Implementation of Persistent Identifiers in Clarin.dk

Mitchell Seaton, Bart Jongejan
Center for Language Technology, University of Copenhagen
Njalsgade 140, bygning 25
2300 København S
E-mail: seaton@hum.ku.dk, bartj@hum.ku.dk

Keywords: pid, identifier, id, resource, unique, persistent, repository, infrastructure

1. Requirements for PIDs in Research Infrastructures

In clarin.dk [1], the repository developed in the DK-CLARIN project (Fersøe & Maegaard, 2009) and now maintained by the University of Copenhagen (Offersgaard et al, 2013), we have implemented persistent identifiers (PIDs) in line with CLARIN B-centre requirements [2] and quality data management, storage and sharing practices. CLARIN has a working PID taskforce[3], which has defined a policy, from which a number of recommendations have been outlined for participating CLARIN centres. We wish to highlight the approaches, tasks and issues involved through our implementation for the benefit of other institutions or infrastructures wishing to follow similar steps to apply unique and persistent identifiers to their research data and resources.

1.1 PID Solutions

There are a number of PID systems available for research infrastructures (ARK [4], Handle System [5], PURL[6]). CLARIN deliverable D2.2 (Wittenburg, 2008) reports on a survey of PID systems and services existing at that time. The document expresses a preference for the Handle System from CNRI and concludes with a list of recommendations. In 2009, the European Persistent Identifier Consortium (EPIC) [7] was established to provide PID services to the European Research Community. EPIC is based on Handle System PIDs. Handle identifiers can be resolved to real addresses suitable for referencing resources of CLARIN-centre infrastructures. Following our discussions with and guidance from the Max Planck Institute in Nijmegen, with an overview of their PID implementation, we reviewed the EPIC services and support, the Handle System, and integration solutions.

Handle System is in two parts; a global registry[8] and individually managed local handle services. This distributed model ensures scalability and redundancy.

1.2 Infrastructure Support for PIDs

Clarin.dk uses eSciDoc [9] middleware to support our Fedora Commons [10] repository and MarkLogic [11] database. Resources in our repository can be resource type specific items with associated content-files or collections. All resources in our repository, metadata as well as content-files, have PIDs. Since eSciDoc software provides version management for the resources, we assign PIDs also to individual versions of a resource.

PID fields in eSciDoc metadata can be left empty or unassigned when creating resources, however once assigned, a PID value cannot be modified, unless a new version of the resource is created. PIDs will be assigned to each new resource version in our repository.

2. Preparation and Decision-Making Stages

Following a review of PID support and inclusion with our existing infrastructure, in alignment with PID requirements for CLARIN, we decided to utilise our own local handle server and in addition, a mirror handle server hosted in another geographic location.

To acquire a Handle System prefix (11221 in our case) we had to consent to the Handle System *Policies & Procedures*[12], which, among other requirements, states that we shall maintain a reasonable quality of service. Once we had our own handle servers up and running, we had a fast and easy way to maintain services for creating and resolving our own PIDs. The global handle resolver (proxy server) at Handle System (hdl.handle.net) forwards all resolution requests for PIDs with our prefix to our own handle resolving service or its associated mirror. The distributed architecture of Handle System is expected to scale well with the use of distributed mirror servers and at a relatively low cost. Currently, the annual service fee for the Handle System prefix is 50 US$, without additional costs or limitations on the number of PIDs that we create.

We decided to assign PIDs to each repository resource

[1] https://clarin.dk
[2] http://www.clarin.eu/content/checklist-clarin-b-centres
[3] https://clarin.eu/content/pid-taskforce-discussion-document
[4] https://confluence.ucop.edu/display/Curation/ARK
[5] http://www.handle.net
[6] https://purl.oclc.org/
[7] http://www.pidconsortium.eu
[8] http://hdl.handle.net
[9] https://www.escidoc.org
[10] http://fedora-commons.org
[11] http://www.marklogic.com
[12] http://hdl.handle.net/4263537/5012

(object PID), to all non-metadata files of a resource (content PID), and to each version of the metadata[13] of the resource (version PID) in our repository. Therefore, each resource added to our repository will have a minimum of three PIDs associated.

