Maintaining and Publishing Metadata Application Profiles with Extensible Authoring Format

著者（英）	Nishad Thalhath REHUMATH
内容記述	Thesis (Master of Science in Library and Information Studies)--University of Tsukuba, no. 41490, 2019.9.25
year	2019
URL	http://hdl.handle.net/2241/00161356

Master's Thesis in Graduate School of Library, Information and Media Studies

Maintaining and Publishing Metadata Application Profiles with Extensible Authoring Format

September 2019

201726107

Nishad Thalhath Rehumath

Maintaining and Publishing Metadata Application Profiles with Extensible Authoring Format

Nishad Thalhath Rehumath

Graduate School of Library, Information and Media Studies

University of Tsukuba

September 2019

Maintaining and Publishing Metadata Application Profiles with Extensible Authoring Format

Student No.: 201726107
Name: Nishad Thalhath Rehumath

Metadata Application Profiles are the elementary blueprints of any Metadata Instance. They act as a key element in metadata interoperability. Singapore Framework for Dublin Core Application Profiles defined the framework for designing metadata application profiles to ensure interoperability and reusability.

There are various accepted formats to express application profiles. Most of these expression formats such as RDF, OWL, JSON-LD, SHACL, and ShEx are machine actionable, and formats like spreadsheets or web pages act as human-readable documentation. Due to limited options in the mutual conversion of these expression formats, they are often created independent of each other and thus makes the process expensive requiring sophisticated skills and time. Proposals for convertible authoring formats to create application profiles have received less acceptance, mainly due to their inability to address various use cases and requirements.

As a result, domain experts find it difficult to create application profiles, considering the technical aspects, costs and disproportionate incentives and the lack of easy-to-use tools for Metadata Application Profile creation.

This study proposes Yet Another Metadata Application Profile (YAMA) as a user-friendly authoring format for creating, maintaining and publishing Metadata Application Profiles. YAMA helps to produce various standard expressions of the Metadata Application Profiles, change logs, 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.

This study argues that the extensible authoring formats are suitable for creating application profiles with custom requirements and different use cases. This would promote the acceptance of application profiles by reducing the associated cost and skill requirements in creation, maintenance, and publishing of application profiles.

Academic Advisors: Principal: SAKAGUCHI Tetsuo
Secondary: NAGAMORI Mitsuharu

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Metadata Application Profiles

The concept of Metadata Application Profiles (MAP) or simply Application Profiles (AP) is not new in the information science community. The initial definition of application profiles is “schemas, which consist of data elements drawn from one or more namespaces, combined together by implementers, and optimized for a particular local application.” [7]

A MAP is the 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. [1, 9]. Application profiles for metadata instances play a crucial role to provide 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 [1, 7, 9]. A general overview of a MAP is given in figure 1.1.

1.2 The Singapore Framework for Dublin Core Application Profiles

The Singapore Framework for Dublin Core Application Profiles [1] is a framework proposed for designing metadata applications. Singapore Framework ensures maximum interoperability and reusability by documenting applications. This framework specifies a set of descriptive components necessary for documenting an application profile and describing the relationship between these documented standards domain models and semantic web standards. This framework establishes a basic guideline for evaluating application profiles for completeness of documentation

