Evolving CLDF: Why and how?

Robert Forkel
Department for Linguistic and Cultural Evolution, MPI EVA, Leipzig

A bit of history

> *“Survey databases are all alike” [Alexis Dimitriadis]*

The CLLD project brought home this message as well.

And after a couple of workshops on “Language Comparison with Linguistic Databases” we decided to try to “externalize” `clld`’s data model into a format specification.

CLDF Design principles

> *“Simple things should be simple, complex things should be possible.” [Alan Kay]*

- MVP (minimum viable product): must “fit” WALS and WOLD
- Prioritize data re-use over data authoring
- Work well with version control

What is CLDF?

- A package format based on CSVW
- bundling cross-linguistic tabular data
- with schema information and metadata

Basically, a database serialization format

https://github.com/cldf/cldf/blob/master/README.md

What role does CLDF play?

CLDF decouples data curation and development of tools and methods while maintaining interoperability.

> *"**Interoperability** is a characteristic of a product or system to work with other products or systems."*

What role does CLDF play?

CLDF creates more research opportunities, because

- not only the maintainers of dataset X can work on methods to analyze it and
- “methods people” do not need to code “their own data”.

Is CLDF interoperable?

Interoperability is dependent on “inter-operation” actually taking place! Hence this workshop :)

To maximise the potential for inter-operation, standards need to be evolving.

The remainder of the talk will be about the mechanics of evolving/extending CLDF.

Extensibility

> *“Extensibility is a software engineering and systems design principle that provides for future growth.”*

- CLDF is extensible because the CSVW spec allows for arbitrary tables and columns.
- CLDF is extensible in the way HTML is: Web browsers will just ignore unknown, new tags – until they implement support for them.

What is the process for extending CLDF?

- People include non-standard data in CLDF datasets, using ad-hoc data models.
- Other people pick up these ad-hoc models as examples for their own datasets.
- Analysis code uses the ad-hoc models to access/understand data.

This puts the ad-hoc model on the “standardization track”.

CLDF Terminology

CSVW	marker	CLDF
table	dc:conformsTo	Component
column	propertyUrl	Ontology terms

How is the workflow implemented?

Standardization of new features is discussed in issues at https://github.com/cldf/cldf

Things that can be added to CLDF are:

- modules
- components
- CLDF ontology terms
- i.e. new tables and columns with fixed, linguistic meanings

An example

- CLLD apps often serve data curated as “edited volume”. E.g. WALS bundles and edits contributions by many authors.
- Both, the WALS and WOLD CLDF datasets already used an ad-hoc table to include this kind of data.
- The “analysis” software making use of this table was the `clld` framework.

CLDF 1.1: `ContributionTable`

To make the relation between datasets and their constituents transparent, CLDF added

- a `ContributionTable` *component* and
- a couple of *ontology terms*, e.g.
  - a bibliographic citation
  - a reference property to refer to contributions from other tables

Extensibility creates complexity

- PHOIBLE lists contributions in `inventories.csv` but has no `ContributionTable` - because its latest release predates CLDF 1.1
- Semantic versioning for datasets breaks down: Is the PHOIBLE 2.0 data with a proper `ContributionTable` v2.0.1 - even though it may break compatibility with some tools?

Extensibility creates complexity

“All problems in computer science can be solved by another level of indirection.” [David Wheeler]

Extensibility creates complexity

- Thanks to CLDF providing a level of indirection between table and column names and their meaning, the `ContributionTable` in PHOIBLE 2.0.1 could still be `inventories.csv` (and columns referencing it could keep their names, too)
- CSVW - and thus CLDF - keep metadata and data separate, thus metadata can be added to data without touching the data.

CLDF 1.1: `MediaTable`

- Driven by use cases: APiCS glossed texts, audio for wordlists.
- Added component `MediaTable` with properties
  - `mediatype`: iana Media Types
  - `downloadUrl`: a URL - possibly prefixed with the `data:` or `file:` scheme.

Again some complexity

- Caveat: Media file content will not be available via CLDF SQL (unless content is stored via `data:` URLs).
- So there's a tension between putting content that SQLite could handle well - like GeoJSON - in tables or in media files.

CLDF 1.2: `TreeTable`

- Driven by use cases: Glottolog, Phlorest, `lingtreemap`
- Added component `TreeTable` with properties
  - `treeIsRooted`
  - `treeType`
  - `treeBranchLengthUnit`

Tree representation

`TreeTable` is special in that

- it just lists metadata about trees and
- links to the Newick representation of trees via `mediaReference` to a file in Newick or Nexus format.
- Should we have added a table listing tree edges?

Judgement calls: Useful properties

Which properties are “actionable” - thus worth standardizing?

The main criterion is clear semantics - ideally defined by software that can act upon them e.g.

- `mediaType` in `MediaTable`
- `treeIsRooted` and `treeBranchLengthUnit` in `TreeTable`

Judgement calls: Concreteness vs. Genericity

- Being too concrete may exclude useful applications in the future.
- Being too generic may exclude useful analysis methods.

Generic:

E.g. one could model “speaker area” information with special `mediaReference` from `LanguageTable`. This would allow linking to images of maps as well as to GIS data like GeoJSON.

Even if the linked file can be identified as GeoJSON via `mediaType`, it’s still unclear which feature actually represents the speaker area.

Concrete:

“Speaker area” could also be a property of `datatype` JSON, which already contains a polygon encoded in GeoJSON.

See https://github.com/cldf/cldf/issues/145

Judgement calls: Backwards incompatible changes

There’s currently no mechanism/process for backwards-incompatible changes of the spec.

The least complicated one would probably be some sort of deprecation process as seen e.g. in Python to prevent bloat.

Conclusion

Join us at https://github.com/cldf/cldf/labels/data modeling