PIDs are stored in the CMDI[14] (Component MetaData Infrastructure) metadata records that we have developed. PIDs occur in two sections in a metadata record: a version PID in the Header section as self-reference and content PIDs in the Resources section as resource references to content-files (Fig. 1). Object PIDs are currently not stored in our CMDI metadata records.

```
<?xml version="1.0" encoding="utf-8"?>
<CMD […]>
  <Header>
    <MdSelfLink>
       hdl:11221/90D1-8104-0082-B-8@md=cmdi
    </MdSelfLink>
    …
  </Header>
  <Resources>
    <ResourceProxyList>
      <ResourceProxy id="_611003">
        <ResourceRef>
          hdl:11221/90D1-8104-0003-7
        </ResourceRef>
        …
      </ResourceProxy>
      …
    </ResourceProxyList>
    …
  </Resources>
  <Components>
    <teiHeader>
      …
    </teiHeader>
  </Components>
</CMD>
```

Figure 1. CMDI metadata with PIDs. *MdSelfLink*: version PID, *ResourceRef*: content PID.

2.1 Handle Format

Our approach to the PID-format was to apply the same structure as used in EPIC handles, so as to have a valid EPIC-supported PID structure in the case of future migration to their services. As a basis, each PID has a structure in the format *prefix/XXXX-XXXX-XXXX-C* where *X* is a uppercase hexadecimal digit, *C* is a hexadecimal checksum digit, and *prefix* is the global handle prefix, 11221 in our case.

The group of twelve hexadecimal digits uniquely identifies the object within the name space of the handle prefix. The 48 bits encoded by these hexadecimal digits are formed from the resource's eSciDoc ID, which also uniquely identifies resources within clarin.dk. This is done as follows. An eSciDoc ID features a prefix *dkclarin:* followed by the resource number reference, currently consisting of 6 decimal digits. Our implementation converts strings of up to eight octets, specifically the last eight characters of the eSciDoc ID reference, for example the underlined substring in *dkclari**n:611022***, to eight sixbit BCD bytes, together producing the 48 bits that are encoded by the twelve hexadecimal *X*-digits above. Using this scheme, we can compute unique handle suffixes for all eSciDoc IDs up to *dkclarin:99999999*, but also for other alphanumeric values, such as short values like *helpdesk*. There is a chance that the generated suffix already is in use if a handle must be generated from a value longer than eight characters, for example *clarinhelpdesk*. Such collisions can be resolved by sequentially searching for an unused number close to the generated number. This can be done at a low cost, because the described mapping from strings to 48 bit numbers almost guarantees that there will be a swathe of unused numbers close to every generated number. As we currently only deal with eSciDoc ID references, this collision detection and resolving is not implemented yet in our PID generation service.

If we instead had chosen to think of the 48-bit number as the numerical part of the eSciDoc ID, we could manage handles for 281 474 976 710 655 of our eSciDoc objects, and not for anything else. We think that 100 000 000 eSciDoc objects is a limit that will not be transgressed in the foreseeable future of clarin.dk.

The checksum digit is computed on the basis of the 48 bits encoded in the twelve hexadecimal digits using the ISO 7064, MOD17,16 Hybrid Recursive method.

Two more fields can be appended to a PID, see Fig. 2.

First, a hexadecimal value indicating the version of the resource is appended to the handle for a version PID as part of an OPTIONAL_APP of the EPIC-based format.

Finally, a handle can be combined with a part-identifier (PI). As the name indicates, part-identifiers can be used to address a part of a resource. In order to support CLARIN B-centre requirements for PIDs, handles with the part-identifier *@md=cmdi* are resolved to the CMDI metadata associated with the referenced resource. Technically, the Handle System software in our handle server, upon detecting a part-identifier in a handle, uses a handle template configured in our local service to resolve the handle to an alternate URL that provides the CMDI metadata of the resource

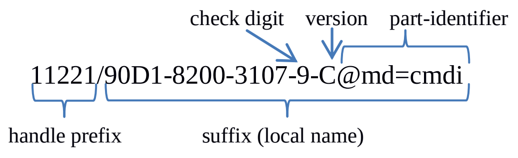

Figure 2. Clarin.dk version PID with part-identifier

For development and testing we have used an optional prefix, OPTIONAL_PRE of the EPIC-based format, with value *DEV-* with our handle format.

[13] A change to a metadata field, the addition of a relation to another resource or the replacement of a content-file can only be done in a new version.
[14] http://www.clarin.eu/content/component-metadata

2.2 Handle Values

Handle System handles have several fields. The most important Handle field is *URL*, which contains the URL address a Handle will resolve to. Additional custom data fields can be stored in a Handle, to hold arbitrary data related to the resource. In handles that refer to content-files we store the MD5 checksums of those content-files as values of type HASH_ALG_MD5[15]. No other additional custom data fields are used in our current handles.

