general, by getting the code base to handle 1.2, and upgrade any 1.1 instances it needs to process using the migration XSLT.

3 Component lifecycle management

There is a need for options for versioning and deprecation of CMDI component and profiles. Such features are believed to be helpful in reducing the perceived 'proliferation' in the Component Registry. Currently, published components cannot be deleted without potentially affecting instances and no formal relations between successors and predecessors can be defined.

3.1 Selected solution

The component specification needs to be extended with a number of header elements that contain the versioning information, as described in the following subsections.

The CMDI Component Registry will have to manage the status of each component and discourage users (both metadata modellers and creators) from using components or profiles that carry a non-production (i.e. *development* or *deprecated*) state. Details of the required changes are given below.

3.1.1 CMDI Component Specification

A number of status headers will be added:

/ComponentSpec/Header/Status

This mandatory element indicates the status of the component at the time of retrieval. It has exactly *one* value out of the following:

- *development* (may still be edited by owner(s); only listed if marked for review)
- *production* (should be listed publicly, read-only to everyone)
- *deprecated* (no longer in public listing, queryable and read-only to everyone)

Only the following status changes will be allowed by the infrastructure: *development* → *production* and *production* → *deprecated*.

/ComponentSpec/Header/StatusComment

This optional element allows for information regarding the reason for deprecation or succession.

/ComponentSpec/Header/Successor

This optional element points to the direct successor of the component, indicating a succession relation. It is only meaningful if the status is *deprecated*. The value should be the full URI of the successor component. Only a single successor can be specified.

Succession is a transitive relation. The final component in a successor chain should initially be a published component. Users should be warned when deprecating a component without specifying a successor component.

/ComponentSpec/Header/DerivedFrom

This optional element points to the component from which the specified component was derived. No structural relation or similarity is implied. It is intended to be used for the construction of 'derivation trees' to the benefit of metadata modellers. Only a single component can be specified here.

Note that the derived-from relation is **not** the inverse of the successor relation.

3.1.2 Infrastructure: Component Registry

Inclusion of deprecated components in listings of publicly visible components and profiles should be toggleable (off by default). Warnings can be given, both in component listings (non-obtrusive) and when a user make a life-cycle status change or tries to utilize a deprecated component (dialogue) The DerivedFrom header value should automatically be populated with the ID of the original component if it is 'edited as new'. The user should be able to override this behaviour.

It is desirable to offer a means of active notification (by e-mail or RSS) to the user when one or more of their components are affected by a life-cycle status change in another component.

3.1.3 Impact on tools

Deprecated versions of profiles should not be advertised to the users of **metadata editors** but there should be no restriction in using them. The metadata editor can indicate the deprecated status of used profiles, and issue (non-obtrusive) notifications about the status of used profiles. In case a successor is available, the editor should notify the user. Existing metadata instance should remain untouched.

Search tools that explicitly distinguish between profiles in the user interface could apply clustering with respect to different versions of the same profile (traversing the succession chain). In those cases where profile names are shown, it can be considered to show the status of the profile.

4 Attributes

4.1 Mandatory attributes

In CMDI 1.1 attributes are always optional. This does not allow for closely mimicking the constraints of some existing models (TEI header, for example, has mandatory attributes). Also it poses a needless restriction.

4.1.1 Selected solution

Add an element 'required' to the attribute definition in component specifications. If set to true, this translates to an additional attribute use="required" in the XSD.

4.2 Generic attributes to CMDI namespace

All attributes in CMDI 1.1 instances are in no-namespace, this counts for both attributes defined in the component/profile specification and the CMDI reserved attributes (@ref and @componentId), which have special semantics. Currently, the Component Registry (when editing a component/profile) and Schematron rules (when importing a component/profile) check for the presence of attributes with these names and rejects them. A better solution would be to allow any name for user defined attributes and move the reserved attributes into a dedicated namespace.