[1]http://dublincore.org/specifications/dublin-core/singapore-framework/

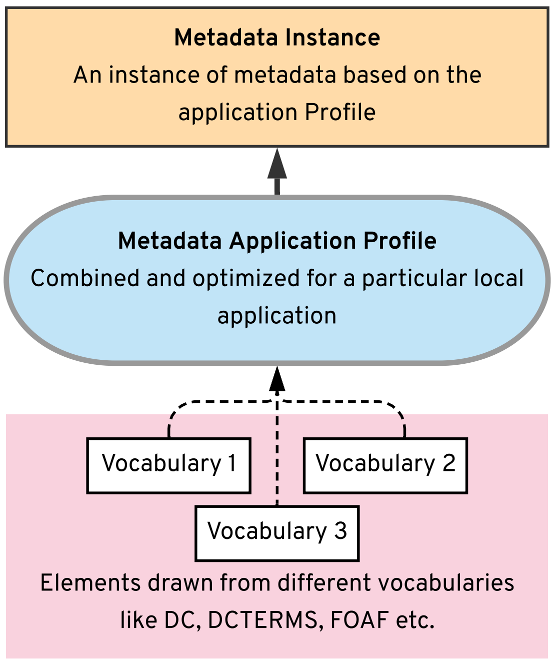

Metadata Instance
An instance of metadata based on the application Profile
Metadata Application Profile
Combined and optimized for a particular local application
Vocabulary 1
Vocabulary 2
Vocabulary 3
Elements drawn from different vocabularies like DC, DCTERMS, FOAF etc.

Figure 1.1: A general overview of a Metadata Application Profile

and confirms with the principles of web-architecture [5]. Singapore Framework is illustrated in figure 1.2.

Singapore Framework defines a Dublin Core Application Profile as a packet of documentation which consists of different components. They are :

1. Functional requirements, which describe the functions that the application profile is support. Also acts as a basis of evaluating the application profile for internal consistency and gives guidance on the aptness of the application profile for a specific use.

2. Domain model, which defines the basic entities described by the application profile and their primary relationships. Domain model defines a necessary scope for the application profile.

3. Description Set Profile (DSP), which is a set of metadata records that are valid instances of an application profile. The Dublin Core Description Set Profile (DSP) offers a simple constraint language based on the DCMI Abstract Model.

4. Usage guidelines, which describe the application of the application profile

5. Encoding syntax guidelines describe any application profile-specific syntaxes and syntax guidelines.

Singapore Framework also includes the other two components, Domain Standards, and Foundation Standards.