The URL Handle Value can point to any relevant and valid URL, however using our typical resource web-addresses as values was considered impractical, because we probably would have to update all URLs in the foreseeable future as we move to a new web application or front-end. As we needed to respond with a variety of resource types (text, video, etc.), we decided to translate a PID to an URL built up from the eSciDoc content model ID (*cm*), the eSciDoc ID (*eID*), the resource's version and, if the PID includes a part-identifier (for CMDI metadata), the name of that part, and to let a single web service (see 3.2) http://clarin.dk/handle interpret the semantic information in the URL to redirect the request to the final URL:

Content
PID: 11221/*XXXX-XXXX-XXXX-C*
URL: http://clarin.dk/handle/*cm*/*eID*/*content-id-ref*

Object
PID: 11221/*XXXX-XXXX-XXXX-C*
URL: http://clarin.dk/handle/*cm*/*eID*

Version
PID: 11221/*XXXX-XXXX-XXXX-C-V*
URL: http://clarin.dk/handle/*cm*/*eID*?v=*version-no*

Object with part-identifier for CMDI
PID: 11221/*XXXX-XXXX-XXXX-C*@md=cmdi
URL: http://clarin.dk/handle/cmdi*?*http://clarin.dk/handle /*cm*/*eID*

Version with part-identifier for CMDI
PID: 11221/*XXXX-XXXX-XXXX-C-V*@md=cmdi
URL: http://clarin.dk/handle/cmdi?http://clarin.dk/handle /*cm*/*eID*?v=*version-no*

Content components represent each content-file of an item resource in our repository. A content component doesn't contain a CMDI metadata record, therefore the associated content PIDs cannot resolve to a valid CMDI metadata resource using the part-identifier *@md=cmdi*.

3. Support Tools & Services

3.1 Workflow program

Because the existing repository already was quite big, we implemented an automatic workflow[16], using job queues, to complete bulk updates of our existing resources with PIDs. New CMDI metadata records are created from the existing metadata, supplemented with the newly generated PIDs for the resource metadata and content-files.

3.2 Redirection service

As a consequence of our design of Handle URL values, a service was required that could redirect requests that were based on the path schema used by these URL values. (See 2.2). We created a basic HTTP web-service (redirection service) to redirect the user or machine to the appropriate resource: web page, CMDI metadata, or content-file download. This basic service operates separately from our Java-based web applications. Technically, the service does support HTTP-Accept header requirements for PIDs referencing CMDI metadata resources.

4. Assignment of PIDs to Resources

4.1 Updating Existing Resources

Initially, we have worked with small collections (text, TEIP5) of existing resources to generate PIDs together with a new CMDI metadata record applied in a single version update. The automatic workflow has been used to process these resources, thereby enabling them to be harvested by CLARIN VLO[17].

4.2 Upgrading Web Applications

To support PIDs in our infrastructure, we have upgraded the depositor web application with two new workflow steps that (1) generate PIDs for the resource, version and content-files, and (2) assign them in the deposit process.

All eSciDoc IDs for the newly created resource are assigned during our resource ingest (deposit), including all content components, which also have their own eSciDoc IDs. No PIDs are initially assigned and referenced in CMDI metadata records. As our PID implementation requires eSciDoc IDs for PID generation, we apply PIDs in a new version of the resource. See Fig 1.

Collection deposits require all member resources to have a version PID on its latest-release version, and a CMDI metadata record. New version PIDs are created during the deposit job process for each member resource, and assigned following validation of the Collection CMDI metadata record.

4.3 Resolving & Testing PIDs

Handle PIDs can be resolved and inspected using the global Handle System proxy server http://hdl.handle.net or our local resolver http://clarin.dk:8000.

Requesting a valid handle URL in a web-browser should redirect successfully. The Handle System also

[15] http://www.handle.net/hs-source/api_javadoc/net/handle/hdllib/Common.html#HASH_ALG_MD5
[16] https://github.com/meaton/escidoc-cmdi-pid-workflow

[17] http://www.clarin.eu/content/virtual-language-observatory

provides command-line tools to test and administer the local service.

An object and version PID (without part-identifier) will redirect to the resource's web-page, from where it can be downloaded. Whereas a version PID always redirects to the same version of a resource, an object PID always redirects to the latest released version of a resource. With the part-identifier extension *@md=cmdi*, resolving will redirect to the CMDI metadata of a resource.

