COMEDI: A component metadata editor

Gunn Inger Lyse*, Paul Meurer†, Koenraad De Smedt‡
University of Bergen*‡, Uni Research Computing†
Bergen, Norway
E-mail: gunn.lyse@uib.no*, paul.meurer@uni.no†, desmedt@uib.no‡

Abstract

The flexibility of component metadata (CMDI) brings about a need for editing tools which are equally flexible. Moreover, such tools should be as user friendly as possible in order to lower the threshold for beginners and to promote efficiency even for advanced users. The current paper presents COMEDI, a new web-based editor which handles any CMDI profile. We evaluate currently existing metadata editors and argue that the COMEDI editor is the first one to combine a good level of user-friendliness with sufficient support for CMDI. COMEDI also offers up to date support for CLARIN features such as current license types.

1 Introduction

Digital language data and tools (hereafter for short called 'resources') benefit from good metadata to promote their visibility, reusability and durability (Trippel et al., 2014; Piperidis et al., 2014; Dima et al., 2012; Lyse et al., 2012; Soria et al., 2012; Wittenburg et al., 2010). CLARIN[1] aims to provide researchers with improved access and added value to existing resources for their reuse. To this end, structured documentation of resources is a key factor, enabling researchers to find existing resources and judge their relevance for the intended purposes.

However, the process of creating and managing metadata is a challenge for the less experienced and time consuming even for the experienced. The quality and completeness of metadata will suffer if the tools for metadata creation and management are cumbersome or have missing functionalities (Withers, 2012). Therefore, it is crucial to offer tools that lower the threshold for filling in and editing metadata in a format that CLARIN requires.

The Component Metadata Initiative (CMDI) (Broeder et al., 2010) has led to a standard with the benefit of modularity through reusable components and standard profiles. The XML-based CMDI format attempts to strike a balance between flexibility and stability. The basic building blocks of CMDI are *components*, which consist of sets of *elements* and other components. CMDI is flexible in that the user can choose any set of components that together constitute a CMDI *profile*. At the same time, the reuse of components offers a certain degree of stability, since equal components may then appear in a number of individual metadata profiles. Existing CMDI profiles and components are stored in the Component Registry of the Component Metadata Infrastructure.[2]

In CLARIN, CMDI is the recommended standard for metadata, and certified CLARIN B-centres are required to offer component based metadata for harvesting via the Open Archives Iniative Metadata Harvesting Protocol (OAI-PMH).[3]

The flexibility and power of component metadata makes it desirable to develop and deploy suitable editing tools which are equally flexible and powerful. Moreover, such tools should be as user friendly as possible in order to lower the treshold for beginners and to promote efficiency even for advanced users.

[1] http://www.clarin.eu. Websites cited in this paper were consulted on June 30, 2015.
[2] http://catalog.clarin.eu/ds/ComponentRegistry/
[3] http://www.clarin.eu/node/3577

In this paper, we evaluate the currently available metadata editors and argue that these do not sufficiently support the full power of CMDI in combination with an adequate level of user-friendliness. We then present COMEDI (COmponent Metadata EDItor), which was designed and implemented in the CLARINO project, the Norwegian part of CLARIN.

COMEDI is a web service which can start from any metadata schema conforming to CMDI. Among its features are an intuitive web interface, cloning of information from existing metadata files and validation. It supports all features of the Resources section, such as defining resource proxies which can be referred to in components. COMEDI also offers up to date support for CLARIN features such as current license types.

2 Existing metadata editors

We propose the following as some desirable features for the design and evaluation of component metadata editors:

- handling of any registered CMDI profile;
- import and export of CMDI files;
- fully online web interface to remote processing, saving and storage;
- navigation and editing with menus as well as keyboard shortcuts;
- reduction of repetitive typing tasks, e.g. by cloning of information pieces;
- controlled vocabulary where needed;
- validation of input;
- authentication and authorization of users, with management of shared access;
- support for up to date CLARIN features such as license types.

2.1 Arbil

Arbil[4] is a metadata editor, browser and organizer tool for CMDI, IMDI and similar metadata formats (Withers, 2012). Arbil has been characterized as 'the reference implementation for a CMDI editor' (Dima et al., 2012), but it is also acknowledged that this tool is primarily directed towards archivists or librarians rather than non-expert researchers (ibid.); the latter group may therefore find the learning curve in Arbil rather steep.

Arbil must be installed on the user's system and can be used offline as well as via a webstart. It allows users to create and edit metadata displaying the underlying XML code as plain table structures. Arbil offers several facilities to reduce the burden of repetitive typing tasks, such as bulk editing of metadata via copy and paste into multiple fields of multiple rows. It displays trees of metadata in its user interface. Frequently used sections of metadata can be collected from the *Favourites* directory and be reused for new metadata, thus reducing the amount of repetitive data entries.

Arbil allows the user to type metadata in any order. It warns the user when a metadata field is missing or is not in the required format but allows the user to continue editing and to save locally, even with errors.

When exporting the metadata, all metadata files are checked for inconsistencies and if necessary, warnings are given. Only at the point of pushing the metadata into the remote archive will the user be blocked if they have not correctly completed all the required fields.

In many metadata sets, the number of fields required to describe the data and its context can be extensive; this can make it difficult for a user to see their relevant information at a glance. The table columns in Arbil are therefore customizable, so that only those relevant to a particular user need to be displayed.

Despite its many useful features, Arbil has some drawbacks, among which are its steep learning curve and the local setup and local saving. This makes Arbil less user-friendly for an occasional user, such as the average metadata provider within CLARIN. Moreover, users report that they experience Arbil as responding very slowly.

[4] `http://tla.mpi.nl/tools/tla-tools/arbil/`

2.2 ProFormA

ProFormA[5] is a web-based CMDI editor (Dima et al., 2012). Through a web interface, the user selects a CMDI profile to start from, and the editor displays the profile as a plain online form hiding the XML code. The user may create new files or upload and edit existing files; records can be downloaded as XML files. The ProFormA editor works on local copies of CMDI records, either created in the tool or uploaded by the user. Since ProFormA only allows local saves during a working session, the user must make sure to export an XML copy from ProFormA before closing a session.

ProFormA allows the user to select and upload any CMDI profile in the Component Registry by typing its profile ID into a field in the editor's start page. The CMDI NaLiDa profiles can be selected directly from a drop-down menu in the editor's start page. An existing CMDI file may be edited by pointing to its URL or by uploading the file to ProFormA. In the case of new files, the user selects a CMDI profile and types a filename for the metadata file, which is then opened as a web form. The file is displayed as a simple online form. At the top of the web form, all components in the relevant profile are listed, allowing the user to navigate between components by clicking. Below the component menu, all elements in the selected component are listed consecutively.