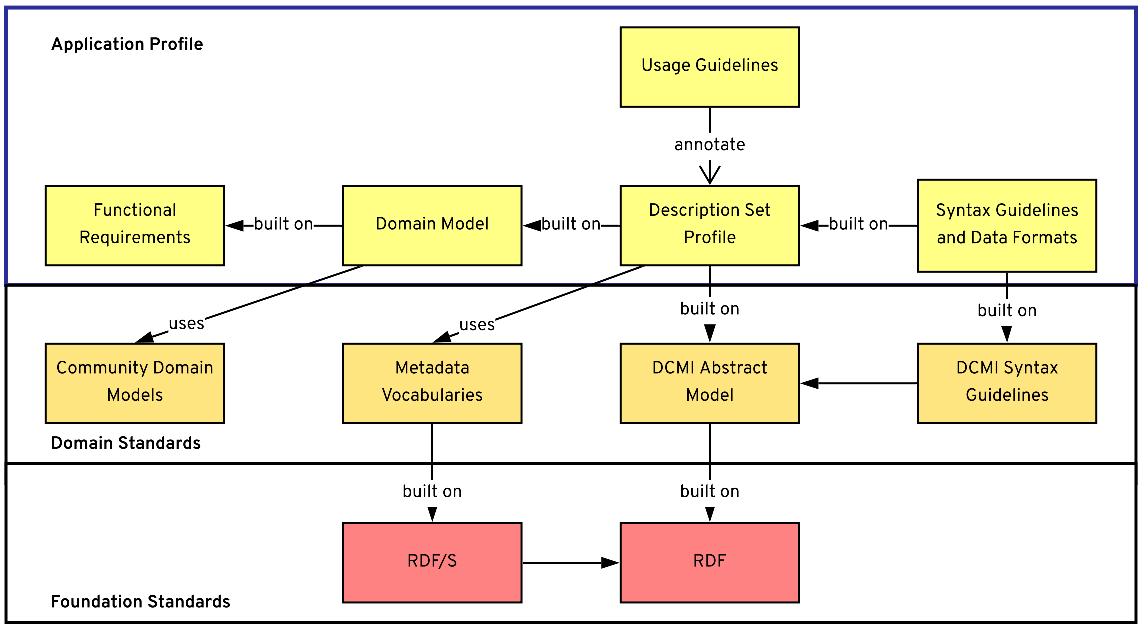


Figure 1.2: Singapore Framework for Dublin Core Application Profiles

1.3 Management of Application Profiles

Metadata application profile management can be separated into three distinct stages. The first and the critical stage is planning and designing of an application profile. In which domain experts design a domain-specific metadata application profile by selecting vocabularies from different namespaces and combining them to make it suitable for a specific local application. The next stage is creating and publishing of this application profile, which involves creating human-readable documentation and machine-actionable expressions of the application profile. This stage is more technical than planning and designing. The third and final stage is updating and maintaining application profiles. This third stage is a continuous process which includes updating the versions of the application profiles along with its development and maintaining changelings as well as ensuring the availability of current and previous versions of the application profiles and different expression formats.

Chapter 2

Status of Application Profile

2.1 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 [20]. 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 [17]. As a centerpiece of Singapore Framework, DCMI proposed DSP, a constraint language for Dublin Core application profiles [17]. In 2009, Guidelines for Dublin Core Application Profiles published and as a translator for DC DSP, MoinMoin Wiki Syntax for DSP to integrate application profiles in webpages and wikis was introduced [5]. Other than DCMI, MetaBridge project introduced a spreadsheet-based application profile format named Simple DSP (SDSP) in 2011 [15]. 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. [4].

2.2 Current Status and Availability of Application Profiles

The availability, maintenance, and distribution of application profiles are not standardized. Identifying MAPs requires human involvement [14]. Curating and

Table 2.1: Attempts on application profile creation

Year	Initiative
2004	DCMI Abstract Model DCMI Recommendation [20]
2007	Singapore Framework for Dublin Core Application Profiles [17]
2008	DCMI DC-DSP [16]
2009	A MoinMoin Wiki Syntax for DSP [5]
2011	MetaBridge Simple DSP [15]
2017	RDF-AP [4]

archiving MAPs are challenging and expensive due to this manual effort involved. Various registry projects still depend on manual contributions than automated approaches. Publication of application profiles is 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, changelogs, and 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 [15]. Also, there is a lack of a standardized way to associate data with the MAP on which it is based [23] [24].

2.3 Expression Formats for Application Profiles

MAP publication consists of both human-friendly and machine-actionable format. The human-friendly format includes documentation of MAP in HTML, Spreadsheet, PDF, and DOCX. Standard machine-friendly expression formats are RDF [1], OWL [2], and JSON-LD [3]. Validation formats like Shapes Constraint Language (SHACL) [4] and Shape Expressions (ShEx)[5] will not only help to validate the data but they are capable of expressing entire application profile actionable as well. Due to limited options in the mutual conversion of these expression formats, they are often created independent of each other and thus makes the process expensive by demanding sophisticated skills and time.

Detailed class diagrams, as well as documentation in different formats, will make the application profiles cover more use-cases. In general, a well-defined metadata application profile requires different expression formats to make it cover the maximum recommendations from the Singapore Framework for Dublin Core Application Profiles [17].

2.4 Authoring Formats for Application Profiles

An authoring format of application profile can be defined as a source for application profile publication, which helps to generate different application profile expression formats using a processor or converter. Authoring formats cannot be treated as an

[1]https://www.w3.org/RDF/
[2]https://www.w3.org/OWL/
[3]https://json-ld.org/
[4]https://www.w3.org/TR/shacl/
[5]http://shex.io/

