Yet Another Metadata Application Profile (YAMA): Authoring, Versioning and Publishing of Application Profiles

Nishad Thalhath
Graduate School of
Library, Information and Media Studies
University of Tsukuba, Japan
nishad@slis.tsukuba.ac.jp

Mitsuharu Nagamori
Faculty of
Library, Information and Media Studies
University of Tsukuba, Japan
nagamori@slis.tsukuba.ac.jp

Tetsuo Sakaguchi
Faculty of
Library, Information and Media Studies
University of Tsukuba, Japan
saka@slis.tsukuba.ac.jp

Shigeo Sugimoto
Faculty of
Library, Information and Media Studies
University of Tsukuba, Japan
sugimoto@slis.tsukuba.ac.jp

Abstract

Metadata Application Profiles are the elementary blueprints of any Metadata Instance. Efforts like the Singapore Framework for Dublin Core Application Profiles define the framework for designing metadata application profiles to ensure interoperability and reusability. However, the number of publicly accessible, especially machine actionable application profiles are significantly lower. Domain experts find it difficult to create application profiles, considering the technical aspects, costs and disproportionate incentives. Lack of easy-to-use tools for Metadata Application Profile creation is also a reason for lack of larger reach. This paper proposes Yet Another Metadata Application Profile (YAMA) as a user-friendly interoperable preprocessor for creating, maintaining and publishing Metadata Application Profiles. YAMA helps to produce various formats and standards to express the Metadata Application Profiles, changelogs, and different versions, with an expectation of simplifying Metadata Application Profile creation process for domain experts. YAMA includes an integrated syntax for recording application profiles as well as changes between different versions. A proof of concept toolkit, demonstrating the capabilities of YAMA is also being developed. YAMA boasts a human readable yet machine actionable syntax and format, which is seamlessly adaptable to modern version control workflows and expandable for any specific requirements.

Keywords: Metadata Application Profile, Metadata Schema, Schema Publication, Metadata Application Profile Creation

1. Introduction

The concept of Metadata Application Profiles (MAP) is not new in the information community. The initial definition of application profiles is "schemas which consist of data elements drawn from one or more namespaces, combined together by implementors, and optimized for a particular local application (Heery & Patel, 2000)." Dublin Core Metadata Initiative (DCMI) defines one of the earliest guidelines with Description Set Profiles (DSP), a constraint language for Dublin Core Application Profiles based on Singapore Framework for application profiles (Nilsson, Baker, & Johnston, 2008).

Even though there are definite needs and guidelines to create application profiles, especially machine actionable, the number of publicly accessible application profiles are fewer than it should be. There is also a lack of availability of machine-friendly DSP. One of the main reasons is because

of the lack of simplified workflows or tools, due to which the task of application profile generation is tedious and with fewer incentives.

Other significant challenges with application profiles are versioning, change management, and machine friendly changelogs. Application profiles are often created with visual oriented tools such as word processors or spreadsheets and serve the purpose of documentation rather than being actionable. A preliminary investigation over available application profiles shows a clear need for simplified options to encourage metadata developers to adopt the actionable MAP development.

The initial proposal was to define the MAP in a tabular matrix, but a tabular form of records comes with its own limitations. Primarily, it is difficult to build up semantics and hierarchical structure of the DSP in a single spreadsheet. The tabular form represents data in repetitive cells and rows. This matrix form is not nested and some additional efforts like splitting into multiple files or introducing some special notations are required to encode hierarchical data in spreadsheets. Also, simple operations like diff or text comparison are complicated due to CSV's nature of holding multiple values in a single line. Eventually, YAML is adopted to represent the MAP. YAML is flexible, understandable text-based data serialization format, which makes it simple to integrate with version control systems like Git, as well as simple text editors to modify and record application profile related changes. Compared to CSV, YAML generally expresses single key-values per line. Also, YAML natively supports comments and blank lines without impacting the data, and this makes it a suitable candidate as an authoring format. Being a text document, YAML can be used as it is for parsing, while various spreadsheets formats may need to be converted into CSV. YAMA can be easily created and maintained with text editors (see FIG 1).