Although ProFormA accepts any CMDI profile, some weaknesses became apparent when testing with a profile other than the NaLiDa profiles. As a test, we uploaded the META-SHARE profile *resourceInfo* for lexical resources.[6] Some display errors related to cardinality, i.e. the number of instances of a component or element, were observed. First, ProFormA does not display the required number of occurrences (the cardinality) of an element or a component, even though this information is available in the profile specification in the Component Registry. Second, we found that in the case of elements that may occur arbitrarily many times (cardinality $0 - \infty$), only one instance could be created; conversely, the editor *did* allow arbitrarily many instances of elements where the CMDI profile only allows at most one item. The screenshot from ProFormA in Figure 1 illustrates that it is possible to create more than one instance of the element *metadataLastDateUpdated*, which, according to the META-SHARE profile *resourceInfo* for lexical resources, has cardinality 1.

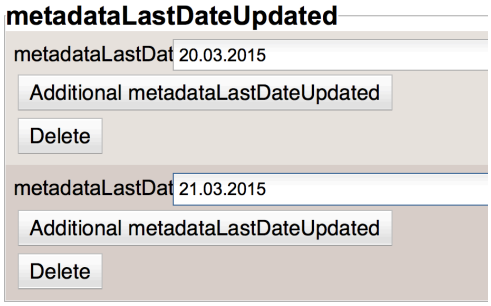

Figure 1: Screenshot from ProFormA illustrating that it is possible to create more than one instance of an element with cardinality [0-1].

Moreover, the hierarchical structure is not always correctly displayed. Sometimes the elements of a subcomponent are displayed without a subcomponent title, which leaves the contents of the subcomponent completely out of context. For example, inside the *distributionInfo* component of the META-SHARE profile *resourceInfo* for lexical resources, a subcomponent *licenceInfo* follows after the element for describing userNature. However, the editor does not display the name *licenceInfo*, so that its contents (person role, surname, given name etc.) do not have an appropriate context (see Figure 2). This is particularly unfortunate since ProFormA also omits the documentation text that ususally accompanies elements and components in the CMDI profile in order to explain the intention of that element.

There are also limitations regarding controlled vocabulary. For some elements (e.g. for *resourceName*) there is an accompanying language element, since a resource may have one English name and another in

[5] `http://www.sfs.uni-tuebingen.de/nalida/proforma/web/`

[6] META-SHARE v3.0 – lexical/conceptual resources; ID: clarin.eu:cr1:p_1355150532312

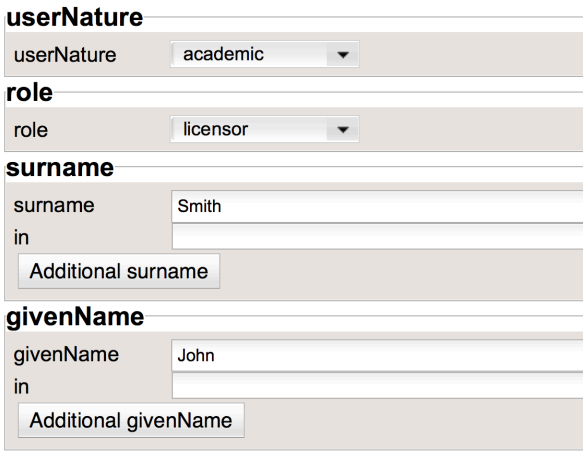

Figure 2: Screenshot from ProFormA illustrating problems with displaying the hierarchical structure of a component correctly.

the native language. ProFormA allows the user to choose language names from a drop-down menu which only lists four languages: Dutch, English, French, and German. It is possible to type other language names manually, but we found that ProFormA does not validate whether the input provided by the user is a correct language name, despite the claim (Dima et al., 2012) that ProFormA validates content provided by the user. Any string, even ill-formed, will pass. Also, ProFormA currently does not support editing persistent identifiers or editing of the header of CMDI files, even though this is an important functionality for archives and repositories.

In conclusion, ProFormA currently appears to be of limited use. It does not provide the full choice of CMDI profiles in the Component Registry and fails to provide some relevant information and functionality to the user.

2.3 META-SHARE

The META-SHARE model is tailored to describe language resources and tools relevant for language technology research and development, and thus has not been designed to support the full CMDI framework. META-SHARE is intended as an open infrastructure for sharing language resources in Europe.[7] It has been created in the META-NET project in cooperation with the META-NORD, META4U and CESAR projects. META-SHARE was deployed Europe-wide in early 2013 and allows anyone to search for language resources online, based on metadata.

META-SHARE offers metadata schemata for four basic resource types: corpus, lexical/conceptual resource, tool/services and language description. Provided that these schemata meet the user's needs, metadata can be created or uploaded, stored, edited, searched and downloaded using an online META-SHARE editor which is implemented as a web-based form. The META-SHARE model has been mapped and incorporated into the Component Registry of CMDI (Piperidis et al., 2014), facilitating a conversion from META-SHARE metadata to CMDI metadata. For this conversion, XSL stylesheets have been written.[8] There is ongoing work to implement and embed an identifier (corresponding to a persistent identifier), using the International Standard Language Resource Number (ISLRN) (Choukri et al., 2012).

The META-SHARE system is constructed as an integrated online solution allowing anyone to search in metadata and allowing authorized users to edit metadata. It is a platform independent, open-source solution, implemented using the Python-based framework Django, which is maintained by active open source communities. META-SHARE is itself open source, released under a BSD licence.[9]

If desired, stored META-SHARE metadata records can be published directly in its online public search interface. Alternatively a metadata record may be tagged as internal (visible and accessible only to au-

[7] http://www.meta-share.eu, http://www.meta-share.org
[8] https://github.com/metashare/META-SHARE/tree/master/misc/tools/CMDIConverters
[9] https://github.com/metashare/META-SHARE

thorized users) or as ingested (visible and accessible in the web editor but not published), or it may be downloaded as an XML file. Resource owners can define editing groups and manage their membership, thereby allowing easy cooperation between several individuals.

The META-SHARE editor supports differing degrees of descriptive detail at two levels: the minimal schema contains obligatory elements (e.g. *Resource name* and a *resourceType* classification), and the maximal schema contains optional information. The minimal schema provides the basic elements for the description of a resource which are therefore obligatory, whereas the maximal schema offers the option to add more detailed information on each resource. Values can be free text or from a limited vocabulary. For example, originally the user could enter license names at will, but this has now become a closed list of license names (plus an option 'other' if the user cannot find the relevant license in the drop-down list).

META-SHARE also has mappings and links to ISO and DC whenever relevant, and also offers certain autofill possibilities. For example, when language is specified, the user can type either the language name (in which case the editor makes autocompletion suggestions) or the ISO code. In either case, the editor fills in the corresponding field for the user.

It is not possible to make an intermediate save without first filling in all obligatory elements. This has the unfortunate side effect that the metadata creator may type 'dummy' information if the relevant information is not available at that time, thus creating a backlog for the metadata creator at a later stage. Even worse, if the user is unable to correct the source of an error message (for the unexperienced user error messages may sometimes be unclear), all data that have been typed so far (even correctly filled in fields) will be lost.

