DCPAPERS

DCMI 2026

Proceedings of the International Conference on Dublin Core and Metadata Applications 2026

ISSN: 1939-1366

Tapir: A Graphical Editor for Tabular Application Profiles

Nishad Thalhath [1], **Mitsuharu Nagamori** [2], and **Tetsuo Sakaguchi** [2]

[1] RIKEN Center for Integrative Medical Sciences, Yokohama, Japan, [2] Faculty of Library, Information and Media Studies, University of Tsukuba, Tsukuba, Japan

Abstract

Application profiles record the local application of metadata vocabulary terms, how they combine, and the values they may take. Two tabular formats are in active use for authoring them. SimpleDSP, from the Metadata Information Infrastructure Construction Project in Japan, has served for over a decade as a tabular form of the Description Set Profile model. DCTAP, from a more recent DCMI working group, covers the same ground without the DSP inheritance. Neither has displaced the other, but practitioners moving between communities have had no single authoring interface that treats the two as peers. This paper compares the two formats along the dimensions that shape application profile authoring in practice, and argues that neither is a reduction of the other. Based on this analysis, the authors present Tapir, a browser-based graphical editor whose internal model is a superset of both formats. Tapir preserves the bilingual character of SimpleDSP and runs entirely in the user's browser, so profile data never leaves the author's machine. Alongside the editor, an updated command-line toolkit and a type-safe authoring route in Apple's Pkl language operate on the same profile model. The editor, the toolkit, and the Pkl package are released as open-source software.

1. Introduction

Metadata application profiles combine terms from one or more vocabularies and customise them for a particular local application [1]. The Singapore Framework [2] treats the profile as the primary documentary artefact of a metadata design, in both human-readable and machine-actionable forms. Profiles are expressed in XML, RDF, JSON-LD, OWL, SHACL [3], and ShEx [4] none of these is a comfortable authoring surface for domain experts. Tabular profiles have been the pragmatic answer: a profile that lives in a spreadsheet, one row per property constraint.

Practice has settled on two such formats. *SimpleDSP* (簡易 DSP in the Japanese original, written "Simple-DSP" or "SDSP" in earlier English-language accounts, including our own; this paper uses the form adopted by current tooling and documentation) was specified around 2011 in the *Guidelines for Metadata Information Sharing* [5] under Japan's Metadata Information Infrastructure Construction Project, alongside the MetaBridge registry [6]. SimpleDSP is a tabular serialisation of Dublin Core's Description Set Profile (DSP) [7]. *DCTAP* [8], developed more recently by a DCMI working group, is not tied to the DSP lineage; it is a flat CSV vocabulary of twelve elements, of which only `propertyID` is required. Neither format has displaced the other, and published work on tabular profiles has described each on its own, or compared one with non-tabular formats [9], without a direct account of the two side by side. This paper provides that account and a tool that treats them as peers.

Full Paper

Available Aug 01, 2026

Correspondence to
Nishad Thalhath
nishad.thalhath@riken.jp

The tool, *Tapir*, is a browser-based editor built over a shared internal profile model, YAMA [10], [11]. YAMA is a YAML-based preprocessor that generates standard profile formats and accepts custom extension elements [9] later work added provenance [12] and RDF data mapping [13]. Tapir and the YAMA command-line toolkit share this internal model, so that SimpleDSP and DCTAP can be produced from it and parsed back into it.

1.1. Contributions

1. A direct comparison of SimpleDSP and DCTAP, by dimension, covering column model, cardinality, value constraints, and bilingualism.
2. Tapir, a browser-based graphical editor filling a gap the current tabular-profile ecosystem leaves open: it edits both formats over one internal model and preserves SimpleDSP's bilingual (Japanese/English) authoring surface.
3. An updated YAMA CLI whose converters are kept output-identical with Tapir's, and *yama-pkl*, a Pkl [14] package for type-safe profile authoring.

2. Background

The Dublin Core account of application profiles developed as a chain: Heery and Patel's original definition [1], the Dublin Core Abstract Model (DCAM) [15] with its Description Set, Description, and Statement vocabulary, and the Singapore Framework [2] that positioned the profile as a documentary artefact. Description Set Profiles (DSP) [7] is the constraint language that sits on DCAM. OWL-DSP [16] encodes DSP as an OWL ontology (`dsp:DescriptionTemplate`, `dsp:StatementTemplate`, properties for cardinality notes, scheme membership, and language-tag constraints). OWL-DSP and SimpleDSP appear together in Chapter 6 of the *Guidelines for Metadata Information Sharing* [5] each SimpleDSP row maps deterministically to an OWL-DSP construct.

DCTAP emerged later, from a DCMI Application Profiles Working Group whose remit was a tabular format easier to adopt than DSP-based serialisations [8]. It is a primer-plus-elements specification with an extension cookbook, and is supported by `dctap-python`[1], `TAP2SHACL`[2], and `rudof`[3] for conversion to SHACL and ShEx. The two lines continue in parallel.