In this paper, we propose a method to define and clarify an application profile format in YAML[1]. We have also developed proof of concept tools to work with the proposed formats and to ensure the validity. The toolkits are anticipated to work with forward and reverse operations related to the MAP development. For example, the toolkit may generate different versions of the application profile from the YAMA document, as well as change log from multiple versions of the application profiles. This study adapts a YAML based format to record, modify and version the MAP creation tasks. A custom YAML specification is used to hold various stages, levels, and releases of MAP. The proposed YAMA format encourages the use of semantic versioning (SemVer)[2] and adopting standards like PAV - (Provenance, Authoring, and Versioning)[3] to release, maintain and distribute versioned releases of MAP (Ciccarese et al., 2013).

```yaml
constraints:

  voc_enc_uri_lcsh: &voc_enc_uri_lcsh
    VocabularyEncodingSchemeURI : http://lcsh.info

  value_string_1_1: &value_string_1_1
    ValueStringConstraint :
      minOccurs : 1
      maxOccurs : 1

statements :

  language : &language
    min         : 0
    max         : 3
    type        : nonliteral
    property    : dcterms:language
    constraints :
      << : [*voc_enc_uri_lcsh, *value_string_1_1]
```

FIG 1 Example of YAMA with re-use of constraint declarations

[1] https://yaml.org/
[2] https://semver.org/
[3] http://purl.org/pav/

2. Significance of Metadata Application Profile

A MAP is an elementary blueprint of a metadata instance. It describes the set of metadata elements, policies, guidelines and vocabularies defined for a particular domain, or implementation. It also declares the metadata terms, information resource, application, or uses. A well-defined application profile documents schemes, vocabularies, policies, and required elements, etc. (Baca, 2016; Hillmann, 2006) . Application profiles for metadata instances play a crucial role in providing the authoritative specification of term usage, support and document the evolution of vocabularies, facilitate interoperability by informing domain consensus, encourage alignment of practice, enable interpretation of legacy metadata and explain the structure of data to data consumers (Baca, 2016; Heery & Patel, 2000; Hillmann, 2006).

2.1. Importance of Changelogs

Creating and maintaining changelogs of application profile versions help to assure the longevity of the metadata. The longevity of the schema is a significant part of metadata longevity. The provenance of metadata schema should be documented and managed for metadata preservation (Li & Sugimoto, 2018). If the changelogs are created in an actionable format, it can be used for different purposes such as creating human-readable changelogs for automated schema migrations. A record of application profile changes is also a record of the metadata changes. Maintaining changelogs for different versions is as important as maintaining accessible formats of the versions itself. Changelogs help to migrate datasets to new profiles or create crosswalks to upgrade the instances. For linked open data (LOD), changelogs help to update linked datasets as well. Changelog permits efficient migration of instances by only migrating the changed parts.

3. Current Status of Application Profiles

3.1. Various Attempts in the Last Decade

Even though the idea of mixing and matching metadata elements was proposed in 2000, a complete recommendation was presented by DCMI in 2004 as DCMI abstract model of MAP (Powell, Nilsson, Naeve, Johnston, & Baker, 2007). In 2007, DCMI presented the Singapore Framework for Dublin Core Application Profiles as a framework for designing metadata applications for interoperability and for documenting for reusability (Nilsson et al., 2008). As a centerpiece of Singapore Framework, DCMI proposed DSP, a constraint language for Dublin Core Application Profiles (Nilsson, 2008). In 2009, Guidelines for Dublin Core Application Profiles published. As a translator for DC DSP, MoinMoin Wiki Syntax for DSP to integrate application profiles in webpages and wikis was introduced (Enoksson, 2008). Other than DCMI, MetaBridge[4] project introduced a spreadsheet-based application profile format named Simple DSP (SDSP) in 2011 (Nagamori, Kanzaki, Torigoshi, & Sugimoto, 2011). The recent development was an ongoing initiative by Karen Coyle, known as RDF-AP, which at the moment utilizes CSV-based notations to create application profiles which are supposed to be machine-actionable as well (Coyle, 2017).