The editor offers autofill (e.g. for today's date) and features controlled vocabularies extensively through drop-down lists (e.g. *Linguality type*) and autocompletion (e.g. language codes), which promotes consistent and correctly typed metadata. The amount of repetitive typing is greatly reduced by storing person info, institutional info and research project info as separate objects in a database which can be pointed at by multiple metadata records. The next time the same object occurs (e.g. the same person), its complete information can simply be selected from this database. If information about an object must be changed later (e.g. changing the e-mail of a person object), the change is automatically updated in all metadata records pointing to this object, provided that the record is stored in the META-SHARE repository. The META-SHARE editor does not support, however, cloning entire records or chunks of information, such as an entire component. Cloning would be desirable in cases where multiple metadata records have a lot of similar information, e.g. if we are creating ten metadata records describing ten parts of a collection, with only minimal differences. In META-SHARE this can only be achieved outside the editor: first, create a record in the editor and making sure it contains all the information you need to duplicate; then download it in XML to your personal computer; create duplicates of your file; then edit each file in XML (and risk syntax errors) or upload each duplicate to META-SHARE and edit it further there. Either way, it is a time-consuming procedure.

A challenge with large metadata schemata is how to portion out elements, components and subcomponents, and META-SHARE appears as very complex and at times confusing in this respect. Subcomponents are usually shrunk initially, but the editor misses a uniform display both for shrunk and expanded components and for how to expand components.[10] For example, main components such as the *Corpus Text* component can be found by clicking via the left-side menu (Figure 3). Whereas the first of the main components, *Administrative Information*, will appear on the same page, the others will appear in a pop-up window.

As illustrated in Figure 4, the component *Distribution* (inside the main component *Administrative Information*) appears in a framed, multiple-line box with documentation text below the component name. The component opens in a pop-up window by clicking a green plus sign. By contrast, the component *Annotations* (within the 'main' component *Corpus text > Recommended*) has a different display: the component name appears in a one-line, grey-coloured box, without documentation text. The component name is a grey font, which makes it even harder to find, especially since it is surrounded on the same page by components that appear as already opened, multiple-line boxes with a blue header line (cf. Figure 5,

[10] See for instance the discussion at: `https://github.com/metashare/META-SHARE/issues/315`.

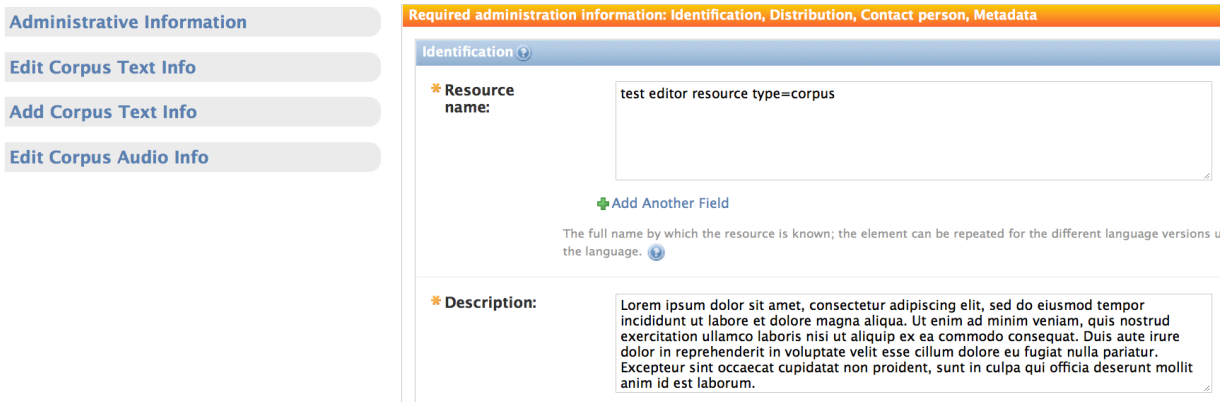

Figure 3: Screenshot from META-SHARE illustrating the heterogenous display for shrunk and expanded components and for how to expand components. Main components are clickable from the left-side menu.

where the component *Annotations* is hidden in the middle of the screenshot). In this case, the component can be opened by clicking on a word *Vis* "Show" after the name,[11] requiring the user to recognize a cue which is quite different from the plus sign used for the same purpose elsewhere. Upon clicking, the component is displayed on the same page instead of as a pop-up window, thus presenting a completely different solution than that in Figures 3 and 4.

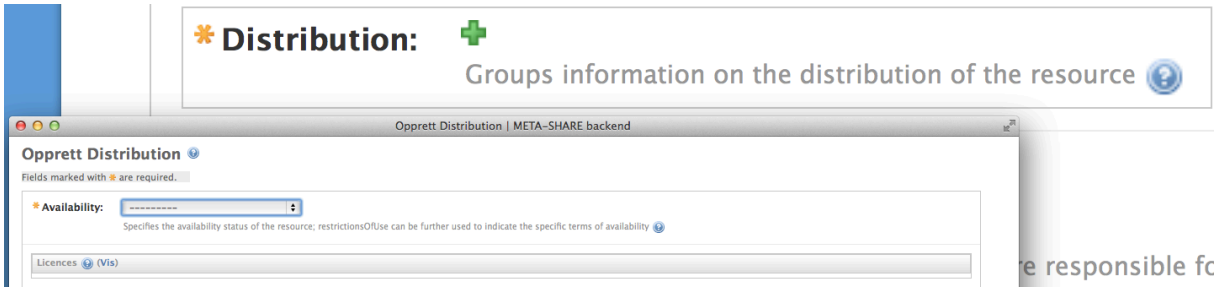

Figure 4: Screenshot from META-SHARE illustrating the heterogenous display for shrunk and expanded components as well as for the expansion of components. In this case, a subcomponent is displayed in a pop-up-window in front of the main page upon clicking the green plus sign.

In conclusion, the META-SHARE editor has some user-friendly features that support efficient and correct editing; however, like ProFormA, it has not been designed to fully support CMDI profiles, which is a non-trivial drawback. The interface shows some lack of consistency and there is also a potential for other improvements.

2.4 Other editors

General purpose XML editors such as Oxygen[12] are challenging to be used by non-experts, since their effective use requires some insight in XML technologies (Dima et al., 2012). The IMDI-editor[13] seems to be more or less replaced by Arbil and will not be further discussed in this paper. CLARIN-D has created the HTML5 web app CMDI Maker,[14] which is now part of the CLARIN infrastructure. It allows users to load files that are part of a resource, to which the researcher can add metadata. CMDI Maker creates IMDI records that may be exported and subsequently uploaded as CMDI with an IMDI profile via Arbil. This editor allows cloning of components for person-related data. It seems limited to a specific set of profiles,

[11] Some elements in the interface have been localized to Norwegian.
[12] http://www.oxygenxml.com/
[13] https://tla.mpi.nl/tools/tla-tools/older-tools/imdi-editor/
[14] http://class.uni-koeln.de/cmdi_maker/

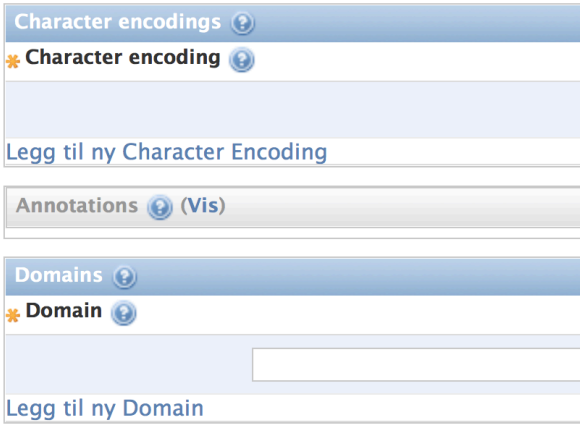

Figure 5: Screenshot from META-SHARE illustrating the heterogenous display for shrunk and expanded components and for how to expand components. In this case, a subcomponent will be opened in the same window after clicking the word *Vis* "Show".

and would therefore not be widely usable.