3. SimpleDSP and DCTAP compared

Both formats target the same authoring audience but arrive from different starting points. SimpleDSP serialises DSP; its column model is shaped by the DSP constructs it encodes. The differences that follow are consequences of those lineages. The full specifications for SimpleDSP, OWL-DSP, and DCTAP live at their canonical locations.[4]

[1] https://github.com/dcmi/dctap-python

[2] https://github.com/philbarker/TAP2SHACL

[3] https://github.com/rudof-project/rudof

[4] SimpleDSP: https://docs.yamaml.org/specs/simpledsp/ (English rendering, verbatim Japanese original, and worked examples). OWL-DSP: https://docs.yamaml.org/specs/owl-dsp/. DCTAP: https://www.dublincore.org/specifications/dctap/.

3.1. Structural model

A SimpleDSP file is a UTF-8 TSV partitioned into bracketed blocks: an optional `[@NS]` namespace declaration and one or more Description Template blocks starting with `[MAIN]`. Inside each block, every non-comment row is a Statement Template carried by *seven positional columns* (name, property, min, max, value type, constraint, comment). Rows beginning with `#` are comments the parser ignores, including the conventional header row. Column meaning is determined by position. Structured values reference other blocks by `#blockId`, and a block may reference itself, the idiom for nested records.

A DCTAP file is a single CSV table. The first row is an authoritative header of named columns in arbitrary order, and shape membership is carried in a `shapeID` column value rather than by block delimiters: rows sharing a `shapeID` belong to the same shape. DCTAP defines a core vocabulary of twelve elements, of which only `propertyID` is required; `shapeID` itself is optional, so a minimally valid DCTAP file can be a single-column CSV of property IRIs. Structured values reference another shape through a `valueShape` column. DCTAP positions itself as extensible: adopters may add columns and constraint types, and parsers are expected to pass unknown values through rather than reject the file. One thing DCTAP does not carry in its core specification is an in-file namespace declaration: the DCTAP primer recommends that prefix-to-IRI mappings be communicated as an external companion table. SimpleDSP's `[@NS]` block, by contrast, keeps prefixes with the profile.

The structural consequences are distinct. A SimpleDSP parser tokenises the file into blocks and interprets each row by position; a DCTAP parser reads the CSV as a single table and groups rows by `shapeID`. Recursive structures are expressed the same way in principle (a reference back to the current shape), but SimpleDSP's block partitioning makes the target typographically visible in a way DCTAP's single table does not.

3.2. Cardinality

SimpleDSP expresses cardinality with two integer columns: Min takes a non-negative integer, Max a non-negative integer or the unbounded marker `-`. The Min column also accepts Japanese keywords such as 推奨 ("recommended") or あれば必須 ("mandatory if present"), which are computationally treated as optional but preserved in OWL-DSP output as `dsp:cardinalityNote` values. DCTAP reduces cardinality to two boolean columns (`mandatory`, `repeatable`), covering the four common cases and nothing else.

Converting SimpleDSP to DCTAP is therefore lossy: numeric bounds beyond the common cases collapse to the boolean representation, and cardinality notes have nowhere to go. Conversion in the other direction is lossless on this dimension, because the boolean model is a proper subset of the integer model; where DCTAP-to-SimpleDSP conversion loses information is in value constraints.

3.3. Value constraints

SimpleDSP packs the constraint and its type into a single column: the Constraint column is interpreted according to the preceding ValueType, holding a datatype list for literal

```
[@NS]
schema  http://schema.org/
@base   http://purl.org/yama/examples/2022/tbbt/0.1/

[MAIN]
#Name      Property          Min Max ValueType Constraint
Name       foaf:name         1   1   literal   xsd:string
Job Title  schema:jobTitle   0   1   literal   xsd:string
Parents    schema:parent     0   -   IRI
```

Listing 1. *The same three-statement shape in SimpleDSP, generated from one source profile by the YAMA toolkit, shown in its native TSV file form with block headers. The equivalent DCTAP rendering is in Table 1.*

values, a block reference or class for structured values, or a vocabulary prefix or list for IRI values. SimpleDSP has no first-class way to express a regular expression, a string length bound, or a language tag.

DCTAP separates the constraint from its type using `valueConstraint` and `valueConstraintType`, giving it first-class constructs for the cases SimpleDSP cannot carry: `pattern`, `languageTag`, `minLength`, `maxLength`, `minInclusive`, and `maxInclusive`. Converting DCTAP to SimpleDSP loses these where a profile uses them: the second direction of lossiness.

A subtler difference appears in multi-vocabulary constraints. SimpleDSP's `ndlsh: bsh:` in a single Constraint cell expresses values drawn from either NDL Subject Headings or Basic Subject Headings (the constraint appears in context in Appendix A). The idiomatic DCTAP equivalent is two rows with the same `propertyID` and different `IRIstem` values, or a local constraint-type extension that accepts multiple stems. A DCTAP author rewriting such a profile faces a multi-row expansion or a local extension; neither preserves the single-cell idiom.

