SSHOpenCluster
Social Sciences & Humanities Resources

Vocabulary Versioning

SSH Vocabulary Commons

Discussion Paper

Document information

Dissemination Level	PU
Date	18/08/2023
Type	Discussion Paper
Number of Pages	21
Version	1.0

Abstract: Versioning is an important aspect for the interoperability of Controlled Vocabularies (CVs) but there is currently a lack of detailed recommendations for versioning practice. This discussion paper is intended to stimulate debate and feedback on a practical approach to the versioning of Controlled Vocabularies.

Author

Organisation	Name	Contact Information
UK Data Service	Darren Bell	dbell@essex.ac.uk

Contributors

Organisation	Name	Contact Information
CLARIN	Daan Broeder	d.g.broeder@uu.nl
CESSDA/FSD	Mari Kleemola	mari.kleemola@tuni.fi
University of South Wales	Douglas Tudhope	douglas.tudhope@southwales.ac.uk
DANS-KNAW	Menzo Windhouwer	menzo.windhouwer@di.huc.knaw.nl

Executive Summary

Work towards a Social Science and Humanities (SSH) Vocabulary Commons started during the SSHOC project. CESSDA, CLARIN, DARIAH and E-RIHs have a common interest to use and manage Controlled Vocabularies (CVs) collaboratively. An important aspect for the technical or format interoperability of vocabularies is versioning but there is currently a lack of detailed recommendations for vocabulary versioning practice.

This document proposes an approach for vocabulary versioning that enables two primary use cases: (1) referencing a SKOS concept as an object at a point in time - a "synchronic" approach which requires an explicitly versioned URI, and (2) referencing a SKOS concept as the most current version (albeit accompanied by provenance information represented in various SKOS predicates such as skos:changeNote) - a "diachronic" approach that does not require an explicitly versioned URI.

The opening section provides and justifies the proposed version approach. Following sections cover the implications and ideal practice for URI construction and provenance.

This discussion paper is released to the community with a view to seeking feedback and consensus.

Abbreviations and Acronyms

Abbreviation	Full name
CESSDA	Consortium of European Social Science Data Archives
CLARIN	Common Language Resources and Technology Infrastructure
CV	Controlled Vocabulary
DARIAH	Digital Research Infrastructure for the Arts and Humanities
DDI	Data Documentation Initiative
DNS	Domain Name System
ELSST	European Language Social Science Thesaurus
E-RIHS	European Research Infrastructure for Heritage Science
FAIR	Findable, Accessible, Interoperable, Reusable
HTML	Hypertext Markup Language
HTTP(S)	Hypertext Transfer Protocol (Secure)
JSON-LD	JavaScript Object Notation for Linked Data
RDF	Resource Description Framework
SKOS	Simple Knowledge Organization System
SSH	Social Sciences and Humanities
SSHOC project	Social Sciences & Humanities Open Cloud project
TXT	text
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
XML	Extensible Markup Language

Table of Contents

1. Introduction

Work towards a Social Sciences and Humanities (SSH) Vocabulary Commons started during the SSHOC project[1]. CESSDA, CLARIN, DARIAH and E-RIHs have a common interest to use and manage Controlled Vocabularies (or just "vocabularies") collaboratively. They all see vocabularies as first-class citizens or FAIR data objects in their own right. An important aspect for the technical or format interoperability of vocabularies is versioning. Changes in vocabularies should be made explicit and be documented but there is currently a lack of detailed recommendations for vocabulary versioning practice.

This document proposes an approach for vocabulary versioning that enables two primary use cases:

(1) referencing a SKOS[2] concept as an object at a point in time - a "synchronic" approach which requires an explicitly versioned URI, and

(2) referencing a SKOS concept as the most current version (albeit accompanied by provenance information represented in various SKOS predicates such as skos:changeNote) - a "diachronic" approach that does not require an explicitly versioned URI.

This discussion paper is released to the community with a view to seeking feedback and consensus. Responses received will be integrated into a future position paper. The opening section provides and justifies the proposed version approach. Following sections cover the implications and ideal practice for URI construction and provenance.

[1] SSHOC project: https://sshopencloud.eu/project
[2] Simple Knowledge Organization System: https://www.w3.org/2004/02/skos/

2. A proposed vocabulary versioning approach

2.1 Vocabulary representations

For the purpose of this document, Controlled Vocabularies ("CVs" or just "vocabularies") are considered to be sets of concepts that can be arranged either as a simple flat list or in a hierarchical structure. Within the SSH community, there are a number of formal standards used to represent vocabularies, principally DDI CodeLists/CategorySchemes[3], SKOS and occasionally Genericode[4]. Less formalised (and hence less interoperable) representations range from TXT files to Excel spreadsheets. This document considers versioning within the context of SKOS representations of vocabularies but makes no prescriptions or recommendations about particular RDF serialisations like RDF/XML or JSON-LD, for example.