3 COMEDI

3.1 Overview

The COmponent Metadata EDItor (COMEDI) is a fully web-based editor which handles any CMDI-compatible profile.[15] In COMEDI, one can create a metadata record from scratch, or upload, edit and download any CMDI XML file. It may be used simply as an editor for filling in metadata (and downloading the resulting XML file), but it also functions as a full metadata server for storing, searching, viewing and managing metadata, with user group management for controlling the access rights to individual metadata records. A metadata record in COMEDI can be exported as a CMDI XML file, and is harvestable with OAI-PMH. In accordance with the OAI-PMH standard, all metadata can also be harvested in Dublin Core format.

The developer's instance of COMEDI is currently integrated in a web framework at the emerging national CLARIN type B center at the University of Bergen, Norway. The system can also be deployed as a stand-alone web service which can be installed on individual servers, and the software is available under a BSD license. A stand-alone instance has been installed at the National Library of Norway, with the intention to provide a national metadata registry that will be harvesting from metadata providers in Norway. Another European CLARIN centre has decided to use the local developer's instance, since it offers adequate facilities to define user groups for the relevant centre to have their own 'workspace' within COMEDI. Moreover, they do not intend to store their data on the installation, only to create metadata and download the resulting XML files.

3.2 Authentication and authorization

The contribution of metadata by unidentified users is generally undesirable; moreover, user identification makes it possible to define user groups with shared editing rights to sets of metadata records. Therefore, users of the editor are authenticated via login. The COMEDI editor provides an Authentication and Authorization Infrastructure (AAI) based on the DiscoJuice IdP discovery service. If this AAI is properly set up at the installation, users can login to COMEDI through the CLARIN IdP, the eduGAIN interconnection of IdP federations, or OpenIdP (a self-registration service offered by the Norwegian academic identity provider).

A user management system has been implemented that operates on two levels: the user and the group level. Authorization with differing degrees of rights can be given to authenticated users on an individual

[15] At present, version 1.1 is fully supported, while support for CMDI 1.2 is planned.

basis, such as administration of other users, creating persistent identifiers, etc. When a metadata record is created, it is owned by the creator, who by default has sole write access to the record. Other users can request write access to the record by clicking a button on the metadata page. The owner, who receives an email notification, may then grant or deny access to the record.

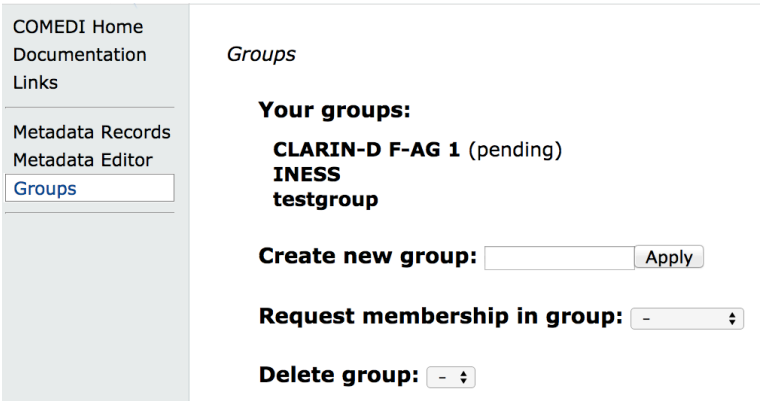

Figure 6: With the user management system, a logged in user may define user groups, apply for membership in existing groups and grant other user access to a group.

In addition, any user can define a user group, initially with that user as its sole member. A group is created by clicking on *Groups* on the left-side main menu of the editor (cf. Figure 6). Again, other users can apply for membership in the group. Subsequently, the group creator, upon receiving an email notification, can grant or deny membership. When a group has been established, a group member can connect an owned record to that group, thus giving all members of the group write access to that record. These operations are reversible. If necessary, a user can be removed from a group, and a group can be deleted.

3.3 Creating, uploading, cloning and searching for metadata records

The left-side menu of the main page has, among other things, a link to a documentation page explaining the functioning of the editor (as can be seen on the left side of the screenshot in Figure 6). By clicking on the *Metadata records* page via the left-side menu, the user finds the main page for starting to work in the editor. At the top of the start page, there are four dialog boxes allowing the user to create a new metadata record, upload a metadata file, clone a record, or search for and select existing metadata in the editor's database for viewing and editing. Each of these options will be described in the following.

Similarly to the solution in ProFormA, the user can create new records by typing the CMDI profile ID (cf. Figure 7), or by choosing among a the set of profiles already registered in the database in a drop-down menu. The user then has to choose a new record identifier which will be used internally for naming the file (e.g. in Figure 7 the name 'demo-corpus' was entered). By clicking the "Go" button, a new, empty record is created based on the selected CMDI profile.

Like in Arbil, COMEDI allows you to use both published and unpublished profiles in the Component Registry, by entering enter the profile ID into the box for choosing a profile. Since profiles published in the Component Registry cannot be modified, it is very convenient to be able to test a metadata profile on real metadata before publishing it in the Component Registry.

Having created a metadata file in COMEDI, the user will find at the top of the page some information about the profile that was used as a blueprint for this metadata file: its profile ID, name, description and the date of fetching it from the Component Registry. There is also a button for refetching the profile, in case the profile is still unpublished and may have been modified in the Component Registry since it was last fetched.

Existing metadata can be uploaded in two ways: via a Web form, or programmatically, using a POST request to a dedicated URL. When the Web form is used (see Figure 8), a CMDI record file is uploaded

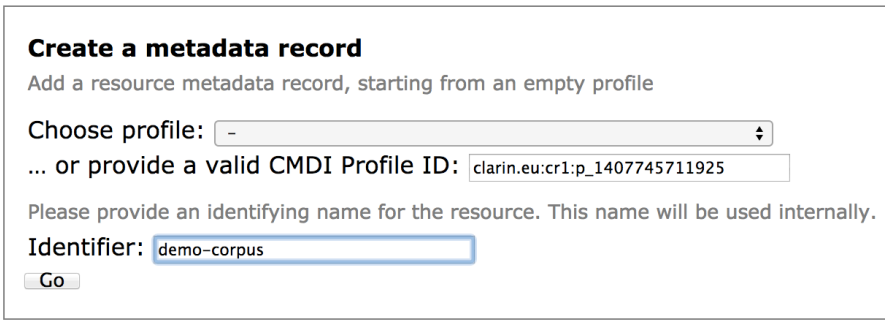

Figure 7: Screenshot illustrating the creation of a new metadata record based on a CMDI profile referred to by its ID.

from a local computer. As COMEDI-internal identifier the name of the file is used; information about the used profile is extracted from the CMDI header.