TABLE 1: Attempts on application profile creation

Year	Initiative
2004	DCMI Abstract Model DCMI Recommendation (Powell et al., 2007)
2007	Singapore Framework for Dublin Core Application Profiles (Nilsson et al., 2008)
2008	DCMI DC-DSP (Nilsson, 2008)
2009	A MoinMoin Wiki Syntax for DSP (Enoksson, 2008)
2011	MetaBridge Simple DSP (Nagamori et al., 2011)
2017	RDF-AP (Coyle, 2017)

[4] https://metabridge.jp/

3.2. Status and Availability

The availability, maintenance, and distribution of application profiles are not standardized. MAP identification is possible only with human involvement (Malta & Baptista, 2014). Curating and archiving MAPs are challenging and expensive due to this manual effort involved. Various registry projects still depend more on manual contributions than automated approaches. Publication of application profiles are done only in human-friendly formats, which requires human involvement in the identification processes to distinguish them from other documents. Extracting structured application profile information from spreadsheets or PDF documents is difficult. Lack of versioning and changelogs as well as limited access to previous versions strongly affect the longevity and provenance of metadata information. Absence of standardized publication formats restricts automated harvesting of application profiles, which eventually limits the number of available application profiles in various registry attempts. Metadata registries were intended to use application profiles to promote and support interoperability and reuse (Nagamori et al., 2011). Also, there is a lack of a standardized way to associate data with the MAP on which it is based (Svensson and Verborgh 2017; Svensson, Atkinson and Car 2019).

3.3. Challenges Involved in MAP Creation

Evolution of linked data and adoption of metadata is encouraging different communities to extend their outputs to incorporate various metadata standards; in order to find new applications, approaches, and insights on their data. Different domain experts are already developing profiles suitable for their metadata applications. Developing MAPs is challenging for most of the domain experts as they may not be well aware of the concepts involved in application profile creation. Different communities have different levels of experience in the technical aspects of application profit creation. A serious lack of guidelines on application profile creation and publishing exists. Also, the limited number of well-defined samples and initiatives to archive and curate application profiles is another blockade. There is not any popular interoperable format or preprocessor for creating application profiles. All these facts make application profile creation and expensive process with very little incentives.

4. Addressing the Challenges

Upon a close examination of previous attempts to promote MAP development, some of the shortcomings were identified. And YAMA format is derived from the limitations of its predecessors. The most important challenge is to promote MAP acceptability in diverse communities to enhance interoperability and reuse of possible vocabularies.

4.1. Interoperable Formats

Lack of interoperable preprocessing systems for MAP is one of the prominent challenges. A preprocessor is a language that takes an input written using some simple language syntax and output another format following the syntax of another language specifications. A preprocessor extends the syntax of existing language by adding new syntactic constructions. The user writes the input format using the extended syntax, and then the processor translates it into one or more different formats. Some of the popular preprocessors are Markdown to process HTML (Gruber, n.d.), reStructuredText[5] to process Python documentation (Jones, n.d.). SCSS[6] to generate CSS and CoffeeScript[7] to generate JavaScript.

There are well-defined standards and specifications to create metadata specific markups and formats. For example, a MAP DSP can be represented in DC DSP XML, RDF or human-friendly spreadsheets and CSVs. However, the interoperability of DSPs is not assured, and there is no

[5] https://docutils.readthedocs.io/en/sphinx-docs/user/rst/quickstart.html
[6] https://sass-lang.com/
[7] https://coffeescript.org/

available preprocessor to handle the creation of these formats in a simplified way with fewer efforts. The limited number of application profile DSPs is mainly due to this reason.

4.2. Changelogs and Roadmaps

Changelogs and roadmaps of application profiles are manually created and are often incomplete or not actionable. Changelogs of most of the publicly published application profiles are either available only as human readable format or are not maintained. Integrating changelog maintenance to application profile creation workflows will help the maintainers to keep them in a seamless way as well as generation of various changelog formats can be automated. It can help to reduce errors and efforts in creating changelogs.

4.3. Tooling

For the creation and management of application profiles, there are recommendations such as Me4DCAP which provides a guideline to define, construct, develop, and validate of MAP (Malta & Baptista, 2013). However, tools or systems dedicated to MAP creations does not exist. Usually, application profile maintainers have to depend on various tools to generate specific formats. Dependency on different tools makes the whole process tedious for most of the domain experts. As a result, application profiles were authored in documentation format rather than machine actionable.

By introducing integrated tooling, YAMA attempts to fix this issue by providing a dedicated preprocessor for application profiles as well as various similar formats. The difference in approach is that YAMA acts as an authoring environment as well as a preprocessing format, with various options to import, export or render different formats and packages without depending on multiple tools or skills.