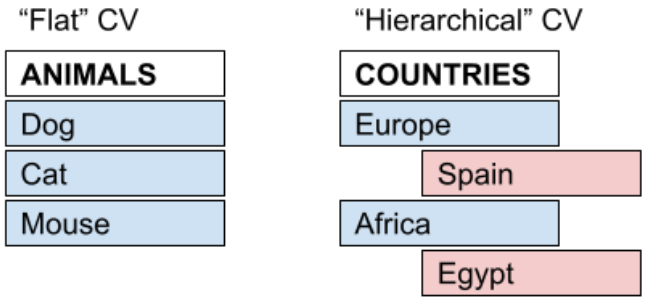

Figure 1

Figure 1 above presents two examples of CVs. Using SKOS terminology, the first example is a ConceptScheme labelled 'ANIMALS' which has three Concepts: Concept 1 is labelled "Dog"; Concept 2 is labelled "Cat" and Concept 3 is labelled "Mouse". The second ConceptScheme is labelled "COUNTRIES" and has four concepts labelled "Europe", "Spain", "Africa" and "Egypt". These are arranged in a meaningful hierarchy, unlike the first "flat" example.

In order for CVs to be useful, they will be published in a system[5] and "consumed" for some purpose by either machines or humans. A typical use case for this kind of consumption would be two different archives referencing the same concept from two different DDI documents in order to perform useful operations related to semantic comparability or federated discoverability.

[3] DDI Codes and Categories : https://doi.org/10.5281/zenodo.5180592
[4] Genericode: http://docs.oasis-open.org/codelist/cs-genericode-1.0/doc/oasis-code-list-representation-genericode.html
[5] Typically software, but in previous decades, ELSST, for example, was issued as a paper document.

2.2 What actually changes over time

In order to properly assess how we best approach versioning with SKOS vocabularies, we need to understand precisely the scope of what we are managing. With typical versioning of a typical entity like a dataset, we might conceptualise it thus:

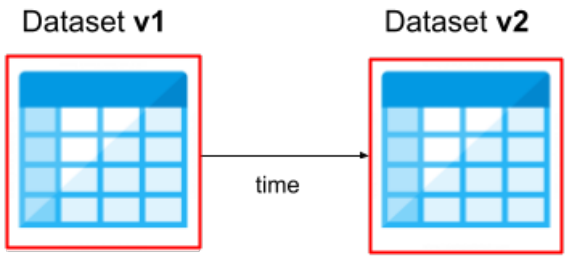

Figure 2

While simplistic, Figure 2 nonetheless illustrates a foundational consideration of versioning: where are the boundaries of the object that is subject to version control? In Figure 2, the red lines indicate where we stop caring about the versioning context of the object; it might be part of a longitudinal collection for example, but if we focus solely on the Dataset object itself, the red lines bound the scope of our versioning concerns. Conversely, within this boundary, we are not concerned about versioning individual cells or columns or rows as child objects of the Dataset object (in this particular example at least), even though granular changes to cells for example will result in a new version of the Dataset object.

When dealing with SKOS vocabularies, the boundaries can be asserted strictly at only two levels of granularity[6]. To some extent, this is what makes SKOS a "Simple" Knowledge Organization System. Here, we have **two** key entities (or "classes" to be more technically correct), namely the CV (a SKOS "ConceptScheme"[7]) and its constituent items (SKOS "Concepts"[8]). Both classes are versionable:

[6] Strictly speaking, skos:Collection is a third available class but out of scope for this document, which focuses on simple CV use cases.
[7] SKOS Concept Scheme : https://www.w3.org/TR/skos-primer/#secscheme
[8] SKOS Concepts: https://www.w3.org/TR/skos-primer/#secconcept

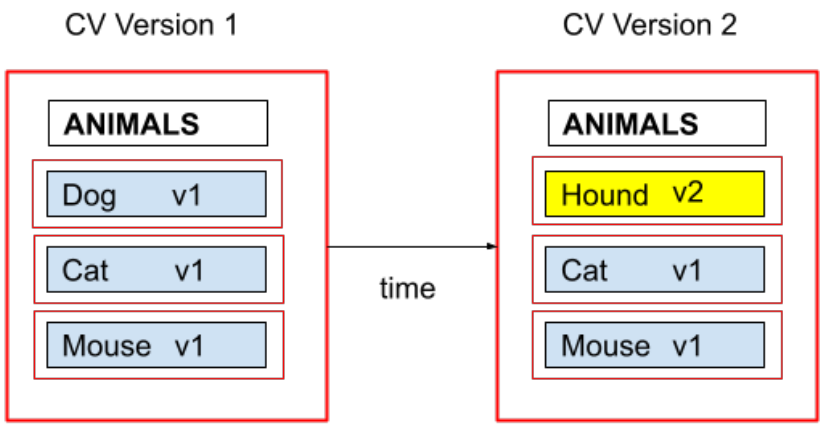

Figure 3

In Figure 3, we have a simple CV labelled “Animals” in English and the first Concept labelled “Dog” (in English) is re-labelled as “Hound”. Not only do we need to consider our object boundary as the CV itself which moves from v1 to v2 but each of its three constituent concepts (labelled “Dog”, “Cat” & “Mouse” respectively in English) are also discrete objects and hence subject to versioning over time.