Figure 8: Screenshot illustrating the dialogue box for uploading CMDI-conformant metadata to COMEDI.

If there are many CMDI files to be uploaded a scriptable mechanism is certainly preferable. To this end, there is a URL that accepts POST requests with a file upload, and an optional parameter that specifies the user group the metadata should be attached to. A slight complication lies in the fact that users who deposit metadata to COMEDI have to be authenticated. Since there is no easy automated way of doing federated authentication, a workaround has been implemented that makes use of the browser-based authentication built into COMEDI: when a user logs into COMEDI a session token is generated that is stored in a cookie. This session token is also exposed on the COMEDI website, and can be used for authentication when programmatically uploading metadata, using the parameter *session-index*. An example shell command for uploading metadata using the Unix *curl* utility could look like the following request, which returns a JSON object signalling either success or failure:

```
curl -F "file=@tiger-treebank.xml"
        "http://clarino.uib.no/comedi/upload?group=CLU&session-index=_a19a23c4"
```

As a last resort, when a service heavily relies on being able to upload metadata, where it is not feasible to manually insert the session index, a permanent access code for uploading can be given upon request.

A metadata record in COMEDI can be accessed at a dedicated URL as a CMDI XML file. Additionally, there is an OAI-PMH endpoint that allows harvesting all published CMDI records. A metadata record has a COMEDI-internal status attached to it which tells whether the record is still unfinished, ready to be published, or finalized. Via the OAI-PMH endpoint, only the finalized records are published and available for harvesting. The OAI-PMH endpoint also makes use of the user groups; they are encoded as OAI-PMH sets. Thus, to harvest all metadata created in the INESS user group one use the following:

```
http://clarino.uib.no/comedi/oai?verb=ListRecords&MetadataPrefix=cmdi&set=INESS
```

Metadata records describing similar resources tend to have much information in common. To ease the creation of very similar metadata records, COMEDI offers the possibility to clone an existing CMDI record.

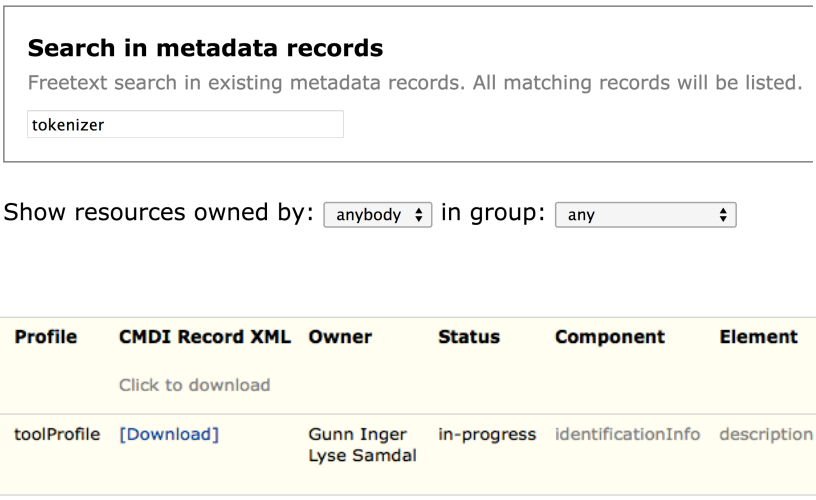

Figure 9: Two screenshots illustrating the dialogue box for searching for content in metadata files in COMEDI (top) and an example of a search result (bottom).

A new CMDI record will be created that is an exact copy of the original record, except that a new identifier has to be supplied, and some fields, like *Self link* and other persistent identfiers relating to the original resource will be cleared, and the header fields *Creator* and *Creation Date* will be adapted.

Below the three dialog boxes for creating, cloning and uploading metadata, there is a box for searching in existing metadata records that are stored in the COMEDI database (cf. Figure 9). Currently this is implemented as a simple string search, and all hits are returned as a list specifying the element and component in which the search string was found. Existing metadata records are listed by resource name and identifier. For instance, Figure 9 shows a resource with name *NLTK tokenize punkt* as specified in the metadata, and below it, the identifier *nltk-tokenize-punkt* is shown. By clicking on the identifier the user may inspect or edit the metadata.

3.4 Component display and navigation

A full metadata schema may appear overwhelming, so the ability to navigate efficiently and selectively show or hide elements to the user is therefore beneficial.

COMEDI displays one top level component at a time, while keeping a menu at the top of the page where the user may switch to another component by clicking on it or by using keyboard shortcuts. The component menu can be seen as the top line menu in Figure 10, where the currently chosen component, *Contact person*, is displayed in boldface[16].

COMEDI offers advanced navigation functionality with consistent shrinking and expansion. All navigation can be done with keyboard shortcuts alone or by mouse-clicks, such as navigating from one component or element to another, switching between edit and view mode, editing content, adding or removing components and elements, showing and hiding subcomponents, or switching between top-level components. The editor also offers basic tab navigation. This range of interaction modes should accommodate both occasional and regular users. An on-line wiki-type documentation is provided.

To accommodate the need for going into detail while typing metadata, and at the same time keeping an overview, COMEDI has two display modes, *view* mode (Figure 10) and *edit* mode (Figure 11). In view mode, the editor displays only the necessary information; specifically, it displays obligatory elements and any other user-provided content. Missing obligatory elements, or user-provided content that does not validate correctly, are marked in red. For instance, Figure 10 shows the *Contact person* component in view mode, where the obligatory e-mail address is missing. Thus, by switching to view mode, the user can easily review the contents filled in thus far and check for missing, obligatory information.

[16]The example uses the META-SHARE corpus profile, ID: clarin.eu:cr1:p_1361876010571

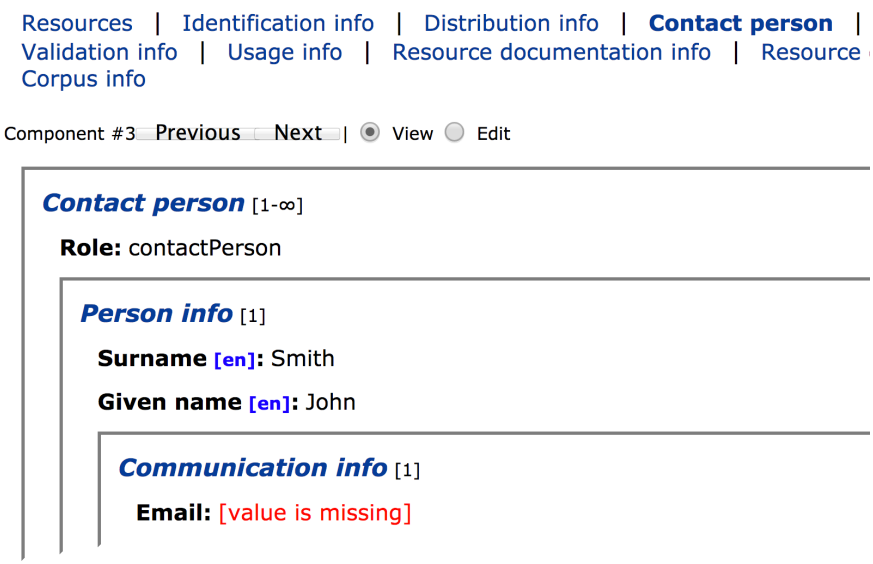

Figure 10: Displaying one top-level component at a time (view mode).