3.4. Bilingualism

SimpleDSP is bilingual by origin. The authoritative specification is Japanese; column meaning is carried by position, so a profile can use either Japanese or English header rows as comments; and four of the five value-type keywords are Japanese: 文字列 ("string"), 構造化 ("structured"), 参照値 ("reference value"), and 制約なし ("no constraint"), the fifth, `ID`, staying in Latin script even in the Japanese text. The English keywords accepted by the YAMA toolkit are later conventions, not part of the published specification. DCTAP is English at the specification level: column identifiers stay in English, and only content-bearing columns (`shapeLabel`, `propertyLabel`, `note`) localise. A practitioner using SimpleDSP can author in one language end to end; a DCTAP author working in another language mixes the two, English for structure and the local language for content.

3.5. Worked example

Listing 1 and Table 1 show the same three-statement shape in both formats, generated from one YAMA source profile by the YAMA toolkit.

Table 1. *The same three-statement shape in DCTAP, rendered as the column table its CSV header defines. The equivalent SimpleDSP form is in Listing 1.*

shapeID	propertyID	propertyLabel	mandatory	repeatable	valueNodeType	valueDataType
	foaf:name	Name	TRUE	FALSE	literal	xsd:string
character	schema:jobTitle	Job Title	FALSE	FALSE	literal	xsd:string
	schema:parent	Parents	FALSE	TRUE	IRI	

Neither format is a superset of the other. SimpleDSP carries information DCTAP cannot (exact numeric cardinality, keyword cardinality notes, a single-cell multi-vocabulary constraint, a bilingual authoring surface). DCTAP carries constraint kinds SimpleDSP lacks in its core (patterns, language tags, length, numeric ranges). A tool treating both as first-class targets needs an internal model that is a superset of both.

4. Tapir

Tabular application profiles exist because the communities who understand the records best (library cataloguers, archivists, curators of research data, domain scientists) do not write SHACL or ShEx. They write spreadsheets. A generic spreadsheet editor cannot tell the author that a prefix is undeclared, that a block reference points to no block, or that a cardinality pattern is malformed; errors surface only downstream. The existing ecosystem for the two formats is uneven on this point: MetaBridge offered SimpleDSP authoring in its registry webapp (unreachable at the time of writing), while the DCTAP side has been served by command-line and library tools (`dctap-python`, `TAP2SHACL`, `rudof`) that expect the profile to be authored elsewhere, joined recently by a single-format browser editor (Section 7). And many of the communities that produce profiles cannot upload them to hosted tooling: galleries, libraries, archives, museums, and research groups working with sensitive data operate under institutional policies that exclude SaaS editors before the question of features arises. Our earlier work on findable datasets for small communities made the same argument on the publishing side [17] the editor presented here applies it to the authoring side.

Tapir[5] is a browser-based editor that treats SimpleDSP and DCTAP as sibling targets over a shared internal model. It runs entirely in the browser: the application ships as a static bundle of HTML, CSS, and JavaScript built with SvelteKit 2 and Svelte 5, no server component is required, and project data lives in the browser's IndexedDB[6] storage via the `idb` wrapper[7]. A reference deployment is hosted at https://yamaml.github.io/tapir/, served as a static site from GitHub Pages; because no profile data leaves the browser, the host only serves code, and an institution that cannot depend on an external operator can serve the same bundle from its own web server. The editor is an installable progressive web application: on first visit, a service worker precaches the application shell and the bundled vocabulary set, and the browser (Chrome, Edge, or any other PWA-capable engine) offers to install Tapir as a standalone app from its address bar.

[5]Source: https://github.com/yamaml/tapir.

[6]https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[7]https://github.com/jakearchibald/idb

After installation, the application launches from the operating system's app launcher or dock and runs entirely offline; editing, diagramming, validation, and export all remain available with no network connection.

4.1. Flavour-neutral internal model

The editor's central architectural decision is its internal model. Tapir's internal representation is neither SimpleDSP nor DCTAP: it is a superset of both. A project holds descriptions, each description holds statements, and each statement carries the union of fields the two formats require. Cardinality is stored as two nullable integers and a free-text cardinality note, which together cover SimpleDSP's integer bounds, its keyword cardinalities, and DCTAP's two booleans. Value constraints are stored in their constituent parts (datatype, picklist values, pattern, length and numeric facets, language tag, named-vocabulary stems, structured-value shape reference, class constraints), so either format's conventions can be reconstructed at export time. The flavour chosen at project creation selects editable fields, interface labels, and default export path, but does not change the data; a profile exports as the other flavour at any time, subject to the lossiness described in Section 3. For SimpleDSP projects, the flavour also carries a language state (Japanese or English) that re-skins the interface without affecting the data. Round-tripping between flavours, and parity between the editor's converters and the command-line toolkit's, both follow from this separation of data and surface.