Changes to CVs could also involve merely a simple change to the title of the CV itself. For example, in our working example, the CV labelled as “Animals” in English might be re-labelled as “Pets” and that might be the only change that moves the CV from v1 to v2, that is to say, no child concepts themselves might have changed - see Figure 4 below:

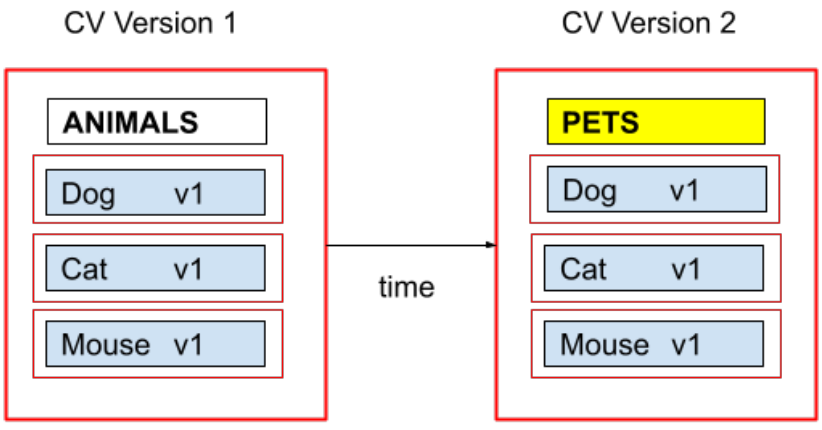

Figure 4

In conjunction with the previous figure, this neatly illustrates a problem that we need to manage: when we have a parent and a child, what are the rules when only the parent or only the child object changes - in this case, either the ConceptScheme or the Concept. Can we truly treat them independently?

2.3 "Wholesale Versioning" and redundancy

Figure 3 illustrates an approach that presumes that a change to the CV's contents constitutes a change to the parent CV digital object and therefore warrants a version increment of the parent CV itself: we say that a change to the child "propagates up" to the parent[9]. However, apart from the concept labelled "Dog" which was re-labelled "Hound" (hence the concept was incremented to v2), the other concepts remain unchanged at v1. Representing all of this in SKOS could look something like this at the URI level:

Table 1 - CV Version 1

Concept Scheme	`https://example.org/Animals/1`
Concept	`https://example.org/Dog/1`
Concept	`https://example.org/Cat/1`
Concept	`https://example.org/Mouse/1`

Table 2 - CV Version 2

Concept Scheme	`https://example.org/Animals/2`
Concept	`https://example.org/Hound/2`
Concept	`https://example.org/Cat/1`
Concept	`https://example.org/Mouse/1`

One might notice that the Cat Concept v1 (with URI `https://example.org/Cat/1`) is a member of two different versions of the CV. This is perfectly fine at a technical level as you can simply use a SKOS inScheme predicate to assert that the Concept `https://example.org/Cat/1` is a member of two ConceptSchemes: `https://example.org/Animals/1` and `https://example.org/Animals/2`.

Similarly, Figure 4 presents a scenario in which the parent object changes: CV "Animals" is renamed to "Pets" but the version increment does *not* propagate down to the child concepts, which remain at v1.

Table 3 - CV Version 1

Concept Scheme	`https://example.org/Animals/1`
Concept	`https://example.org/Dog/1`
Concept	`https://example.org/Cat/1`
Concept	`https://example.org/Mouse/1`

[9] If we consider a loose dataset analogy, changing the value in an individual cell in an Excel spreadsheet leads to a new version of the spreadsheet file.

Table 4 - CV Version 2

Concept Scheme	`https://example.org/Pets/2`
Concept	`https://example.org/Dog/1`
Concept	`https://example.org/Cat/1`
Concept	`https://example.org/Mouse/1`

This "sparse" approach to non-aligned version numbers between a ConceptScheme and its constituent Concepts can be counter-intuitive for a lot of metadata managers and casual users.