In edit mode, metadata information is displayed in an appropriate level of enhanced detail. In both modes, the components and their elements are shown hierarchically as boxes containing boxes (cf. Figure 10). Akin to the profile display in the Component Registry, components can be expanded and shrunk to adjust the level of detail. Initially, all but the top-level components are shrunk. Optional uninstantiated components are hidden in view mode, but visible in gray in edit mode. Obligatory components and elements are by default open in edit mode whereas optional ones are initially shrunk. Optional components and elements that are not instantiated are also gray.

To illustrate the edit mode, Figure 11 shows the *Communication info* subcomponent inside the *Contact info* component illustrated in Figure 10. In edit mode, each component and each element is displayed along with its documentation from the Component Registry. Adjacent to the component or element name, inside square brackets, the user can see the number of instantiations of a component or element (the left-hand number), along with its allowed minimum and maximum, i.e. its cardinality (the right-hand number). For instance, the element *Given name* in Figure 10 has been instantiated once and may occur zero times or once, hence we see: [1/0-1] in Figure 11. Instances of elements and components can be created or deleted using [+] and [−] buttons, if allowed by the component definition. To prevent accidental deletion of metadata, the user will be asked to confirm before deleting any data.

The user may enter metadata in any order desired. As opposed to META-SHARE, a reliable save functionality is available regardless of whether all user content is valid according to the profile specification. Moreover, after every edit operation, the current metadata is stored in the server database; no explicit save command is necessary. In addition, dated snapshots can be stored at any time, and if necessary, a user can revert to a previous snapshot. Furthermore, through the web interface, some ordinary editing functionality is provided by all modern browsers, such as spell checking and editing functions, including undo, within a form field. COMEDI supports Unicode character input and right-to-left scripts.

Validation, controlled vocabulary and the automatic insertion of metadata are indispensable tools to reduce the amount of inconsistencies and errors in the metadata, while also saving time for the metadata creator. The COMEDI editor validates input according to the *ValueScheme* specification in the element definition and displays an error message on invalid input, for instance for language names, correctly typed `s` and date. Support for vocabulary services like OpenSKOS is planned in the transition to CMDI version 1.2. If a value is invalid, an error message is displayed below the value (Figure 11). Some fields, like metadata creation date and last change date, are filled in automatically.

COMEDI allows easy cloning of components from existing metadata records stored in the repository, thus greatly reducing the work burden of repetitive typing tasks. When clicking on *select component*

Figure 11: Edit mode: validation of language code on the fly.

next to the component name (in edit mode), a list of existing components with the same *ComponentId* appears (Figure 12) and the user can scroll through the list. Since the list of existing components may be extensive, the list may be filtered by entering a search string. In Figure 12, the full list of existing *Funding project*s was narrowed down to 3 existing components containing the string *META-*. The illustrated existing component is the first of three, (denoted at the top of the component as *1/3*), and this first listed component instance appears in 14 metadata records (denoted by *(14) at the top of the component)*. Upon selecting one of the items, its content is copied into the component in focus. To make this feature safe to use, a component can only be filled with new content if it is empty; otherwise, it has to be cleared first. If a component already has content, the user can clear it by clicking *Remove content*. Similarly to Arbil, useful components can be marked as favorites in the component selection box; they will then appear first next time the user evokes the component list for this component type. Cloned contents may be edited as desired. Such editing will not affect the contents of the original, and the edited version will in that case simply appear as a new item in the list of components.

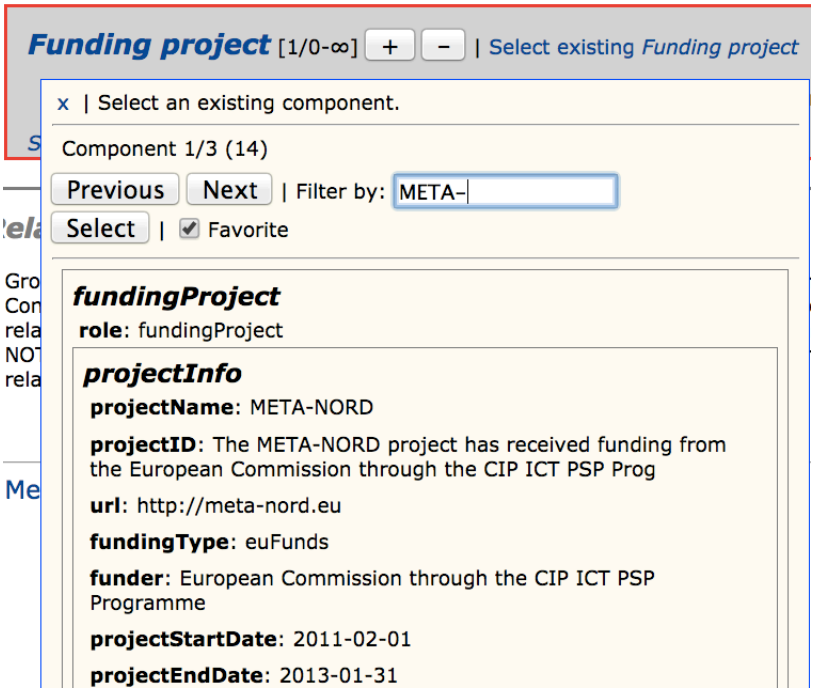

Figure 12: Cloning components: selecting an existing component for cloning.

COMEDI supports all features of the Resources section. Resource proxies can be defined, and they can be referred to in components. This is for instance useful to express that certain parts of the metadata, such as a license or a contact person, only refers to parts of the resource. Users can choose the naming of the *id* values themselves, but uniqueness is checked in COMEDI. The Resources section of the metadata record can be edited in much the same way as ordinary components.

Figure 13 illustrates an example where the user is creating metadata for a treebank and wants to define one contact person in the metadata for questions about the treebank (named in the example with the proxy ID 'iness-nob') and another for questions about a search interface (named with the proxy ID 'corpuscle'). To achieve this, the user must first define the needed resource parts in the Resources section. Next, the user must create two instances of the *Contact* component. As illustrated in Figure 13, every component instance gets a clickable *ref* ('reference') section to the right of its name. When clicking on this reference section (in edit mode), a list for selecting proxy IDs appears. The user may then select, for example, that the first *Contact* instance should refer to 'iness-nob' whereas the second instance should refer to 'corpuscle'.