Here are some examples of redirections, leading from PID to destination[18]. Redirections are indicated with =>:

11221/90D1-8104-0082-B-8 (**Version**)
=> http://clarin.dk/handle/14001/611022?v=8
=> https://clarin.dk/clarindk/item.jsp?id=dkclarin:611022:8

11221/90D1-8104-0082-B-8@md=cmdi (**Version+PI**)
=> http://clarin.dk/handle/cmdi?http://clarin.dk/handle/14001/611022?v=8

11221/90D1-8104-0006-1 (**Content)**
=> http://clarin.dk/handle/14001/611022/611006
=> https://clarin.dk/clarindk/download-proxy.jsp?item=dkclarin:611022&component=dkclarin:611006

We tested resolving a valid version PID, with a part-identifier *@md=cmdi*. We have confirmed that if the HTTP-Accept header *text/html* is defined, the CMDI output is transformed to HTML and returned in the response. If the HTTP-Accept header is defined *application/x-cmdi+xml* then the content-type of the response is provided as *application/x-cmdi+xml*. Likewise for *application/xml*, which is the default Content-Type returned. Request headers can be configured and tested using a web-browser plug-in, a developer console or the curl utility[19].

A content PID will redirect to a download page for the content-file. Restrictions on some resources in our repository prevent unauthorised users from accessing the resource. Therefore, users may be redirected to accept the associated CLARIN license before being able to download the content-file. Authorised users of clarin.dk have their acceptance saved.

5. Experiences & Conclusions

When a new resource is deposited, our depositor workflow first creates an eSciDoc item without a PID. A PID is then computed from the eSciDoc ID of that item. In an update step, the self-reference field in the metadata of the already deposited resource is updated with this PID. In the same update step, the PIDs of components (content-files) are added to the CMDI metadata.

In an alternative approach, we could have created all handles first, for example using a random number generator and with a dummy value in the URL field of the handles. Then, using those (cached) PIDs, we could have created the complete resource in a single step. In this scenario the workflow would not have been shorter, because we then would have to update the handles with the actual URLs of the deposited resources, but we would not necessarily be required to create a new version of the resource's metadata the instant after the first version was created. That alternative scenario requires the wasteful housekeeping of two independent systems to identify the same resources. The ideal scenario, which we considered but were not able to implement due to errors in testing, is to have eSciDoc middleware create an eSciDoc ID and then, using a configured PID service, to create a handle (PID), computed from that eSciDoc ID. Finally, eSciDoc would ingest the resource in the repository with the created PID references assigned, without any need of later updates. We find our currently implemented solution to be operating successfully, and will fulfil our immediate and timely needs.

6. Acknowledgements

Thanks go to Daan Broeder, Willem Elbers and Dieter van Uytvanck at the Max Planck Institute in Nijmegen for giving us an excellent introduction to the practicalities of PIDs, to Gudrun Siedersleben for providing PID Manager source code, and to Michael Hoppe for giving eSciDoc support.
Our visit to the Max Planck institute in Nijmegen was funded by a CLARIN ERIC Mobility Grant.

7. References

Fersøe, H & Maegaard, B. (2009). CLARIN in Denmark – European and Nordic Perspectives. In: *Nordic Perspectives on the CLARIN Infrastructure on Common Language Resources*, NEALT Proceedings Series, Vol. 5, pp. 6-11. Electronically published at Tartu University Library (Estonia) http://hdl.handle.net/10062/9944.

Offersgaard, L., Jongejan, B. & Hansen, D. H. (2013). CLARIN-DK – status and challenges. In: Proceedings of the workshop on Nordic language research infrastructure at NODALIDA 2013. Linköpings universitet: Linköping University Electronic Press, s. 21-32 12 s. (NEALT Proceedings Series; Nr. 20).

Wittenburg, P. (2008), *Persistent and unique Identifiers*, CLARIN specification document CLARIN-2008-2. Daan Broeder, Malte Dreyer, Marc Kemps-Snijders, Andreas Witt, Marc Kupietz, Peter Wittenburg (Eds.). http://hdl.handle.net/1839/00-DOCS.CLARIN.EU-30

[18] The redirection paths can be made visible with the curl utility: curl –L –I http://hdl.handle.net/*PID*?auth=true
[19] For example:
curl -H "Accept: text/html" -L http://hdl.handle.net/11221/90D1-8104-0082-B-8@md=cmdi?auth=true