A spreadsheet analogy may be helpful. Imagine an Excel file with 10 rows and 10 columns of data i.e. 100 cells. The approach above would require you, at the point of creating the spreadsheet, to create 100 persistent identifiers for each cell and one for the spreadsheet. From the outset, you will need to manage persistent identifiers for 100 cells, create a new versioned URI for each cell after each change and ensure that every versioned instance of a cell version was explicitly related to a specific spreadsheet version.

Reverting back to our SKOS perspective, let us assume that you are on version 20 of a ConceptScheme. It contains 20 Concepts, each of which may have a different version number (assuming for example that every concept was changed at least once for versions 1 through 20 of the ConceptScheme). The assortment of non-aligned version numbers at the concept level, while formally complete and technically correct, can be hard to disentangle for data managers and developers, let alone casual users and browsers of a CV.

Additionally, the main use case for asserting a specific Concept version (independent of its parent CV context) is that you may want to reference it from a DDI document at a specific point in time. If the only information in my DDI document about the concept is that its URI is `https://example.org/Cat/1` and `https://example.org/Cat/1` is associated with twenty different versions of the Animals ConceptScheme, then writing business logic for applications potentially becomes more complex, as I have to perform additional work to establish unambiguously which instance in time of `https://example.org/Cat/1` I am actually intending to reference.

All of this has been leading up to what is an essentially imperfect but more practical and intuitive solution and we refer to this as "wholesale versioning". Essentially, this means that for a SKOS ConceptScheme and its constituent Concepts, at the point of publishing (i.e. the point when we would necessarily want to increment our version numbers) we increment and align *both* the ConceptScheme version number and all of the Concept version numbers, regardless of whether an individual Concept has changed or not.

This has consequences for the construction of concept URIs. Instead of each Concept effectively having its own URI and ambiguous namespace (See Tables 1 and 2 above), we associate all concepts consistently with the parent CV namespace e.g., instead of:

CV Version 1 -

Concept Scheme	`https://example.org/Animals/1`
Concept	`https://example.org/Dog/1`
Concept	`https://example.org/Cat/1`
Concept	`https://example.org/Mouse/1`

CV Version 2 -

Concept Scheme	`https://example.org/Animals/2`
Concept	`https://example.org/Hound/2`
Concept	`https://example.org/Cat/1`
Concept	`https://example.org/Mouse/1`

we instead base a URI for concepts derived from the parent CV namespace:

CV Version 1 - *unified namespace of* *`https://example.org/Animals`*

Concept Scheme	`https://example.org/Animals/1`
Concept	`https://example.org/Animals/1/Dog`
Concept	`https://example.org/Animals/1/Cat`
Concept	`https://example.org/Animals/1/Mouse`

CV Version 2

Concept Scheme	`https://example.org/Animals/2`
Concept	`https://example.org/Animals/2/Dog`
Concept	`https://example.org/Animals/2/Cat`
Concept	`https://example.org/Animals/2/Mouse`

Following on from our examples in Figure 3 and Figure 4, this approach would manifest itself as follows (Figures 3b and 4b):