5. YAMA: Yet Another Metadata Application Profile

YAMA is not defined as a new standard of MAP, but YAMA is defined as an easy to use preprocessor to create standard MAP formats. YAMA is intended to be simple for the domain experts can use it without extensive knowledge on MAP. YAMA attempts to solve the absence of an application profile authoring workflow. Considering the rising popularity of GitHub-based workflows, output formats and extensibility to various proposals like ShEx[8], DCAT[9], PROV[10] eliminate the need of repetitive tasks in application profile maintenance. YAMA is an intermediary format for generating or converting various existing standard formats of application profiles. YAMA is usable in other requirements like Data Catalog Vocabulary (DCAT) and CSV on the Web (CSVW)[11], but not limited to these.

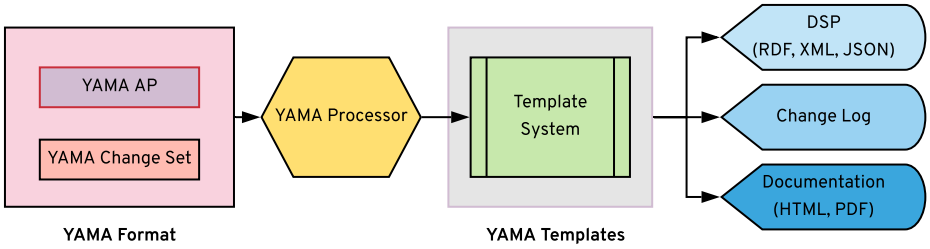

FIG 2: YAMA processing

[8] https://shex.io/
[9] https://www.w3.org/TR/vocab-dcat-2/
[10] https://www.w3.org/TR/prov-o/
[11] https://www.w3.org/TR/tabular-data-primer/

MAP publication consists of both human-friendly and machine-actionable format. The human-friendly format includes documentation of MAP in HTML, Spreadsheet, PDF and DOC. Standard machine-friendly formats are RDF, OWL, and JSON-LD. Validation formats like Shapes Constraint Language (SHACL)[12] and Shape Expressions (ShEx) will not only help to validate the data but they are capable of expressing entire application profile actionable as well.

5.1. Use Cases and Aims

There are different use cases for a preprocessor in MAP creation, some of the possible scenarios can be listed as:

1. Acts as a meaningful and actionable authoring tool for MAP.
2. A preprocessor format acts as a source for MAP, which enhance the maintainability of MAPs.
3. A structured textual format, which fits well with collaborative development environments and version control systems, and make the change tracking convenient with basic diff operations to advanced continuous integration systems.
4. A structured preprocessor format makes it easy to validate the MAP and helps to eliminate errors and logical complexities.
5. A preprocessor with optional change records will act like a development roadmap and will permits generation of previous versions, actionable or human friendly changelogs as well as formats for metadata crosswalks and migrations.

5.2. YAML as a Promising Format

YAML Ain't Markup Language (YAML) is a robust human-friendly data serialization standard with various implementations in most of the popular programming languages. As per the latest specification - version 1.2, YAML is considered as a superset of JSON (Ben-Kiki, Evans, & döt Net, 2009). The strict adherence to readability makes YAML a superior choice of data serialization format for manual creation and modifications. The popularity and acceptance of YAML over JSON are growing in recent years due to its flexibility to express structured data in a textual way without complex syntaxes. Unlike CSV, YAML is friendlier with version control systems like Git and text editors. Being an open format, it prevents any vendor locking on the documents and permits the development of methods and systems to interact with the YAML documents programmatically.

YAML is adapted as a format for projects like OpenAPI Specification (OAS) which defines a standard interface to RESTful APIs (OpenAPI Initiative, 2011), YARRRML which is a human-readable text-based representation for declarative Linked Data generation rules (Heyvaert, De Meester, Dimou, & Verborgh, 2018) and Dead simple OWL design patterns (DOS-DP) which is a simple system for specifying OWL class design patterns (Osumi-Sutherland, Courtot, Balhoff, & Mungall, 2017).

Application profiles are supposed to be structured documents with a descriptive logic. An Application profile written in YAML is structured without any complicated processing. Also, the potential of comments, syntax formatting and highlighting with modern text editors will help to keep the visual and logical organization of an application profile considerably more comfortable. YAML can make a YAMA document self-explanatory and by itself acts as a documentation for the development of MAP.