4.2. Converter layer

The converter layer is a set of pure TypeScript functions with no dependency on the DOM or the persistence layer. The functions are ports of the command-line toolkit's converter modules (Section 5), and tests on both sides hold the two implementations to identical output: a profile round-tripped between the tools is byte-identical. Three formats are bidirectional, parsed as well as generated: YAMA's YAML form, SimpleDSP, and DCTAP. Six more are generate-only: SHACL (via `N3.js`[8]), ShEx compact syntax, OWL-DSP, Frictionless Data Package, JSON, and HTML or Markdown documentation. A diagram pipeline produces DOT source for the profile's shape graph, which the editor lays out with `Elk.js`[9] and renders as SVG; a package generator composes these artefacts into a multi-file ZIP via `fflate`[10], assembled in memory without passing through any server. A profile authored once in Tapir reaches every one of these formats through the same set of modules, replacing what would otherwise be a chain of format-specific tools. For property autocompletion, the vocabulary subsystem bundles a compact core (RDF, RDFS, OWL, XSD, Dublin Core elements and terms, FOAF, SKOS, DCAT, PROV, SHACL, ShEx, and related) and loads additional vocabularies on demand from pre-built JSON chunks; there is no SPARQL endpoint.

[8] https://github.com/rdfjs/N3.js

[9] https://github.com/kieler/elkjs

[10] https://github.com/101arrowz/fflate

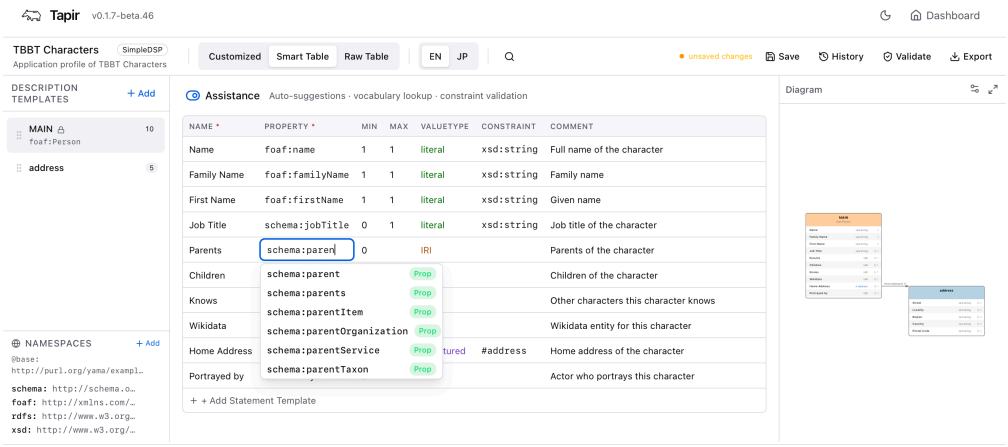

Figure 1. *Tapir's Smart Table mode on a SimpleDSP profile (TBBT Characters fixture). The centre panel shows the seven SimpleDSP columns as an editable table, with the property-autocomplete dropdown open on the* Parents *row offering* `schema:` *terms from the loaded vocabulary. The toolbar above the table carries the three-way mode switch (Customized, Smart Table, Raw Table), the SimpleDSP language toggle (EN / JP), and actions for save, history, validate, and export; the unsaved-changes indicator shows the autosave state. The shape sidebar (left) lists the profile's Description Templates and namespace declarations; the live shape-graph diagram appears in the right panel.*

4.3. *Editor surface*

Tapir offers three views over the same project (Figure 1). *Customized mode* presents each statement as a card with labelled form fields, for users who do not wish to see the tabular format at all. *Smart Table mode* is an editable table with inline validation, colour-coded value types, and clickable shape references; an assistance toggle switches autocompletion on or off. *Raw Table mode* mirrors the native file format: seven positional columns with bracketed block headers for SimpleDSP, or the twelve named DCTAP columns. Edits in any mode are visible in the others without re-parsing.

In a SimpleDSP project, a language toggle in the toolbar switches the entire interface between English (OWL-DSP nomenclature with value-type keywords `literal`, `IRI`, `structured`) and Japanese (the original MetaBridge terminology, with column headers 項目規則名, プロパティ, 最小, 最大, 値タイプ, 値制約, コメント, and value-type keywords 文字列, 参照値, 構造化, 制約なし). When the interface is in one language, no text from the other appears anywhere. On export, the chosen language controls the SimpleDSP output so that Japanese-mode files match the original MetaBridge specification. The toggle is more than convenience. SimpleDSP's specification is Japanese, and its column meaning is carried by position rather than by header text; an editor labelled only in English would erase exactly that property.