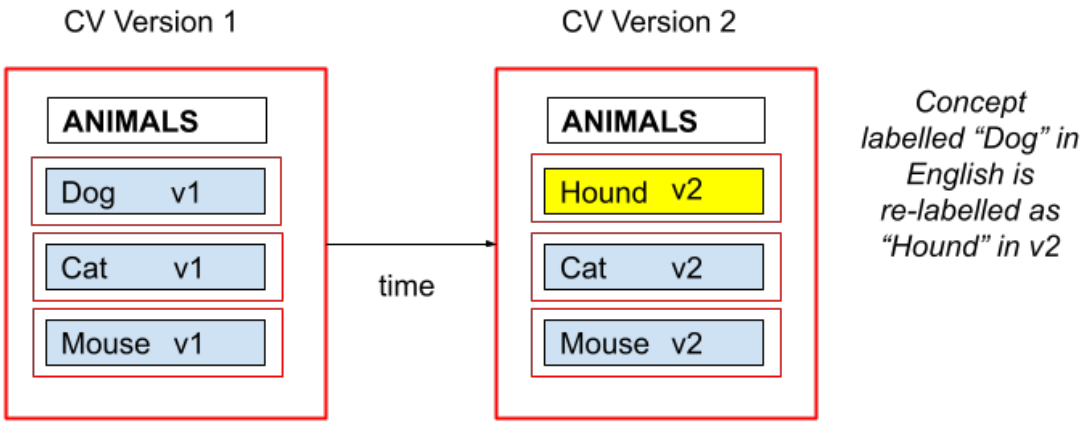

Figure 3b
Versioning Propagated UPWARDS

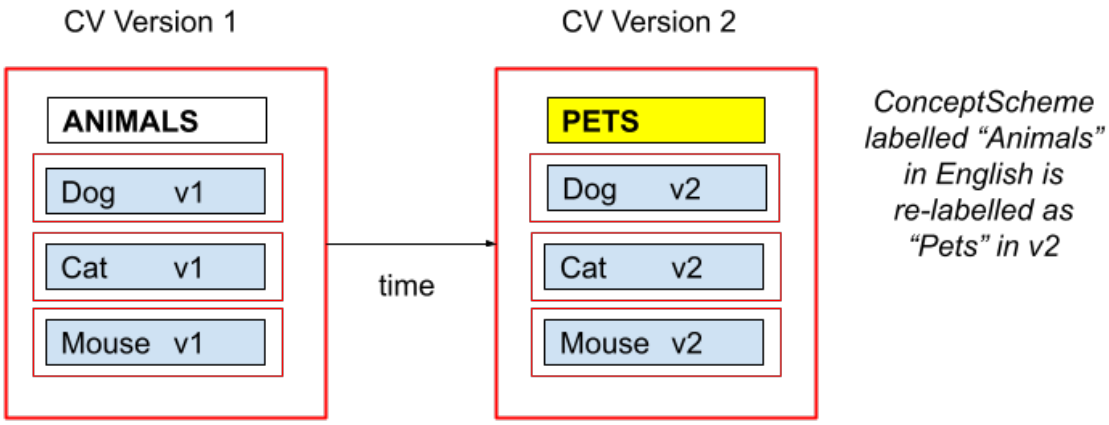

Figure 4b
Versioning Propagated DOWNWARDS

While this approach is unquestionably less "elegant" in that we increment the version number for a concept object not exclusively based on whether it has changed or not, but rather as a consequence of its association with the ConceptScheme object and associated Concepts, at least one of which will have changed).

Now when I reference `https://example.org/Animals/2/Cat` from my DDI document, I know unequivocally that I am referencing a particular instance of the Concept within the context of a Concept Scheme published at a particular point in time.

2.4 Versioning number schemes

So far we have used the simplest and most obvious version numbering scheme available to us i.e. an integer[10]. When we create a new version, we simply add 1 to the version number. This is perfectly acceptable if all we want to do is assert that there has been "a change" of some unspecified type.

Often however, we want to indicate the granularity of change which helps consumers of the CV to determine how impactful the change is. A common version number syntax in common use and recommended by CESSDA is semantic versioning[11]. This has emerged from software development and works on the basis of three integers delimited by a period, representing "major", "minor" and so-called "patch" changes (for "patch", read "sub-minor" in the context of this discussion).

So, for example, if our CV number is currently **2.0.0**, then a major change would be represented as **3.0.0**, a minor change would be represented as **2.1.0** and a sub-minor change would be represented as **2.0.1**.

So far so simple, but it is important that there be a shared and documented understanding of what "major", "minor" and "sub-minor" mean for a particular community. A typographic correction may be classed as trivial or important depending on the use case and this document does not prescribe what kinds of changes should be classed as "major", "minor" or "sub-minor" but it does recommend that there be a mapping of change types to these levels of versioning granularity. A good example is provided by the DDI Alliance which has an enumeration of "major", "minor" and "sub-minor" changes and examples of operations on CVs that warrant these designations[12].

[10] Strictly speaking, we don't even to have to use integers. There is no special reason why "v1", "v2" and "v3" could not be represented by some other string like "uyii7", "kQuuhi" and "~#%$". However, not only would we need to manage and maintain additional semantic *information to interpret these strings correctly, it would be complex to consistently parse such constructions. For all practical purposes, versioning syntax is almost always based around some integer scheme*.
[11] Semantic versioning: https://semver.org/
[12] DDI Alliance CV policy: https://ddialliance.org/controlled-vocabularies#policy

3. URI Construction

3.1 Parts of a well-formed URI

A URI is an identifier for ConceptScheme and Concept resources. One essential thing to grasp is that in this context URIs are *NOT URLs for web pages*. Consider this example exported from an earlier incarnation of the CESSDA Vocabulary Manager:

`https://ddialliance.org/Specification/DDI-CV/AggregationMethod_1.1.html#Mode`

This is not an optimally constructed URI for the purposes of resolving to RDF data. Firstly, it's referencing an HTML web page and secondly, it is using a hash to represent an anchor in the page. Although linked data URIs can superficially resemble URLs, they are intended primarily to retrieve data, not to retrieve browsable web content.

So what are the various elements we should consider as the constituent parts of a well-formed URI in the Controlled Vocabulary arena? The potential segments are summarized in Table 5 and then explained in more detail.

Table 5 - Potential segments in a SKOS URI for a Concept.