5.3. Why Extensibility Matters?

Extensibility of a format is the critical element for its acceptance, similar to the philosophy of Dublin Core. Extensibility of an application profile is helpful for various communities to adopt a simple base format and bring in changes from the domain specific requirements. Also, it will help

[12] https://www.w3.org/TR/shacl/

to generate various standard formats from the same YAMA document without depending on the mutually inclusive elements. As a use case, application profile creators can use YAMA as a single source preprocessor to generate various file types, formats, or specifications such as but not limited to DC-DSP, Bibframe JSON or Interactive web documentation.

YAMA is extensible with custom elements and structure. For example, to create constraints, elements from ShEx can be used in the form of structured YAML and custom elements can be added to the document tree, as per the demand of the use case. The only restriction is that custom elements cannot be from reserved element sets. This will help to extend the capabilities of YAMA without any large-scale implementation changes. Any such extension is possible within the scope of YAML specification. There is also a user variables section which is a straightforward approach to add any user-defined variables without altering the structure of a YAMA document.

5.5. Formats and Templates

YAMA helps to generate outputs programmatically or using temples. Templates are the easiest option for domain experts, who are less experienced in various output formats. Authors experimented with temples for various format and observed that the output is as good as programmatically generated counterparts. This approach makes customization of output easier for various communities by rather editing the temples than going through any programming challenges.

Templating based approach is helpful for creating actionable formats like RDF-OWL, XML and JSON-LD. Also, the templates make it easy to render human friendly HTML pages as a web publication medium and can be converted into printer friendly formats like PDF. For advances users, various tools can be used to achieve desired output formats.

6. YAMA Syntax and Specification

6.1. Application Profiles in YAMA

YAMA specification defines textual syntax and specifications for writing YAMA documents in a natural text form. YAMA syntax is based on YAML 1.2 specification. YAMA is parsable with any YAML 1.2 parser, but processing capabilities of YAMA documents are limited to YAMA specific implementations. A complete specification for YAMA format is accessible at `https://purl.org/yama/spec/latest`.

A YAMA document should strictly follow YAML specifications. Document should start with a valid YAML declaration and YAMA version should be mentioned before starting the structure of the document. If a valid YAMA specification version is not declared, then the last available version is assumed to be used.

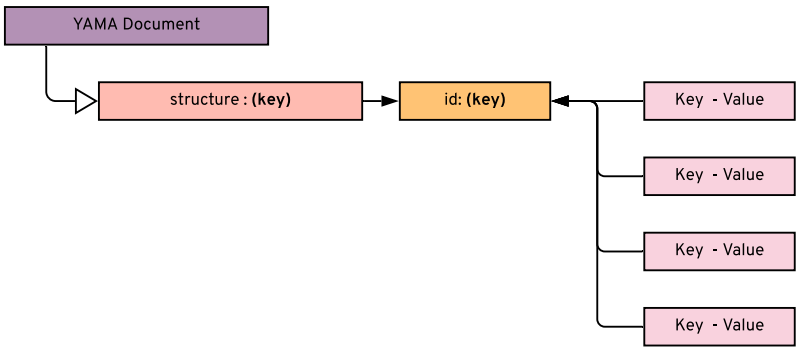

FIG 3: YAMA document tree

YAMA document is structurally organized as Description Set metadata, name spaces, descriptions, statements, constraints, change sets and user defined values.

Metadata section is intended to express basic information of the specific MAP. Generally administrative metadata of the MAP is expressed as a key value pair. Important property from this section is the version and creator. This information is used in generating publishable formats and creating provenance information as well as change log of the MAP.

Single resources are described under descriptions with a unique ID for each description. Every unique descriptions ID can have multiple key value pairs to describe that resource. A statement is a single data element used to describe a resource that is defined as a description. The statement defines the possible values, and any other constraints. A description can have one or more statements, but descriptions without any statements is not actionable. Constraints are reusable components and can be callable through their unique id. Multiple constraints can be mixed and matched to satisfy complex requirements, as well as constraints are permitted to include custom key values or structures.