A validation rule catalogue, mirrored in the command-line toolkit, covers four categories of checks. Referential checks resolve prefixes against the active namespace map and verify that shape references in `valueShape` columns name a description that exists in the profile. Structural checks enforce cardinality coherence (min ≤ max, both non-negative), require SimpleDSP profiles to open with a `[MAIN]` block, and forbid duplicate description names. Value-type checks cross-validate `valueNodeType`, `valueDataType`, and shape references; a statement marked `literal` that carries a shape reference,

or an IRI-typed value with a datatype, is flagged. Vocabulary-aware checks read the `rdfs:domain` and `rdfs:range` annotations in the bundled vocabulary chunks: a property declared on a description whose target class is incompatible with the property's domain, or assigned a `valueDataType` that contradicts its range, produces a warning. The vocabulary-aware checks fire only when the property is found in a bundled chunk and the chunk carries the relevant annotations; profiles using custom or unbundled vocabularies see no new warnings, keeping the editor permissive toward unfamiliar terms. Fields are checked as the user leaves them (a red accent and a short message), and a toolbar action opens a bottom panel with the full structured report. Messages name the offending value rather than the rule that fired:

```
Unknown prefix "xyz" in property "xyz:foo"
```

and where the remedy is mechanical the editor offers it as an action: a banner above the table declares the missing prefixes in a single step.

The editor saves to the browser's IndexedDB storage on every edit (800 ms debounce), keeps an automatic and manual snapshot history with semantic diffs, and renders a live Elk.js diagram of the profile's shape graph alongside the editor. Import accepts the three authoring formats with auto-detection on ambiguous CSVs; export offers sixteen formats across six categories (tabular, constraint languages, other, diagrams, reports, package).

5. Ecosystem: YAMA CLI and Pkl authoring

The YAMA CLI[11], which grew out of the tooling published with YAMA [10] and YAMAML [13] and has been substantially extended since, is a Deno application informed by the Command Line Interface Guidelines[12], with nineteen commands in four roles: scaffolding and validation (`init`, `validate`); export from the YAMA canonical form to SimpleDSP, DCTAP, SHACL, ShEx, OWL-DSP, JSON, Frictionless Data Package, RDF instance data and vocabularies, diagrams, HTML or Markdown documentation, or a full publication package; import from each tabular format, SHACL, or ShEx into the canonical form; and a rendering utility.

The `validate` command runs the same rule catalogue as the editor (Section 4.3) with human-readable or JSON output, and returns exit code `0` for a valid profile or `1` for one with errors; continuous integration is the motivating scenario. `package` produces a directory containing the profile in every supported format, diagrams, HTML and Markdown documentation, and a `README.md` listing each file with a pointer to its canonical specification (Appendix B); the package is self-documenting and can be consumed without the YAMA toolkit installed.

[11] https://github.com/yamaml/yama-cli

[12] https://clig.dev

yama-pkl[13] is a Pkl [14] package that defines a type-safe schema for authoring YAMA profiles. A profile amends the schema package, declares namespaces and a base URI, and populates `descriptions` with typed `Description` and `LiteralStatement` objects; `pkl eval -f yaml` produces a YAMAML file that the CLI consumes (Appendix C). Pkl's `amends` and `import` mechanisms give first-class composition, a pattern neither SimpleDSP nor DCTAP offers in the tabular layer. Whether this route attracts adoption beyond users already comfortable with Pkl is open; the audience that values static types is not the audience the tabular formats were designed for.

6. Validation

The editor's converter layer (Section 4.2) is exercised by 443 converter tests across its fourteen parser and generator modules, with a further 364 tests covering file I/O, validation rules, IRI utilities, vocabulary loading, IndexedDB persistence, and internal types (807 tests in total).[14] The suite is not a correctness proof; it demonstrates that round-trip between the canonical YAMA form and each supported authoring format preserves the fields defined in Section 4.1, and that the generators produce syntactically well-formed output.

For an end-to-end round-trip, we used a profile of three Description Templates and seventeen Statement Templates, covering literals with datatypes, repeatable IRI references, and cross-block structured references. The profile is 233 lines of YAML, 31 lines across four blocks of SimpleDSP TSV, and 20 lines of DCTAP CSV. Converting the canonical YAMA form to SimpleDSP, back to YAMA, and again to SimpleDSP yielded byte-identical SimpleDSP files. The same round-trip through DCTAP yielded byte-identical DCTAP files. Cross-flavour round-trip (SimpleDSP to DCTAP and back) is lossy in the two directions identified in Sections 3.2 and 3.3: within a single flavour, it is lossless on this fixture.

Beyond the round-trip check, the constraint-language outputs are verified against external validators. The generated SHACL files parse cleanly under pySHACL[15] and the generated ShEx files under `shex.js`[16] the equivalence between SimpleDSP and DCTAP is also checked at the RDF level by serialising each side to its constraint-language form and comparing the resulting shapes. Together, these checks give some confidence that the generated artefacts are semantically consistent with the underlying profile model, not merely syntactically well-formed. Profile-level validation, covering the referential, structural, value-type, and vocabulary-aware coherence checks shown in the editor's validation panel, is described in Section 4.3.