#	Segment	Example
1	BaseURI	`https://example.org/`
2	ConceptSchemeIdentifier	Animals
3	ConceptSchemeVersion	2.0.0
4	ConceptIdentifier	Dog
5	ConceptVersion	*Implied by ConceptSchemeVersion* *See earlier discussion on "wholesale" versioning*

Example: `https://example.org/Animals/2.0.0/Dog`

BaseURI
The base URI must be resolvable through DNS to an HTTP(S)-based service that can provide more information about the URI that is passed to it.

Another real-world example: the ELSST concept "ACADEMIC ABILITY" (concept version 3) has a URI of `https://elsst.cessda.eu/id/3/bf0d664a-e89a-4ec3-80d2-04f664b359ab` and the BaseURI is `https://elsst.cessda.eu/id/`

ConceptSchemeIdentifier

In the ELSST example above, the BaseURI actually encompasses the ConceptSchemeIdentifier. That is to say, because https://elsst.cessda.eu/id/ as a web address only hosts one ConceptScheme i.e. ELSST, there is no need to further partition the URI to indicate a specific ConceptScheme. Compare this to two examples of ConceptSchemes from the EU's multilingual and multidisciplinary thesaurus Eurovoc[13]:

- Combined Nomenclature, 2019: http://publications.europa.eu/resource/authority/cn2019
- SDMX glossary 2018: http://publications.europa.eu/resource/authority/sdmxglossary2018

Here you can see that the BaseURI is `http://publications.europa.eu/resource/authority/` but because many hundreds of ConceptSchemes are being hosted, it makes sense to append the ConceptScheme Identifier as well.

ConceptSchemeVersion

It is recommended to indicate the version of the concept scheme in the ConceptScheme URI based on the following pattern:

```
<BASE_URI>/<CONCEPTSCHEME_IDENTIFIER>/<CONCEPTSCHEME_VERSION>/
```

For example: `https://example.org/Animals/2.0.0/`
where 2.0.0 is the numeric version of the ConceptScheme.

ConceptIdentifier

Following the "wholesale" versioning pattern identified earlier, the version of the concept is implied by the version of the concept scheme, so the following pattern results for a Concept URI:

```
<BASE_URI>/<CONCEPTSCHEME_IDENTIFIER>/<CONCEPTSCHEME_VERSION>/<CONCEPT_IDENTIFIER>
```

For example: `https://example.org/Animals/2.0.0/Dog`

It is the strong recommendation of this document that the use of natural language be *avoided* in URIs, unless it can be unequivocally guaranteed that any natural language string in the URI will not change over time. Doubtless, avoiding natural language sacrifices human readability but it is arguable that URIs are not primarily for human consumption in any case.

Continuing the concept URI example above, a more robust approach would be to render

`https://example.org/Animals/2.0.0/Dog`

instead as:

`https://example.org/652/2.0.0/132`

where '652' is a machine-readable ConceptSchemeIdentifier and '132' is a machine-readable ConceptIdentifier.

[13] EuroVoc: https://eur-lex.europa.eu/browse/eurovoc.html

Integers are used here for illustration but equally, more complex identifiers could be used. For example, ELSST uses UUIDs for all of its ConceptIdentfiiers: see https://elsst.cessda.eu/id/3/c9167160-43b8-4afb-8291-d6b9887104aa for an example where `c9167160-43b8-4afb-8291-d6b9887104aa` is the persistent ConceptIdentifier. This also has the advantage of being globally unique.

3.2 Summary of recommended syntax

To summarise, this document makes the following recommendations for URIs:

- Avoid the use of natural language in URIs

- A URI for a ConceptScheme should be constructed thus:
  - `<BASE_URI>/<CONCEPTSCHEME_IDENTIFIER>/<CONCEPTSCHEME_VERSION>/`
    - Example 1:
      `https://testrdf-vocabulary.ddialliance.org/cv/AggregationMethod/1.1/`
      Where *`https://testrdf-vocabulary.ddialliance.org/cv/`* is the BaseURI, *AggregationMethod* is the ConceptScheme identifier and *1.1* is the ConceptScheme version.
    - Example 2:
      `https://elsst.cessda.eu/id/3/`
      Note that here *`https://elsst.cessda.eu/id/`* functions as a combined BaseURI & ConceptScheme identifier and 3 is the ConceptScheme version.

- A URI for a Concept should usually be constructed thus:
  - `<BASE_URI>/<CONCEPTSCHEME_IDENTIFIER>/`
    `<CONCEPTSCHEME_VERSION>/<CONCEPT_IDENTIFIER>`
    - Example:
      `http://testrdf-vocabulary.ddialliance.org/cv/AggregationMethod/1.1/d35e61`
      Where *`https://testrdf-vocabulary.ddialliance.org/cv/`* is the BaseURI, *AggregationMethod* is the ConceptScheme identifier, *1.1* is the ConceptScheme version and *d35e61* is the Concept identifier.