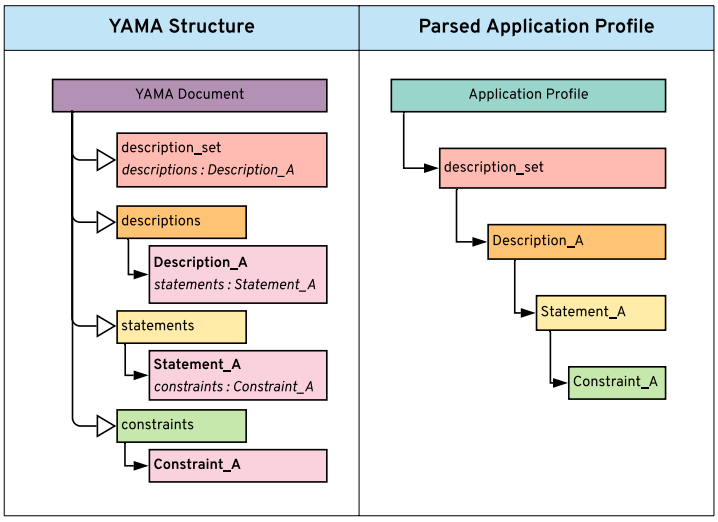

FIG 4: YAMA application profile parsing

6.2. Expressing Change Sets in YAMA

YAMA also proposes syntax and specifications of Change-Sets. Change-Sets concept can be explained as a structured syntax to record modifications of a YAMA document. YAMA change-sets can be used in preprocessing existing documents to create modified version or use to record changes of a document over any other versions. Change-Sets is inspired from RFC 6902[13] JavaScript Object Notation (JSON) Patch (Nottingham & Bryan, 2013), with the changes marked with an action to a path with an additional special reserved value as 'status' to indicate status changes like 'deprecation'.

6.3. Integrating, Extending and Maintaining YAMA

YAMA is expressed in YAML, which is a simple text format. Application profile developers can use any standard text editor to create the YAMA format. Syntax highlighting, prettification, validation, and linting can be achieved with various tools. YAML fits well with Git based workflows and is capable of handling comments as well as blank lines for readability. Using these featured, YAMA can be used as a documentation and roadmap of application profile development.

[13] https://tools.ietf.org/html/rfc6902

YAMA format can be programmatically generated from spreadsheets or other data formats. Some of these approaches were attempted to demonstrate through the proof of concept tool-kit.

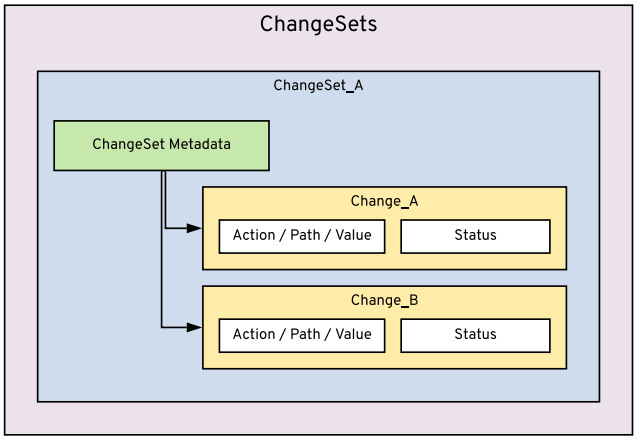

FIG 5: Changeset structure in YAMA

6.4. Recommendations and Best Practices

YAMA documents should be versioned to utilize its capabilities on versioning, changesets, and changelogs. Semantic versioning (SemVer) is highly recommended. Also, calendar versioning (CalVer)[14] can be considered if it fits the requirements. With the proper version number, YAMA processors can automate various versioning related tasks as well as can generate publishable versioned output formats. Version number should be used in output file(s) naming convention as well. A version named file is self-explanatory in URLs as well as Git based authoring systems. Change sets specification is strictly adhered to version numbers and a semantic logic on versioning. In semantic versioning approach MAJOR.MINOR.PATCH is considered in MAP as patch versions doesn't break any MAJOR, MINOR definitions with forward and reverse compatibility. It can be used internally as part of development and for fixes and corrections related to typos and or less significant changes. But public releases can follow MAJOR.MINOR approach with a MINOR being compatible within the same MAJOR releases. Changes which breaks the compatibility should strictly follow a MAJOR version change.

Various standard formats of application profiles are recommended to publish as a single package, accessible in a persistent web URL, which includes the profile id, version, format, and extension in a self-explainable way.