4.2.1 Selected solution

Leave user defined attributes without namespace and put the reserved attributes in the same namespace as the elements, namely *http://www.clarin.eu/cmd/1* - an obvious namespace prefix would be 'cmd':

- @cmd:ref
- @cmd:componentId
- @cmd:ValueConceptLink

5 Vocabularies

The objective for this part of CMDI 1.2 is to utilise external vocabularies as value domains for CMDI elements and attributes. While the solution for this should be generic and service-agnostic on the model level, on the operational level - as supported by the core CMDI infrastructure - it will be designed specifically to work with the OpenSKOS-based CLAVAS vocabulary service.

The desired workflow is that metadata modellers can associate a vocabulary (identified by its URI) with an element in their components and profiles. The metadata creator will then be able to pick values from the specified vocabulary or (in the case of open vocabularies) still choose to use a custom value that does not appear in the vocabulary.

Editors like Arbil need to be extended to access the CLAVAS public API for retrieving potential values from vocabularies.

CLAVAS is CLARIN-NL's application of the OpenSKOS vocabulary service. The vocabularies are available through a set of web services hosted at Meertens. Currently, the following vocabularies are available in CLAVAS:

- Languages
- Organisations
- All public metadata ISOcat categories of type *closed* (as concept schemes) or *simple* (as entries in these schemes)

5.1 Selected solution

There are two ways of using the OpenSKOS vocabularies in a **component specification**:

- Importing vocabularies as closed value domains for CMDI elements or attributes. Schema based validation is possible.
- Using one or a combination of vocabularies for dynamic lookup and retrieval of values for a CMDI element or attribute. Here a non-exclusive (open) use of items from the vocabulary must be assumed, as validation against such external vocabularies is not practicable.

The following changes to the General Component Schema accommodates vocabulary use for both Element and Attribute:

- New element Vocabulary in ValueScheme (replaces enumeration)
- Vocabulary optionally has an enumeration element. If so, this defines an internal, closed vocabulary (imported or locally specified). If enumeration is not present, then the vocabulary will be considered to be external and open, and should be accessed by means of the API by tools supporting this.

Vocabulary has attributes for specifying the vocabulary's URI and which property and language to use as the stored value.

At the **instance level**, an attribute *ValueConceptLink* (in the "cmd" namespace) will be allowed on fields that have a vocabulary linked to hold the URI of the selected value, semantically marking the chosen option.

5.1.1 Impact on tools

- Metadata editors must facilitate vocabulary lookup. Arbil, as the most generic editor - should be prioritized.
- Component Registry must facilitate import of vocabularies. Interface for specifying value domains for elements and attributes must be updated.
- Discovery services (VLO a.o.) could provide assistance for users through vocabularies. E.g. vocabulary-based browsing.

6 Header

6.1 Mandatory MDProfile header element

In CMDI 1.1, all header elements are optional including <MdProfile>. Because of this, and the fact that the @xsi:schemaLocation attribute cannot be made mandatory either, it is possible to create CMDI instances that do not link back to the profile they are based on.

6.1.1 Selected solution

The solution is to make the <MdProfile> element a mandatory element. All CMDI instances must therefore contain this element and thus, it is possible to always definitely determine their profile. It was decided against allowing a unknown value for the <MdProfile> element, because it would essentially make this element optional again. The automatic up-conversion to CMDI 1.2 is therefore not possible for CMDI 1.1 instances, that lack a proper <MdProfile> element and @xsi:schemaLocation attribute and users need to intervene manually.

7 Cues for tools

7.1 Localised Documentation and Documentation for Components

In CMDI 1.1 documentation is only allowed on elements. Furthermore the current solution only supports one instance per element and has no language property; hence, there is no way to provide documentation in multiple languages.

7.1.1 Selected solution

The current *Documentation* attribute on elements in the component specification will be refactored into a new repeatable child element *Documentation* available on components, elements and attributes. It will have an optional *xml:lang* attribute for specifying the language of the provided documentation. In the profile schema this will wrapped in an xs:annotation element.