3.3 “Version-neutral” URIs

So far, we have focussed on specific versioned instances of both Concepts and ConceptSchemes. However, it may be that as a consuming agent you are not particularly interested in what the label of a concept was at a specific point in time. Rather, you may simply want to know what the most up-to-date label is (and perhaps other metadata) for a particular concept.

A method to achieve this is with “version-neutral” URIs. Let's say that http://example.org/652/1/132 is the concept labelled “DOG” in version 1 of the ‘Animals’ ConceptScheme and http://example.org/652/2/132 is the same concept re-labelled as “HOUND” in version 2 of the same ConceptScheme. Let us also say that I want to always reference the *latest* version of this concept, rather than targeting the concept as it was at a specific point in time. I could provide mechanisms to resolve `http://example.org/652/132`

to the latest version, in this case `http://example.org/652/2/132`. Once the ConceptScheme had been incremented to v3, then `http://example.org/652/132` would instead resolve to `http://example.org/652/3/132` and so on over time.

Note that this kind of redirection requires a relatively sophisticated infrastructure, with linked data endpoints being able to provide the correct resolution over time to the latest version. This is normally achieved in practice with redirection and URL rewrite mechanisms on reverse proxies like Nginx[14] or HAProxy[15]. The CESSDA ELSST Thesaurus[16] provides a rich real-world implementation where all ConceptSchemes and Concepts have both multiple version-specific URIs and a single "version-neutral" URIs (and URNs for expression in XML documents, typically DDI).

3.4 A note on multilinguality

Concept labels very often need to be translated into multiple languages. It is *critical* to understand that the translation of labels and notes into other languages does *not* imply that separate objects with separate URIs are required. For example, a representation along these lines might look like:

`http://example.org/652/2/132_en` and `http://example.org/652/2/132_fr`

which are two URIs that resolve to the (version 2) concept with identifier '132". The English concept label is 'DOG' and the French concept label is 'CHIEN'.

While at one level it might seem intuitive to approach translations this way (indeed, earlier incarnations of CESSDA Vocabulary Manager took this approach), it cannot be emphasised enough that consuming agents need to be completely unambiguous about which concept *object* is being referenced. Segmenting concept objects into multiple language objects causes significant additional complexity for comparability and interoperability[17]. The underlying language characteristics of the concept are, in one sense, secondary semantic issues. Indeed, the design of DDI schema has been predicated on the axiom that languages are secondary attributes of concept objects, and the @xml:lang attribute is used precisely for this purpose. The DDI Alliance *strongly* discourages the instantiation of language-specific objects with distinct URIs. Following on from our example above, the much preferred approach is to represent 'DOG' and 'CHIEN' with a single URI like so:

```
<skos:Concept rdf:about="http://example.org/652/2/132>
      <skos:prefLabel xml:lang="en">DOG</skos:prefLabel>
      <skos:prefLabel xml:lang="fr">CHIEN</skos:prefLabel>
</skos:Concept>
```

[14] Nginx: https://www.nginx.com/
[15] HAProxy: http://www.haproxy.org/
[16] ELSST: https://thesauri.cessda.eu/en/
[17] Note that SKOS-XL allows you to create multiple label and note objects which can contain richer metadata about the label/note. These can be language specific. Nonetheless, these label/note objects will still be associated with a single Concept object, and so the principle of not creating multiple Concept URIs for multiple languages still holds.

4. Provenance

While it's one thing to imply a change by the increment of a version number, provenance information specifies the details of what has been changed and when and ideally by whom. While we have previously emphasised the machine-readability of SKOS vocabulary metadata, this is a scenario where human readable information can be valuable.

Typically, a metadata manager, data steward, ontologist or researcher may reasonably ask of a Concept or ConceptScheme: "show me a concise summary of what has changed since I last looked at this CV".

4.1 Recording change

At the human readable level, there are three key properties that can be used to provide human-readable information about changes to a Concept or ConceptScheme.

1. **skos:historyNote**
   A top-level summary intended for users of the ConceptScheme, documenting significant changes to the meaning, form, or state of its child concepts. History notes can be applied to Concepts as well, but it is the recommendation of this document that they be reserved for use as top-level summary of changes to the CV, analogous to release notes.
2. **skos:changeNote**
   A change note is intended for documenting more granular changes to a concept for the purposes of administration and management.
3. **owl:versionInfo**
   A textual, human-readable representation of the version information for either a ConceptScheme or a Concept. Although the version number may be indicated in the URI, this may not necessarily always be the case.

```
<skos:ConceptScheme rdf:about="http://example.org/652/2/>
      <skos:historyNote rdf:parseType="Resource">
      <rdf:value>This is the human readable change log for version 2
      of the ANIMALS CV. Concept 132 was re-labelled in English from
      'DOG' to 'HOUND'</rdf:value>
      <dc:date>2021-10-08</dc:date>
      <owl:versionInfo>Version 2</owl:versionInfo>
      </skos:historyNote>
</skos:ConceptScheme>
```

In version 2 of the ANIMALS CV above, skos:historyNote is used to provide a summary of the CV changes from version 1. At the concept level, there could be a similar construction using skos:changeNote but this would require slightly more involved techniques like reification.