Such as :

```
[ap-id]_[version]/[ap_id]_[version]_[format].[extension]
```

example :

```
http://example.com/ap/my-book-case_1.4/my-book-case_1.4_dsp.rdf
http://example.com/ap/my-book-case_1.4/my-book-
case_1.4_documentation.pdf
http://example.com/ap/my-book-case_1.4/my-book-case_1.4.shex
```

7. YAMA Toolkit

7.1. Python Package and CLI

A proof of concept toolkit is developed as a Python package to work with YAMA specification. This Python package can be used as a module for Python application development, or as a command

[14] https://calver.org/

line tool to work with YAMA format files. This toolkit can parse a YAMA format and return structures applicant profile data or render the structured application profile using custom or built-in Jinja2 templates. Python package is available at `https://purl.org/yama/tools/pyyama`.

With the use of the python package, other template system or advanced libraries such as but not limited to RDFLib and PyShEx can be used to process the parsed structure to generate or manipulate the formats programmatically.

7.2. Templates and Generators

To maintain simplicity and to ensure extensibility and customization capabilities, YAMA Python toolkit can use templates to generate any formats from the parsed structured application profile data. Templates can be used to generate virtually any formats without the complexities of dealing with complex libraries. By default, YAMA toolkit uses a Python templating system named Jinja. Jinja2 is a full featured and one of the most used template engines for Python. It is fast, widely used, and secure with a configurable syntax and logic (Ronacher, 2008). Originally, Jinja2 was designed for HTML templates, but it is suitable for all kind of text-based formats. Using Jinja2 templates, not only HTML, but also complex RDF or JSON data files can be generated. Some of the standard formats are provided as ready-to-use templates. The tool-kit will be updated with more templates depending on various use-cases.

Advanced users can use any Python templating systems to extend the package's templating, or custom python scripts to render the structured AP without templates. The package can optionally use generator scripts other than templates, which are python scripts to generate output formats or packages programmatically without templating. A generator can use complex programming logic or depend on external libraries or programs to generate the desired output.

```
<?xml version="1.0" ?>
<DescriptionSetTemplate xmlns="http://dublincore.org/xml/dc-dsp/2008/03/31">
{% for key, description in dsp.descriptions.items() %}
  <DescriptionTemplate>
    {% for key, statement in description.statements.items() %}
    <StatementTemplate>
      <Property>{{ statement.property }}</Property>
    </StatementTemplate>
    {% endfor %}
  </DescriptionTemplate>
{% endfor %}
</DescriptionSetTemplate>
```

FIG 6: Jinja template for "Simple" Dublin Core DSP

8. Discussion and Future Development

YAMA is developed to be a direct adaption of DC-DSP. It is heavily inspired by the SDSP format developed for the MetaBridge project (Nagamori et al., 2011). YAMA is an attempt to promote the acceptance of application profile concepts to various communities with less technical expertise. As a simplified format, it has many limitations in expressing complicated application profiles or use cases. However, advanced users can still create them manually or use YAMA programmatically to overcome such limitations. Improvements for the YAMA specification and toolkit is being investigated. The modular structure is expected to expand to more object-oriented design compatible with the simplified format. Also, the toolkit will include the standard libraries to deliver some of the advanced features.

Even though YAMA is efficient in authoring application profiles, YAML format and the structure could pose a challenge to editors. In other words, the ideal user of YAMA would need to be comfortable editing YAML directly, that requires the users to have the expertise of using a real text editor.

Conclusion

MAP is getting a lot of popularity and acceptance in different communities. YAMA format is an attempt to express the idea of simplifying the tooling and to support some of the tedious processes involved in creating and maintaining application profiles. Compared to its predecessors, YAMA provides an authoring environment, format, and provision for a toolkit for application profiles. Evolution of proposals like ShEx and DC-DSP2 give application profile more use cases and functions.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number JP18K11984.

References

Baca, M. (2016, July 20). Introduction to Metadata [InteractiveResource]. Retrieved April 11, 2019, from http://www.getty.edu/publications/intrometadata

Ben-Kiki, O., Evans, C., & döt Net, I. (2009, October 1). YAML Ain't Markup Language (YAML™) Version 1.2. Retrieved April 10, 2019, from https://yaml.org/spec/1.2/spec.html

Ciccarese, P., Soiland-Reyes, S., Belhajjame, K., Gray, A. J., Goble, C., & Clark, T. (2013). PAV ontology: Provenance, authoring and versioning. Journal of Biomedical Semantics, 4(1), 37. https://doi.org/10.1186/2041-1480-4-37