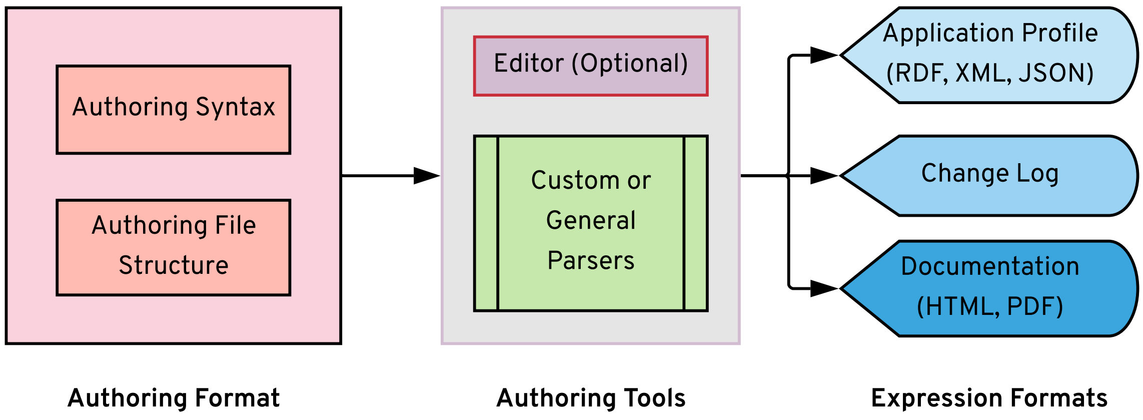


Figure 2.1: Authoring formats and expression formats for application profiles

expression of an application profile, as they may not be a standard representation of an application profile, and their capability to do so always depends on its processors or designated tools. This clear separation between authoring and expression formats is illustrated in Figure 2.1.

There were different attempts to define and develop authoring formats or tools in application profile creation. The first notable attempt is by DCMI, along with DCMI's proposal for DSP as a constraint language for Dublin Core application profiles, introduced a translator which used MoinMoin Wiki Syntax as an authoring format for DSP to integrate application profiles in webpages and wikis was introduced [5]. MetaBridge project introduced a spreadsheet-based application profile authoring format named Simple DSP (SDSP) [15]. One of the recent developments is an ongoing initiative named RDF-AP, which at the moment utilizes CSV-based notations to author application profiles [4]. The BIBFRAME project from the Library of Congress has developed two web-based profile editors named BIBFRAME Profile Editor, which allows to modify or create profiles [3]. BIBFRAME editor is in mainly intended to create the profiles of the BIBFRAME vocabulary but can be usable for other profiles as well. Linked Data for Production 2 (LD4P2) project published a modified BIBFRAME editor as a general online application profile authoring tool named Sinopia Profile Editor [21].

All these stated authoring formats are not extensible. Extensibility of a format is critical for its acceptance, which helps various communities to adopt a simple base format and bring in changes from the domain-specific requirements. Also, it will help to generate various standard formats from the same source document without just depending on the mutually inclusive elements.

Chapter 3

Problems and Challenges in MAP Authoring

3.1 Challenges in Metadata Application Profile Creation

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 the Singapore framework for application profiles. [17] 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. Also, there is 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 human visual-oriented tools such as word processors or spreadsheets and serve the purpose of documentation other 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.

3.2 MAP creation for various domains

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 are 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 severe 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 minimal incentives.

3.3 Limitation of existing Authoring formats

There are not many authoring formats for application profiles, and the available options are either minimal or specific to particular purposes. A general-purpose application profile authoring format is not yet wholly proposed by any other initiatives. Most of these authoring formats are either of limited functionality or highly optimized for a specific purpose. For example, in the case of simple DSP, the CSV input format is heavily optimized and simplified for encoding only bare minimal application profile information. Authoring environments such as Bibframe profile editor is useful for making the process simplified. On the contrary, bibframe profile editor or its variant Sinopia profile editor [1] are restricting the users to stick within the derived framework of bibframe profiles. Even though they are useful in creating application profiles in general-purpose, it requires a significant level of expertise in creating application profiles.

3.4 Provenance and Change logs

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. [12]. If the changelogs are created in an actionable format, it can be used for different purposes such as creating human-readable changelogs to automated schema migrations. A record of application profile changes is also a record of the metadata changes. Maintaining change logs for different versions is as crucial as maintaining accessible formats of the versions itself. Changelogs help to migrate datasets to new profiles or create crosswalks to upgrade the instances. For LOD, changelogs help to update linked datasets as well. Changelog permits efficient migration of instances by only migrating the changed parts.