7.1.1.1 Impact on tools

- Component Registry must support (multiple and multilingual) specification of documentation for components, elements and attributes (for new **and** existing components and profiles).
- Metadata editors may provide (localised) documentation if available. Arbil, as the most generic editor, should be prioritized.

7.2 Extended display information

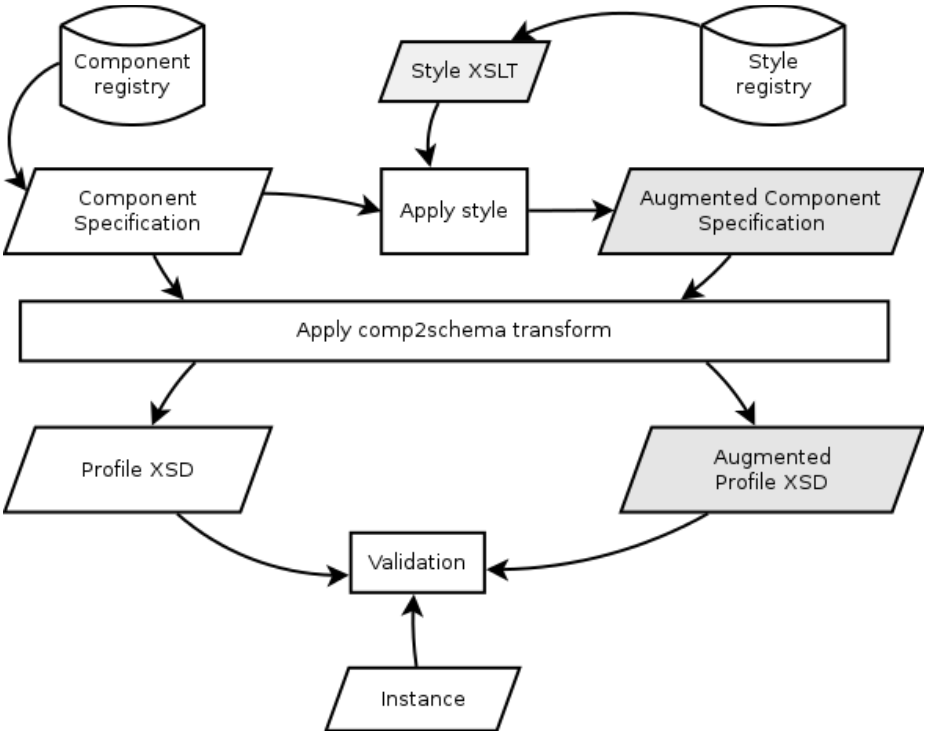
CMDI profiles contain very little dedicated information about *how* information contained in instances should be presented. It would be desirable to have some unified and extensible way of providing (sets of) display cues related to, but physically separate, from the description of the logical structure as with (conceptually) CSS and XML.

Display aspects might include grouping and merging information, salient element selection (as an alternative to 'displayPriority') and an indication of importance beyond cardinality.

7.2.1 Selected solution

A **separate namespace**, i.e. *http://www.clarin.eu/cmdi/cues/1*, will be used for display cues. In this namespace attributes are defined (in a separate schema) for all kinds of display and interaction cues. The general component specification allows any attributes from this namespace on all components, elements and attributes. To define a 'style' for a given component or profile, a style sheet has to be created that augments the component specification with these cue attributes with the desired values.

The component-to-schema XSLT picks up all cues and wraps them in 'app info' blocks in the profile XSD. When requesting a profile from the Component Registry, it can be instructed to apply a style. This workflow is sketched in the diagram below.



The required **infrastructure** extensions will be developed at a later stage and are not part of the CMDI 1.2 specification.

Note that different versions of the style 'language' can potentially exist and tools can support one or more (subsets) of these variations.