[13] https://github.com/yamaml/yama-pkl

[14] https://github.com/yamaml/tapir/tree/main/tests

[15] https://github.com/RDFLib/pySHACL

[16] https://github.com/shexjs/shex.js

7. Related work, limitations, and future work

The editor extends prior work by the present authors on the YAMA toolkit [10], [11] and the YAMAML RDF mapping language [13] onto the web. The 2019 extensibility comparison [9] evaluated SimpleDSP against the BIBFRAME profile editor and a YAML-based alternative; this paper addresses a different axis, the comparison of the two tabular formats that have become established since. On the publishing side, our earlier work on findable datasets for small communities [17] made the privacy and sovereignty argument for static publishing. The closest tool in purpose is the MetaBridge registry webapp, which implemented SimpleDSP authoring but not DCTAP and was unreachable at the time of writing; the closest in surface is a plain spreadsheet, which cannot check prefixes, references, or cardinality. The BIBFRAME Profile Editor [18] and Sinopia [19] are profile editors for the BIBFRAME vocabulary; they are not flavour-neutral and require a hosted backend. The DCTAP command-line tools (Section 2) operate on profiles someone else has authored. DCTap-Dancer [20][17], a spreadsheet-style browser editor for DCTAP released by the Library of Congress while Tapir was in development, addresses the same gap for that single format; it has no SimpleDSP side and no flavour-neutral model. LinkML [21][18] shares the goal of making schemas authorable and transformable to formats including SHACL, ShEx, JSON Schema, and OWL, and has seen adoption across biomedicine and adjacent sciences. Its scope is broader, covering general data modelling rather than the application-profile constructs that shape SimpleDSP and DCTAP, and its reference tooling is a YAML-first command-line toolkit (`linkml`) rather than a graphical editor; Tapir complements this line by providing a browser-based visual editor dedicated to the tabular application-profile formats.

7.1. Limitations

Profile-level validation covers referential, structural, value-type, and vocabulary-aware coherence (Section 4.3); it is not a model-theoretic check on the profile as a whole. The vocabulary-aware tier reads the `rdfs:domain` and `rdfs:range` annotations carried in the bundled vocabulary chunks, and is therefore limited to properties drawn from the bundled set. Three further classes of semantic check remain unimplemented: cross-vocabulary class equivalence (which would need shared `owl:equivalentClass` data the chunks do not carry), `rdfs:subPropertyOf` chain reasoning, and reasoner-backed validation of generated SHACL or ShEx shapes against instance data. The first two are local additions blocked only by data preparation. The third would require either bundling a SHACL or ShEx engine in the browser or calling an external service; we have done neither. Import from SHACL and ShEx is best-effort: target classes, boolean cardinality, datatypes, and value-shape references round-trip, but SHACL property paths longer than one step, shape disjunctions, and ShEx triple expressions outside the DSP-style subset are imported as notes. Tapir also deliberately does not cover two adjacent tasks:

[17] Source: https://github.com/lcnetdev/dctap-dancer.

[18] https://linkml.io

RDF instance-data generation (handled by the YAMAML subset [13] and the CLI) and collaborative editing.

7.2. Future work

The semantic-validation limitations noted in Section 7.1 come first. Cross-vocabulary class equivalence and `rdfs:subPropertyOf` reasoning are incremental extensions of the bundled-vocabulary pipeline, and we plan to work on them next. Reasoner-backed validation of generated shapes against instance data is a larger intervention, changing what validation means in the editor from profile coherence to instance conformance. Recent work on in-browser reasoning over the same underlying RDF library [22] suggests this is tractable; we have not prototyped it.

Large Language Models (LLMs) are plausibly useful in two parts of the authoring task: property discovery (suggesting candidate properties from a relevant vocabulary) and profile drafting (producing a skeleton from a sample record). Tapir's flavour-neutral internal model is a reasonable output format for such assistance, and the validation rules flag specific problems in an LLM-generated draft for the author to fix before they reach downstream tools. We have not integrated LLM features into the shipped editor. There is no settled convention for which vocabularies to suggest from, and no agreed protocol for a privacy-preserving model call from a client-side application; nobody has yet published a baseline for what a good LLM-suggested profile looks like.

8. Conclusion

SimpleDSP and DCTAP share an authoring audience and a spreadsheet surface but derive from different lineages: the Dublin Core Description Set Profile model on one side, a recent DCMI working-group design on the other. The two persist side by side, and an author moving between communities has had to change tools at the format boundary. Tapir is a practical response. Its internal model is a superset of what either format carries, and its features follow from that choice. Profile data stays in the author's browser, so the institutional policies that exclude hosted editors do not exclude Tapir. The tool does not presume to settle which format should prevail.

Appendix A. Worked example: the NDL bibliographic profile in SimpleDSP

The canonical SimpleDSP example from the original specification [5] describes a bibliographic record for the National Diet Library. The `[MAIN]` block describes the book itself; a second block describes its title as a nested structured value with a literal form and a reading. This appendix shows the profile first in an English rendering and then in the Japanese original form.