[1]https://profile-editor.sinopia.io

Chapter 4

Proposed Solutions

Upon a close examination of previous attempts to promote MAP development, some of the shortcomings were identified. Moreover, YAMA format is derived from the limitations of its predecessors. The most crucial 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 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 [6], reStructuredText to process Python documentation [11]. Sass to generate CSS and CoffeeScript 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 such 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 Change logs 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 [12]. Integrating change log maintenance to application profile creation workflows will help the maintainers to keep them in a seamless way as well as generation of the 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 scope definition, construction, development, and validation[13]. 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.

4.4 Extendable Authoring Formats

A better authoring format, which includes the above-said solution, can be considered as the best and single-shot solution. Moreover, such authoring format should provide a dedicated preprocessor for application profiles as well as various similar formats. The best recommendation for such authoring format is, it should act 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.

Chapter 5

YAMA: Yet Another Metadata Application Profile

This study proposes a YAML based authoring formats named Yet Another Metadata Application Profile (YAMA), an applicaiton profile authoring 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 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 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. The data expression capability of YAML is superior to that of CSV in many ways. For example, CSV expresses only simple tabular data in rows and columns, but YAML can represent complex data in simple key-value pairs to nested trees. 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, but various spreadsheets formats may need to be converted into CSV. YAMA can be easily created and maintained with text editors.

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 enough that 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, various output formats and extensibility to various proposals like ShEx, DCAT, PROV and 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), but not limited to these.

5.1 YAMA as an Authoring Format

Yet Another Metadata Application Profile (YAMA) is proposed by the authors as an extensible authoring format for application profile, which addresses some of the shortcomings of the previous proposals [25]. YAMA is intended to be simple enough that domain experts can use it without extensive knowledge on MAP. YAMA uses YAML Ain't Markup Language (YAML), which is a robust human-friendly data serialization standard with various implementations in most of the popular programming languages and considered as a superset of JSON [2].

YAMA is also an attempt to solve the absence of an application profile authoring workflow. Considering the rising popularity of GitHub-based workflows, various output formats and extensibility to various proposals like ShEx, DCAT, PROV eliminate the need for repetitive tasks in application profile maintenance. YAMA is an intermediary format for generating or converting various existing standard formats of application profiles.

YAMA is extensible with custom elements and structure. For example, 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. YAMA is based on DC-DSP, and a minimal DC-DSP is mandatory to express a MAP in YAMA. Along with the extensible key-values and structure YAMA also includes a structured syntax to record modifications of a YAMA document named as change-sets. YAMA change-sets can be used to record changes of a MAP over any other versions. Change-sets are adapted from RFC 6902 JavaScript Object Notation (JSON) Patch [18], with the changes marked as an action to a path. Every change use 'status' as a reserved value to indicate status changes like 'deprecation.' This extensible nature of YAMA documents is explained in Figure 5.1.

5.2 YAML as a Promising Format

YAML Ain't Markup Language (YAML)[1] is a robust human-friendly data serialization standard with various implementations in most of the popular programming