4.2 Recording relationships between versions

As well as representing summary change logs for the CV with skos:historyNote and more granular concept changes with skos:changeNote, it is important that each versioned instance of either a ConceptScheme or Concept can be correctly related to its preceding and succeeding versions. Commonly used RDF predicates for this purpose include:

- dct:hasVersion (inverse dcterms:isVersionOf)
- owl:priorVersion

These point to resources, rather than containing contextual notes, so following on from our example above, if we were referencing version 3 of the Animals CV, the following snippet illustrates how that might be constructed:

```
<skos:ConceptScheme rdf:about="http://example.org/652/3>
      <owl:priorVersion rdf:resource="http://example.org/652/2"/>
      <dcterms:isVersionOf rdf:resource="http://example.org/652/"/>
</skos:ConceptScheme>
```

Similarly, version 2 of the Animals CV looks like:

```
<skos:ConceptScheme rdf:about="http://example.org/652/2>
      <owl:priorVersion rdf:resource="http://example.org/652/1"/>
      <dcterms:isVersionOf rdf:resource="http://example.org/652/"/>
</skos:ConceptScheme>
```

If we want to maintain a "version-neutral" instance of the 'Animals' ConceptScheme with, for example, a URI of `http://example.org/652/` we may wish to indicate which specific versions of this ConceptScheme are available:

```
<skos:ConceptScheme rdf:about="http://example.org/652/>
      <dcterms:hasVersion rdf:resource="http://example.org/652/3/"/>
      <dcterms:hasVersion rdf:resource="http://example.org/652/2/"/>
      <dcterms:hasVersion rdf:resource="http://example.org/652/1/"/>
</skos:ConceptScheme>
```

4.3 Dealing with deprecated concepts

Generally speaking, it is not recommended to fully *delete* concepts. Once you are working with linked data and URIs, the foundational idea is that URIs should be persistent and should resolve to something, even if the concept is no longer actually used. In order to represent these, **owl:Deprecated** can be used in conjunction with skos:changeNote to provide a "stub" of information i.e. a tombstone record.

```
<skos:Concept rdf:about="http://example.org/652/4/132>
      <skos:inScheme rdf:resource="http://example.org/652/4/"
      <owl:priorVersion rdf:resource="http://example.org/652/3/132"
      <owl:deprecated rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
            true
      </owl:deprecated>
      <skos:changeNote rdf:parseType="Resource">
            <rdf:value>
            The concept labelled 'HOUND' in English (ID 132) was deprecated
            in version 4 of the Animals CV
            </rdf:value>
            <dc:date>2021-12-19</dc:date>
            <owl:versionInfo>Version 4</owl:versionInfo>
      </skos:changeNote>
</skos:Concept>
```

In the example above, the 'Animals' CV is now at version 4 and the concept labelled 'HOUND' (with identifier '132') has now been deprecated. Nonetheless, sufficient information is provided here that will allow a consuming agent to establish this unambiguously and also to traverse to the previous version 3 of the concept (`http://example.org/652/3/132`), if required.

4.4 A note on semantic drift

During the lifecycle of a vocabulary, the specification or meaning of a concept might change and result in a split or merge of concepts. The "new" concept(s) should not reuse a previous ConceptIdentifier and continue incrementing its version sequence. Instead, a new ConceptIdentifier should be used (in the case of a merge) and new ConceptIdentifiers (in the case of a split). Good governance practices should be in place to clearly designate who can make decisions on new Concepts or ConceptSchemes[18].

5. Conclusion

This discussion paper is intended to stimulate debate and feedback on a practical approach to the versioning of Controlled Vocabularies, an approach which should enable consuming agents to reference a particular instance of the Concept within the context of a Concept Scheme published at a particular point in time. Additionally, we considered the boundaries and scope of versioning for CVs, how URIs should optimally be constructed, and the trade-offs involved between perfect information modelling and usability, as well as some simple mechanisms for capturing provenance and change information. We are keen to continue the discussion and welcome feedback.

[18] For more guidance on "content custodians", see Cox et al. (2021). Ten Simple Rules for making a vocabulary FAIR. PLoS Comput Biol 17(6): e1009041. https://doi.org/10.1371/journal.pcbi.1009041