A.1. English version

The English version below is adapted from the original Japanese specification: block names, field names, and comment text have been rendered into English, while the profile's structure, vocabulary prefixes, and value types are unchanged.

```
[@NS]
dcndl    http://ndl.go.jp/dcndl/terms/
ndlsh    http://id.ndl.go.jp/auth/ndlsh/
bsh      http://id.ndl.go.jp/auth/bsh/
ndlbooks http://iss.ndl.go.jp/books/
@base    http://ndl.go.jp/dcndl/dsp/biblio

[MAIN]
#Name       Property            Min Max ValueType  Constraint
Comment
BibID       foaf:Document       1   1   ID         ndlbooks:
Record identifier
Title       dcterms:title       1   1   structured #StructuredTitle  Title
of the document
Creator     dcterms:creator     0   1   structured foaf:Agent
Author of the document
IssuedDate  dcterms:issued      1   1   literal    xsd:date
Publication date
Subject     dcterms:subject     0   -   IRI        ndlsh: bsh:
Subject of the document

[StructuredTitle]
#Name         Property             Min Max ValueType Constraint Comment
LiteralForm   xl:literalForm       1   1   literal             The title
string itself
Transcription dcndl:transcription  0   1   literal             Reading of
the title
```

A.2. Japanese version

The canonical form as it appears in the original Japanese specification.

```
[@NS]
dcndl    http://ndl.go.jp/dcndl/terms/
ndlsh    http://id.ndl.go.jp/auth/ndlsh/
bsh      http://id.ndl.go.jp/auth/bsh/
ndlbooks http://iss.ndl.go.jp/books/
@base    http://ndl.go.jp/dcndl/dsp/biblio

[MAIN]
#項目規則名   プロパティ          最小 最大 値タイプ 値制約          説明
書誌ID        foaf:Document      1    1    ID       ndlbooks:      文書のID
タイトル      dcterms:title      1    1    構造化   #構造化タイトル 文書の表題
著者          dcterms:creator    0    1    構造化   foaf:Agent     文書の作者
発行日        dcterms:issued     1    1    文字列   xsd:date       文書の発行日
主題          dcterms:subject    0    -    参照値   ndlsh: bsh:    文書の主題

[構造化タイトル]
#項目規則名   プロパティ                最小 最大 値タイプ 値制約 説明
リテラル値    xl:literalForm           1    1    文字列          タイトル自身
読み          dcndl:transcription      0    1    文字列          タイトルの読み
```

Appendix B. The yama package output directory

The directory produced by `yama package`. The same output is available from Tapir's export dialog.

```
my-profile/
|-- index.html              HTML documentation (with diagram)
|-- profile.md              Markdown documentation
|-- README.md               Format index with spec links
|-- diagram.svg             Overview diagram
|-- diagram-detail.svg      Detailed diagram
|-- diagram.pdf             Overview diagram as vector PDF
|-- profile.yaml            YAMA source
```

```
|-- profile.json            JSON representation
|-- simpledsp.tsv           SimpleDSP (English)
|-- simpledsp-jp.tsv        SimpleDSP (Japanese)
|-- dctap.csv               DCTAP
|-- shacl.ttl               SHACL shapes
|-- shex.shex               ShEx
|-- owl-dsp.ttl             OWL-DSP
\-- datapackage.json        Frictionless Data Package
```

Appendix C. A minimal profile in yama-pkl

A minimal YAMA profile authored in Pkl. The `amends` declaration imports yama-pkl's schema package; the typed classes (`Description`, `LiteralStatement`) give editor auto-completion and compile-time validation.

```
amends "package://pkg.pkl-lang.org/github.com/yamaml/yama-pkl/yama@0.2.0#/
Profile.pkl"

base = "http://example.org/people/"

namespaces {
  ["foaf"] = "http://xmlns.com/foaf/0.1/"
  ["xsd"]  = "http://www.w3.org/2001/XMLSchema#"
}

descriptions {
  ["person"] = new Description {
    a = "foaf:Person"
    statements {
      ["name"] = new LiteralStatement {
        property = "foaf:name"
        min = 1
        max = 1
        datatype = "xsd:string"
      }
    }
  }
}
```

Appendix D. Declaration of Generative AI and AI-assisted Technologies in the Writing Process

In preparing this work, the authors used Grammarly and the OpenAI and Anthropic APIs to improve readability, correct grammar, refine language, and assist with LaTeX typesetting. After using these services, the authors reviewed and edited the content as necessary and take full responsibility for the manuscript.