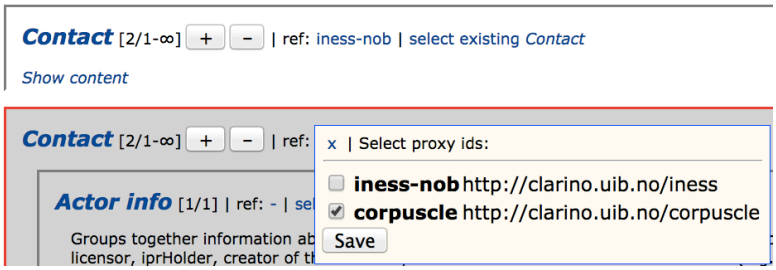

Figure 13: Resource proxies can be defined and subsequently referred to in components.

3.5 Integration of persistent identifiers

In CLARIN, persistent identifiers (PIDs) should be based on the *handle* system offered by the Handle.Net organization. If an installation of COMEDI has access to a local handle server with a dedicated prefix, the handling of handles can be tightly integrated into the system. This is at present done in two places: in the metadata self-link, and in PID elements. A self-link handle can be registered automatically with one click (if appropriate and desired); this handle will point at the URL of the metadata record. For profiles that use the *IdentificationInfo* component, a handle can be registered automatically when the component's PID element is created, and in case there are instantiated URL elements in that component, the newly created handle will point at them. Vice versa, when a handle already exists, newly created URL elements are automatically connected to it. When a URL element is deleted again, it will also be removed from the handle in the handle system, and when a PID element is deleted, the corresponding handle is deleted from the handle server, thus avoiding dangling handles. In addition, the PIDs can also be inserted manually if needed.

The handles that are created in the system are by default EPIC-compatible[17]; they consist of the handle prefix (a number identifying the handle owner) and a suffix of 12 hexadecimal digits plus a checksum digit; they are devoid of semantics (with the exception that the creation time is coded into the suffix). A typical handle thus looks like the following:

```
hdl:11495/D8B8-3AA2-3332-1
```

Handle.Net handles support a flexible handle template mechanism, which allows one to add extensions to the base handle. Only the base handle is registered and resolved, whereas the extension is attached to the resolved URL, possibly in a transformed shape.

In COMEDI, the extension is attached verbatim to the resolved URL and thus retains the full flexibility of URL parameters.

[17]EPIC, the European Persistent Identifier Consortium, is a handle service based on Handle.Net; see `http://www.pidconsortium.eu`

As an example, the handle

```
hdl:11495/D8B8-3AA2-3332-1@version=1.1
```

would resolve to a request for version 1.1 of a resource:

```
http://clarino.uib.no/corpuscle/landing-page
    ?identifier=bul-treebank&version=1.1
```

3.6 Support for CLARIN features

COMEDI also offers support for CLARIN license information.[18] Among other things, CLARIN has introduced a classification system where licenses are placed in one of the three main categories PUB (publicly available), ACA (available for persons with an academic affiliation) and RES (restricted availability on a case-by-case basis). A major challenge for the non-legal expert is to provide consistent and correct license information in metadata. The CLARINO project, which is implementing the Norwegian contribution to the CLARIN infrastructure, has classified existing licences with respect to user Category, license family and conditions of use, and the CLARIN Legal Issues Committee (CLIC) has quality-checked the license table. This license table is a good candidate to be published as external vocabulary in the CLARIN Vocabulary Service in OpenSKOS (CLAVAS) which will be supported in CMDI 1.2. With a CMDI profile that uses CLARINO's license component, the user only needs to fill in the license family and the license name. COMEDI will automatically add information from the license table related to the given license, such as the correct user category, a license URL and conditions of use, as illustrated in Figure 14.

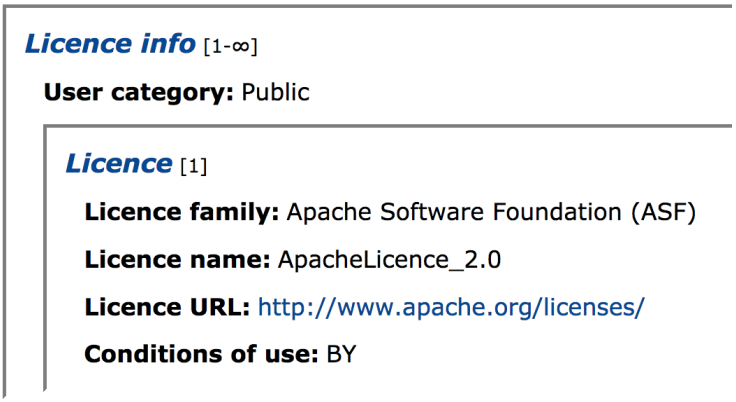

Figure 14: With the CLARINO license component in COMEDI, the user only supplies the license name, and the editor automatically fills in related metadata such as conditions of use.

3.7 User evaluation

Even though a clean web interface and features such as validation and cloning are known to add to user-friendliness, the actual user experience is an empirical issue. As a pilot study, COMEDI was tested on a researcher of linguistics with high technical skills in general, but without previous experience with metadata. The researcher was asked to fill in metadata for a lexical resource that he knows well. Overall, the test person found that the threshold for using COMEDI was low: getting started was easy and it was easy to keep track of the editing process thanks to the top-level component menu at the top of the page and the effortless shift between view and edit mode. The user identified some weaknesses which were easily improved in a subsequent version of COMEDI. For instance, the researcher missed an autosave function when navigating from edit to view mode. This has been implemented. Also, it proved confusing that the action of clicking on a title causes different things to happen depending on whether it is an element title (sending the user to the CLARIN Concept Registry Browser (CCR) page documenting that element) or a component title (shrinking or enlarging the component by clicking on it). To avoid this confusion, the

[18] For information on the CLARIN license classification, see `http://www.clarin.eu/content/license-categories`.

link to the CCR is instead now available as an explicit link next to the title. As the COMEDI user base is currently expanding, we are responding to user feedback to continuously improve the usability.

3.8 Changes to profiles

A particular feedback which we received referred to rare cases when nodes were not processed. While it is always recommended to create metadata based on a stable CMDI profile, it may happen in certain circumstances that a profile needs to be changed, and consequently metadata may lose elements or components upon importing. Such changes are hopefully rare, but should not be overlooked. The consequences of such changes may be very complex and unforseeable in scope, so that automatic adaptation would be intricate. The strategy adopted in COMEDI for coping with such situations is therefore limited to detecting when a change in a profile has occured, thus leaving it up to the user to fix the metadata. Concretely, COMEDI shows an warning if nodes have not been processed, as exemplified in Figure 15.

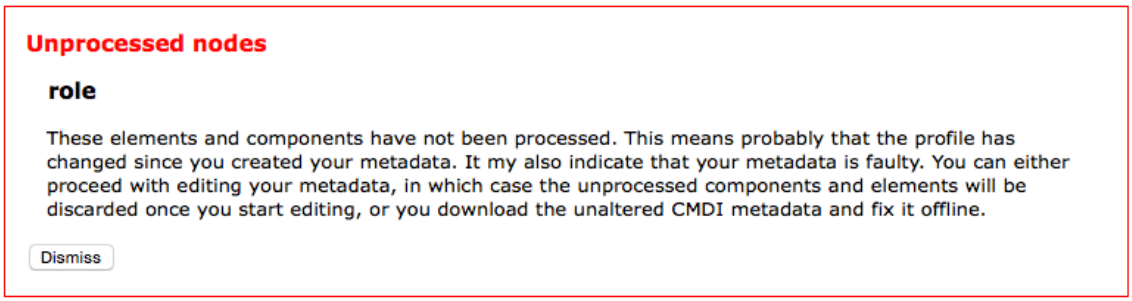

Figure 15: Warning that nodes have not been processed when a profile has been changed.