Coyle, K. (2017). *RDF-AP*. Retrieved from https://github.com/kcoyle/RDF-AP (Original work published 2017)

Enoksson, F. (2008, October 6). DCMI: A MoinMoin Wiki Syntax for Description Set Profiles. Retrieved March 11, 2019, from http://www.dublincore.org/specifications/dublin-core/dsp-wiki-syntax/

Gruber, J. (n.d.). Daring Fireball: Markdown. Retrieved May 14, 2019, from https://daringfireball.net/projects/markdown/

Heery, R., & Patel, M. (2000). Application Profiles: Mixing and Matching Metadata Schemas. *Ariadne*, (25). Retrieved from http://www.ariadne.ac.uk/issue/25/app-profiles/

Heyvaert, P., De Meester, B., Dimou, A., & Verborgh, R. (2018). Declarative Rules for Linked Data Generation at Your Fingertips! In A. Gangemi, A. L. Gentile, A. G. Nuzzolese, S. Rudolph, M. Maleshkova, H. Paulheim, … M. Alam (Eds.), *The Semantic Web: ESWC 2018 Satellite Events* (Vol. 11155, pp. 213–217). https://doi.org/10.1007/978-3-319-98192-5_40

Hillmann, D. (2006). Metadata standards and applications. Retrieved April 26, 2019, from http://managemetadata.com/msa_r2/

Jones, R. (n.d.). A ReStructuredText Primer — Docutils 3.0 documentation. Retrieved May 14, 2019, from https://docutils.readthedocs.io/en/sphinx-docs/user/rst/quickstart.html

Li, C., & Sugimoto, S. (2018). Provenance description of metadata application profiles for long-term maintenance of metadata schemas. *Journal of Documentation*, *74*(1), 36–61. https://doi.org/10.1108/JD-03-2017-0042

Malta, M. C., & Baptista, A. A. (2013). *A Method for the Development of Dublin Core Application Profiles (Me4DCAP V0.2): Detailed Description*. 14.

Malta, M. C., & Baptista, A. A. (2014). A panoramic view on metadata application profiles of the last decade. *International Journal of Metadata, Semantics and Ontologies*, *9*(1), 58. https://doi.org/10.1504/IJMSO.2014.059124

Nagamori, M., Kanzaki, M., Torigoshi, N., & Sugimoto, S. (2011). *Meta-Bridge: A Development of Metadata Information Infrastructure in Japan*. 6.

Nilsson, M. (2008, March 31). DCMI: Description Set Profiles: A constraint language for Dublin Core Application Profiles. Retrieved March 1, 2019, from http://www.dublincore.org/specifications/dublin-core/dc-dsp/

Nilsson, M., Baker, T., & Johnston, P. (2008, January 14). DCMI: The Singapore Framework for Dublin Core Application Profiles. Retrieved May 18, 2019, from http://dublincore.org/specifications/dublin-core/singapore-framework/

Nottingham, M., & Bryan, P. (2013, April). JavaScript Object Notation (JSON) Patch. Retrieved March 18, 2019, from https://tools.ietf.org/html/rfc6902

OpenAPI Initiative. (2011, August 10). OpenAPI Specification. Retrieved May 15, 2019, from https://swagger.io/specification/

Osumi-Sutherland, D., Courtot, M., Balhoff, J. P., & Mungall, C. (2017). Dead simple OWL design patterns. *Journal of Biomedical Semantics*, *8*(1), 18. https://doi.org/10.1186/s13326-017-0126-0

Powell, A., Nilsson, M., Naeve, A., Johnston, P., & Baker, T. (2007). DCMI: DCMI Abstract Model. Retrieved May 1, 2019, from http://www.dublincore.org/specifications/dublin-core/abstract-model/

Ronacher, A. (2008). Jinja2 (The Python Template Engine). Retrieved April 11, 2019, from http://jinja.pocoo.org/

Svensson, L. & Verborgh, R. (2017, March 9). Negotiating Profiles in HTTP. Retrieved May 18, 2019, from https://profilenegotiation.github.io/I-D-Accept--Schema/I-D-accept-schema

Svensson, L. G., Atkinson, R., & Car, N. J. (2019, April 30). Content Negotiation by Profile. Retrieved May 16, 2019, from https://www.w3.org/TR/dx-prof-conneg/