7.2.2 Impact on tools

- Extension of the Component Registry with some kind of 'style registry' and 'style editor'
- Extension of the Component Registry API with additional parameter ('style') to activate the augmentation of the component specification with style information

- Metadata editors and viewers (like the VLO) may use this information for enhanced display (may in some cases support only a subset)

7.3 Derived values

Most parts of a metadata record need to be created manually but in some cases parts of it can be derived: either from the described resource (mostly technical details) or from other metadata values, sometimes in combination with external data. The actual derivation of values would have to be carried out by the tool (i.e. the editor).

Some examples of rules that might be supported are *current date*, *file size* (of the described resource), *language name* (from a filled in language code), *actor age* (based on birth of date and current date).

7.3.1 Selected solution

The General Component Schema is extended with one additional optional attribute (*AutoValue*) for the element/attribute specification. This attribute contains a keyword that is used by the metadata generator or editor to generate the content of the respective element/attribute. The transformation that creates profile schemata puts the attribute verbatim onto element and attribute definitions so that it is readily available to the editor.

The supported attribute only provides a "hook" to extend CMDI components with derivation functionality. There will be no concrete specification of supported functions, syntax or of the mechanism to reference content of other elements in the CMDI 1.2 specification. Rather, the specific implementation and coordination thereof is left to the (development) community.

7.3.1.1 Impact on tools

- Component Registry must support adding derivation keywords (as free text) to elements/attributes.
- Metadata editors should provide support for keywords. Specifically this means that editors should automatically insert values for elements/attributes if applicable.

8 Schema sanity

8.1 General component schema consistency

In CMDI several XML schemas play a role. A general fixed schema, the general component schema, determines how a profile or component is specified. Throughout the years several developers have worked on this schema and made different design decisions, e.g., what should be an element and what should be an attribute, what should be capitalized or not. The upgrade to CMDI 1.2 is a chance to clean this up.

8.1.1 Selected solution

The clean-up of the general component schema is to return to, as far as possible, its original approach where information was expressed as Attributes. Next to that some other small clean-up, e.g., removing unnecessary prefixes, is also planned for CMDI 1.2.

8.2 Namespaces

Based on the profile and component specifications a profile specific XSD is generated to validate instances of this profile. In CMDI 1.1 all profiles use the same namespace. This simple approach leads to problems with the basic assumptions about XML, namespaces and schemas in the world outside of CLARIN. For example, the OAI-PMH protocol, also 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, for each profile a different one. Other tools, i.e., Xerces2-J, assume (supported by the XSD recommendation) that a namespace indicates an unique schema and use the namespace for caching purposes. Xerces2-J is commonly used and this problem thus appears in other tools as well. For example, eXist-db runs into validation problems with CMD records that use different profiles but the same namespace. Another parser, schema-aware Saxon-EE (the commercial version of Saxon-HE), behaves similar to Xerces2-J and caches the schemas based on namespaces.

8.2.1 Selected solution

The single namespace approach of CMDI 1.1 is to be replaced by a general namespace for the CMDI envelope and profile specific namespaces for the payload.

The following namespaces are to be used in CMDI 1.2 instances:

- *http://www.clarin.eu/cmd/1* (for the common CMD envelope, suggested prefix *cmd*)
- *e.g. http://www.clarin.eu/cmd/1/profiles/ clarin.eu:cr1:p_1234* (for the profile specific content of a hypothetical profile with id “clarin.eu:cr1:p_1234”, suggested prefix *cmdp*)
- *http://www.clarin.eu/cmd/cues/1* (for display information and other cues for tools, see Section 7.2)

8.2.2 Impact on tools

In principle this touches any tool and resource in the CMD Infrastructure, as they have to deal with more namespaces. However, the impact can be in most cases limited by skipping the namespaces in XPath expressions, e.g., XPath 2.0 allows wildcards to match any namespace, or even to load the CMDI record in such a way that the namespace is ignored or changed to a common one. See the wiki pages for more details. But tools that create CMD 1.2 records will always have to deal with the profile specific namespace to produce valid CMD records.