References

[1] R. Heery and M. Patel, "Application Profiles: Mixing and Matching Metadata Schemas," *Ariadne*, no. 25, 2000, [Online]. Available: https://www.ariadne.ac.uk/issue/25/app-profiles/

[2] M. Nilsson, T. Baker, and P. Johnston, "The Singapore Framework for Dublin Core Application Profiles," 2008. [Online]. Available: https://dublincore.org/specifications/dublin-core/singapore-framework/

[3] H. Knublauch and D. Kontokostas, "Shapes Constraint Language (SHACL)," 2017. [Online]. Available: https://www.w3.org/TR/shacl/

[4] E. Prud'hommeaux, J. E. Labra Gayo, and H. Solbrig, "Shape Expressions (ShEx) 2.1 Primer," 2019. [Online]. Available: https://shex.io/shex-primer/

[5] Ministry of Internal Affairs and Communications of Japan, "Guidelines for Metadata Information Sharing, Chapter 6: Metadata Schema Definition Language," 2011. [Online]. Available: https://www.soumu.go.jp/main_content/000132512.pdf

[6] M. Nagamori, M. Kanzaki, N. Torigoshi, and S. Sugimoto, *Meta-Bridge: A Development of Metadata Information Infrastructure in Japan*. in Proceedings of the International Conference on Dublin Core and Metadata Applications. 2011. doi: 10.23106/dcmi.952135745.

[7] M. Nilsson, "Description Set Profiles: A Constraint Language for Dublin Core Application Profiles," 2008. [Online]. Available: https://www.dublincore.org/specifications/dublin-core/dc-dsp/

[8] DCMI Application Profiles Working Group, "Dublin Core Tabular Application Profiles (DCTAP)," 2024. [Online]. Available: https://www.dublincore.org/specifications/dctap/

[9] N. Thalhath, M. Nagamori, T. Sakaguchi, and S. Sugimoto, *Authoring Formats and Their Extensibility for Application Profiles*. in Digital Libraries at the Crossroads of Digital Information for the Future (ICADL 2019), no. 11853. Springer, 2019. doi: 10.1007/978-3-030-34058-2_12.

[10] N. Thalhath, M. Nagamori, T. Sakaguchi, and S. Sugimoto, *Yet Another Metadata Application Profile (YAMA): Authoring, Versioning and Publishing of Application Profiles*. in Proceedings of the International Conference on Dublin Core and Metadata Applications. 2019. doi: 10.23106/dcmi.952141854.

[11] N. Thalhath, M. Nagamori, and T. Sakaguchi, "Metadata Application Profile as a Mechanism for Semantic Interoperability in FAIR and Open Data Publishing," *Data and Information Management*, 2025, doi: 10.1016/j.dim.2024.100068.

[12] N. Thalhath, M. Nagamori, T. Sakaguchi, and S. Sugimoto, *Metadata Application Profile Provenance with Extensible Authoring Format and PAV Ontology*. in Semantic Technology (JIST 2019), no. 12032. Springer, 2020. doi: 10.1007/978-3-030-41407-8_23.

[13] N. Thalhath, M. Nagamori, and T. Sakaguchi, *YAMAML: An Application Profile Based Lightweight RDF Mapping Language*. in From Born-Physical to Born-Virtual: Augmenting Intelligence in Digital Libraries (ICADL 2022), no. 13636. Springer, 2022. doi: 10.1007/978-3-031-21756-2_32.

[14] Apple Inc., "Pkl: A configuration-as-code language with rich validation and tooling." [Online]. Available: https://pkl-lang.org/

[15] A. Powell, M. Nilsson, A. Naeve, P. Johnston, and T. Baker, "DCMI Abstract Model," 2007. [Online]. Available: https://www.dublincore.org/specifications/dublin-core/abstract-model/

[16] M. Kanzaki, "OWL-DSP: An OWL ontology for Description Set Profiles." [Online]. Available: https://www.kanzaki.com/ns/dsp

[17] N. Thalhath, M. Nagamori, T. Sakaguchi, D. Kasaragod, and S. Sugimoto, *Semantic Web Oriented Approaches for Smaller Communities in Publishing Findable Datasets*. in Metadata and Semantic Research (MTSR 2020), no. 1355. Springer, 2021. doi: 10.1007/978-3-030-71903-6_23.

[18] Library of Congress, "BIBFRAME Profile Editor." [Online]. Available: https://bibframe.org/profile-edit/

[19] Linked Data for Production 2 (LD4P2), "Sinopia Profile Editor." [Online]. Available: https://github.com/LD4P/sinopia_profile_editor

[20] Library of Congress, "DCTap-Dancer." [Online]. Available: https://www.bibframe.org/dancer/

[21] S. A. T. Moxon *et al.*, "LinkML: an open data modeling framework," *GigaScience*, vol. 15, 2026, doi: 10.1093/gigascience/giaf152.

[22] J. Wright, *N3.js Reasoner: Implementing reasoning in N3.js*. in Proceedings of the ISWC 2024 Posters, Demos and Industry Tracks (23rd International Semantic Web Conference), no. 3828. CEUR-WS.org, 2024. [Online]. Available: https://ceur-ws.org/Vol-3828/paper23.pdf