3.9 Implementation

COMEDI is written in Common Lisp, like the other advanced tools in the emerging CLARINO centre. The advantages of Common Lisp are, among other things, a very high level, extensible language allowing rapid development and seamless integration of components, since even the web server is written in the same language. The metadata are stored in a relational database.

Central to all operations performed on metadata in COMEDI are the CMDI profiles. When needed, a profile is fetched by COMEDI from the Component Registry in the XML-based CMDI Component Specification Language (CCSL). Schema descriptions of metadata are derived from the CCSL format.

The main idea in the implementation of COMEDI is to keep the profile, as a description of possible metadata, tightly connected to the metadata as a valid instantiation of the profile. In concrete terms, both the profile and the (complete or emerging) metadata are aspects of the same in-memory tree representation.

To start with, the profile (in CCSL format) is parsed into a DOM tree using a DOM parser. The DOM parser is however modified in such a way that it adds specific wrapper nodes around the CMD_Component and CMD_Element nodes of the profile: first, each CMD_Component is wrapped into a CMD_Component_Wrapper. Its cardinality attribute is set to 0 or 1 (or even higher), in accordance with the value of the component's CardinalityMin. Then, each CMD_Element is wrapped into a CMD_Element_Wrapper. If the element's CardinalityMin is 1, a CMD_Element_Realization is appended as a child node of the wrapper after the CMD_Element node.

This is the state when the metadata is still empty, or, more precisely, minimal. All components that have to be instantiated have been given cardinality 1 in the wrapper, and all elements that have to be instantiated are represented as a CMD_Element_Realization, but their values are empty.

Editing of the metadata is reflected in changes in the DOM tree: When an instantiation of an element is added in the editor, a new CMD_Element_Realization node is created. When a component is added in the editor, the wrapper's cardinality is increased, and unless the new cardinality equals 1, the CMD_Component node itself is cloned and added as a new child node to the wrapper. Editing of values

simply results in changing the element realization's value attribute. New values are immediately validated against the CMD_Element. Basically the same operations are executed when existing metadata is read.

Starting from the unified DOM tree representation of both the profile and the metadata, the HTML and Javascript/AJAX code of the editor can be generated in a quite straightforward way: The DOM tree is serialized to XML, and appropriate XSL and CSS stylesheets create the HTML code. Since all component and element information of the profile is available in the XML, the stylesheets can create the necessary buttons and input elements to manipulate the metadata, and specifically, will only create those buttons and elements that are in accordance with the profile and result in admissible manipulations.

4 Conclusion and future work

Since CMDI has been adopted as a metadata standard across the whole CLARIN infrastructure, good support for handling this fairly new metadata format may be of great importance. We presume that our work on COMEDI fills a current gap and hope it will be useful to researchers throughout the whole CLARIN infrastructure.

As we have seen in Section 2, previously existing editors have their individual strengths, but they also exhibit weaknesses, in particular concerning user-friendliness and coverage of the full CMDI specification. The development of COMEDI is well motivated since it offers several advantages over existing editors, above all a clear but highly functional web interface abstracting away from technical details in an elegant manner while still keeping the internal structure of the metadata explicit, thus helping to produce metadata faster and more consistently (component cloning, validation of user input). It also features advanced navigation through keyboard shortcuts.

The editor is currently fully functional but will benefit from further testing and user feedback for continued development. Among other things, we foresee the need to improve the search possibilities in metadata. Whereas simple cloning has been implemented, a similar but distinct feature is the availability of instantiated components that can be pointed at from different places, in the sense of structure sharing. When such a component's content is changed in one place, the changes will be reflected in all metadata records referring to it. This feature, which is provided in the META-SHARE editor (cf. Section 2.3), is particularly useful for components describing person or institutional info and the like, where changes should be propagated *passim*. A problem with implementing this is how to assign editing rights in a safe way, to avoid uncontrolled overwriting of existing information that may be shared by several users.

The system can be installed in as many centers as desired, but since metadata creation and management does not require very large computing resources, a few installations may suffice to cover CLARIN-wide needs.

COMEDI is available in the public domain under a BSD license.

5 Acknowledgements

The research reported in this paper has received support from the Research Council of Norway through the CLARINO project.

References

[Broeder et al.2010] Daan Broeder, Marc Kemps-Snijders, Dieter Van Uytvanck, Menzo Windhouwer, Peter Withers, Peter Wittenburg, and Claus Zinn. 2010. A data category registry- and component-based metadata framework. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

[Choukri et al.2012] Khalid Choukri, Victoria Arranz, Olivier Hamon, and Jungyeul Park. 2012. Using the international standard language resource number: Practical and technical aspects. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

[Dima et al.2012] Emanuel Dima, Christina Hoppermann, Erhard Hinrichs, Thorsten Trippel, and Claus Zinn. 2012. A metadata editor to support the description of linguistic resources. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

[Lyse et al.2012] Gunn Inger Lyse, Carla Parra Escartín, and Koenraad De Smedt. 2012. Applying Current Metadata Initiatives: The META-NORD Experience. In *Describing LRs with Metadata: Towards Flexibility and Interoperability in the Documentation of LR (Workshop at LREC'2012)*, Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), pages 20–27.

[Piperidis et al.2014] Stelios Piperidis, Harris Papageorgiou, Christian Spurk, Georg Rehm, Khalid Choukri, Olivier Hamon, Nicoletta Calzolari, Riccardo Del Gratta, Bernardo Magnini, and Christian Girardi. 2014. Meta-share: One year after. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA).

[Soria et al.2012] Claudia Soria, Núria Bel, Khalid Choukri, Joseph Mariani, Monica Monachini, Jan Odijk, Stelios Piperidis, Valeria Quochi, and Nicoletta Calzolari. 2012. The flarenet strategic language resource agenda. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

[Trippel et al.2014] Thorsten Trippel, Daan Broeder, Matej Durco, and Oddrun Ohren. 2014. Towards automatic quality assessment of component metadata. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA).

[Withers2012] Peter Withers. 2012. Metadata Management with Arbil. In *Describing LRs with Metadata: Towards Flexibility and Interoperability in the Documentation of LR (Workshop at LREC'2012)*, Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12) 21.-27. May, pages 72–76.

[Wittenburg et al.2010] Peter Wittenburg, Nuria Bel, Lars Borin, Gerhard Budin, Nicoletta Calzolari, Eva Hajicova, Kimmo Koskenniemi, Lothar Lemnitzer, Bente Maegaard, Maciej Piasecki, Jean-Marie Pierrel, Stelios Piperidis, Inguna Skadina, Dan Tufis, Remco van Veenendaal, Tamas Váradi, and Martin Wynne. 2010. Resource and service centres as the backbone for a sustainable service infrastructure. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).