9 Resource proxies and references

9.1 Single reference from a component

With OAI-PMH it’s possible to retrieve multiple records with one request (bulk retrieval). When one validates the returned document its possible that IDs clash, as they are unique within the original XML document but there was no requirement that they should have been globally unique.

9.1.1 Selected solution

9.1.1.1 Discussion

It's possible in XSD to limit the scope of ID and IDREF using xs:key and xs:keyref (see the implementation example below). A showstopper for this solution is currently that @ref can refer to multiple ResourceProxies, i.e., the type of @ref is IDREFS, while there is no mechanism to ensure uniqueness of the items in IDREFS across documents (no "xs:keyrefs").

An investigation of the around half million harvested CMD records shows that instances of the collection profile are currently the only users of this feature, e.g., the IPROSLA Abel CMD collection record. Here the root collection component refers to all the ResourceProxies. Its debatable if that is needed, as this should be the default interpretation if there is no @ref.

No other use-cases for keeping the @ref as IDREFS has been put forward.

9.1.1.2 Proposal

In CMDI 1.2 we propose to make the type of @ref IDREF (as it was once), and use xs:key and xs:keyref to constrain the @ref value (see implementation example below). This will allow bulk retrieval through OAI, while not posing a serious reduction of expressivity - considering current practice.

10 Relations

10.1 IsPartOf list

The IsPartOf list was described as follow by Thorsten Trippel in his 2012 document:

Resources that are defined in bundles are listed under ResourceProxy. The individual parts can be seen as independent resources as well, such as a subcorpus that can also be distributed on its own. To point out that a resource is part of a larger unit or created as part of a larger unit, the IsPartOfList is introduced referring to one or more larger units by giving the PID of the larger units with the IsPartOf element.

In other word, the list can be used to provide 'uplinks' to collections that the described resources and/or the metadata are part of. This is generally considered a useful feature, however there clearly is some ambiguity as to what is the *subject* of the is-part-of relation(s). The name implies a single subject, but the element itself is a child of /CMD/Resources which contradictory implies a statement about *all* of the referenced resources. If the is-part-of relations should be considered meta-metadata (i.e. say something about the metadata record rather than the resources), the list should not be in /CMDI/Resources but rather a separate list in /CMD.

10.1.1 Selected solution

10.1.1.1 Discussion

Consider the 2 sub-issues mentioned above:

- the name "**isPartOf**" indicates that the (implicit) subject part of the relationship can be referred to as **one** thing

- its current location within <Resources> may be interpreted to indicate that it pertains to **all** ResourceProxies

In combination these issues make the IsPartOf's semantics unclear.

10.1.1.2 Proposal

In CMDI 1.2 we propose to move the <IsPartOfList> out of - and to the same level as - <Resources>, see example below. Any IsPartOf instance should then express a *partitive relationship between the described resource as a whole and some collection or larger resource*.

Any need for expressing partitive relationships involving individual ResourceProxies must then be handled by ResourceRelations or components, using @ref to indicate the related resources.

10.2 Specification of resource relations

CMDI 1.1 has an optional element /CMD/Resources/ResourceRelationsList that can look something like the following:

```
<!-- cmdi 1.1 -->
<Resources>
    <ResourceRelationList>
        <ResourceRelation>
            <RelationType>describes</RelationType>
            <Res1 ref="a_text"/>
            <Res2 ref="a_photo"/>
        </ResourceRelation>
    </ResourceRelationList>
</Resources>
```

This is a relatively little used feature and it has even been argued that it can be removed. However, it is being used in practice and has sensible use cases (at least theoretically). The problems with this implementation are the lack of clear semantics, a forced but implicit relational direction (e.g. Res1 describes Res2) and inelegant naming (Res1, Res2).

10.2.1 Selected solution

10.2.1.1 Investigations and discussion

A query targeted towards all harvestable metadata early February 2014 resulted in only 32 valid occurrences of ResourceRelation. On the other hand, an investigation into metadata from Clarin-D revealed a plethora of ways to express relations between resources, none of which used the ResourceRelation mechanism. It is obvious that exploiting tools will have a hard time interpreting relationships between resources, irrespective what is done to ResourceRelationList.

During discussion, several solutions have been proposed:

- Declare source and target resources explicitly as well as connect the relation type to a concept registry, and thereby make the semantics clearer for binary relations

- Generalize the above by allowing the modeller to specify the resources' roles in the relationship, instead of just source and target.
- Remove the ResourceRelationList altogether, on account of the almost total lack of usage, and the richness of other ways to express relationships in CMDI

From the exchange on the wiki it is clear that discussions on this topic will have to go on beyond the deadline for CMDI 1.2. Hence, it is felt that no fundamental change should be performed in CMDI 1.2. The proposed solution merely attempts to clarify the semantics of the current specification, all the while keeping the door open for expressivity extension at a later date.

10.2.1.2 Proposal

There should always be two *Resource* elements (replacing Res1 and Res2). In these elements, a mandatory @ref attribute an optional "Role" element with an optional @ConceptLink is added.

10.2.1.3 Consequences of proposal

- Relationships are constrained to binary relations for the time being.
- No forced direction of relationships. Even so, any metadata creator is free to limit the resource roles to source and target, and thereby retaining a strict from/to relationship specification
- Semantic marking of both relation type and resource roles.

10.2.2 Comment

Although outside the scope of the actual CMDI model, it should be pointed out that the metadata investigations performed as part of this work, indicate **urgent need for proper guidance to express relationships in CMDI**. Good examples, training and documentation of best practice are all important elements in such guidance.

This proposal currently only deals with relationships between Resources described in one CMDI record, but does not deal with relationships between multiple CMDI-records.

11 Outlook

Note: at the time of this update, the implementation of CMDI 1.2 has taken place and support has been added to the core infrastructure. A specification has also been drafted with a final version expected to be published autumn 2016. More information is available at https://www.clarin.eu/cmdi12.

This document described the changes for CMDI 1.2. Upon approval this revision has to be implemented. To enable the various centres and tool developers to use CMDI 1.2 first various core components of the infrastructure have to be updated or made available:

- The CMDI toolkit, consisting of the general component XSD and the XSLT style sheet to generate the XSD, will be the foundation and include the changes described in this document;

- Next the 1.1 to 1.2 migration XSLT style sheets for both components/profiles and CMD records can be developed (see Section 2);
- Once migration of components and profiles is possible the Component Registry services can be updated to support 1.2 (next to keep on offering 1.1) (see Section 2.1.1);
- The Component Registry can then also offer functionalities in its web user interface that are specific to 1.2.

Other tools, like editors, web service engines, etc., can gradually add 1.2 support. This is even more true for the repository systems in use by the centres, i.e., as 1.1 keeps on being supported they can move over to 1.2 at their own speed - possibly with help of a bulk conversion (see Section 2.1.3).

As the discussions in this document and the CLARIN wiki show there is a need for more consolidated CMDI documentation. The core parts of that are:

1. a specification document describing CMDI on the schema level;
2. a document describing the best practices as they are to be applied within CLARIN, i.e., some of these are not, and cannot even be, enforced by CMDI (1.2) schemas.

Both implementation of CMDI 1.2 and better documentation of the CMDI specification and CLARIN's best practices should be on the TODO list of this Taskforce.

Appendix XML samples

The original version of this document contained a number of XML examples in this appendix. These have been removed in this update since more extensive, up-to-date XML examples can be found in the CMDI specification, which can be accessed via https://www.clarin.eu/cmdi12.