[1]https://yaml.org/

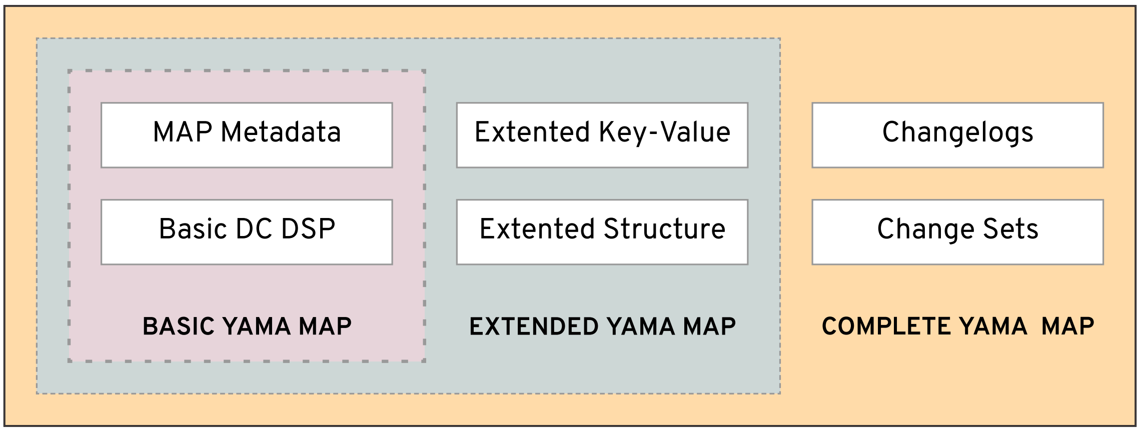


Figure 5.1: Extensibility of a YAMA Application Profile

languages. As per the latest specification - version 1.2 [2], YAML is considered as a superset of JSON [2]. 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) [3] which defines a standard interface to RESTful APIs [10], YARRRML which is a human-readable text-based representation for declarative Linked Data generation rules [8]. and Dead simple OWL design patterns (DOS-DP) [4] which is a simple system for specifying OWL class design patterns [19].

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 Extensibility of YAMA

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

[2]https://yaml.org/spec/1.2/spec.html
[3]https://www.openapis.org/
[4]https://github.com/INCATools/dead_simple_owl_design_patterns

various communities to adopt a simple base format and bring in changes from the domain specific requirements. Also, it will help to generate various standard formats from the same YAMA document without just 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.

5.4 Expected outputs for YAMA

YAMA document has different components which act as part of different output purposes. YAMA document has mandatory metadata for application profiles, section to record application profile, optional changesets to record changes between different versions and optional changelog for recording actionable changelog information. These sections can be used in various purposes, for example application profile metadata can be used to ensure provenance of the metadata the application profile. Same way ChangeSets and changelogs in congestion with the administrative metadata of the application profile can be used to generate longevity information such as human-readable and machine actionable changelogs as well as different versions of the application profile. However the main functionality of YAMA document is to act is an authoring format for application profiles. This can be achieved by combining the administrative metadata part of the application profile and the application profile section of the document. This clearly illustrates the capability of YAMA to be an authoring format for metadata application profiles, as well as a suitable way to ensure provenance and longevity.

A detailed illustration of this components to output mapping is given in figure 5.2.

5.5 Possible use-cases for YAMA

YAMA acts as a meaningful and actionable authoring tool for MAP. As a preprocessor format, it serves as a source for MAP. A source format enhances the maintainability of MAPs. Being YAMA, a structured textual format, YAMA fits well with collaborative development environments and version control systems. That makes the change tracking convenient with basic diff operations to advanced continuous integration systems. As a structured authoring format, YAMA makes it easy to validate the MAP and helps to eliminate errors and logical complexities.

YAMA's capability of recording optional change records will act as a development roadmap and will permit the generation of previous versions, actionable or human-friendly change logs as well as formats for metadata crosswalks and migrations. YAMA is also suitable to generate human-friendly documentation from the serialized source, with the help of templating or other means. As a template-first system, It is easy to customize the output formats. YAMA's extensibility helps to tailor it to suit for different community/use cases. Being TAML, a text-based format, YAMA is highly interoperable with various text editors/tools.

5.6 Advantages of YAMA as an authoring format

As an authoring format, YAMA significantly reduces the efforts required in authoring and maintaining application profile. YAMA helps to record an application profile into a highly flexible and extensible data-serialization format. Also allows generating the expressions of an application profile in various standard expressions, such as RDF, XML, ShEx, SHACL, and JSON. This simplification helps users with moderate skills in these formats to create application profiles with minimal efforts. As a result, the barriers involved - such as the cost, expertize, and other challenges - in application profile creation can be lowered.

Instead of proposing a new format for application profile, YAMA helps to record the core elements required to produce various existing standard formats and generate them. This expressions are created either using simple templates for the normal user or programmatically for advanced users. The significant advantage of YAMA is, it can be a single source for different expressions. A single source makes the maintenance of application profiles more comfortable than maintaining multiple independent formats.

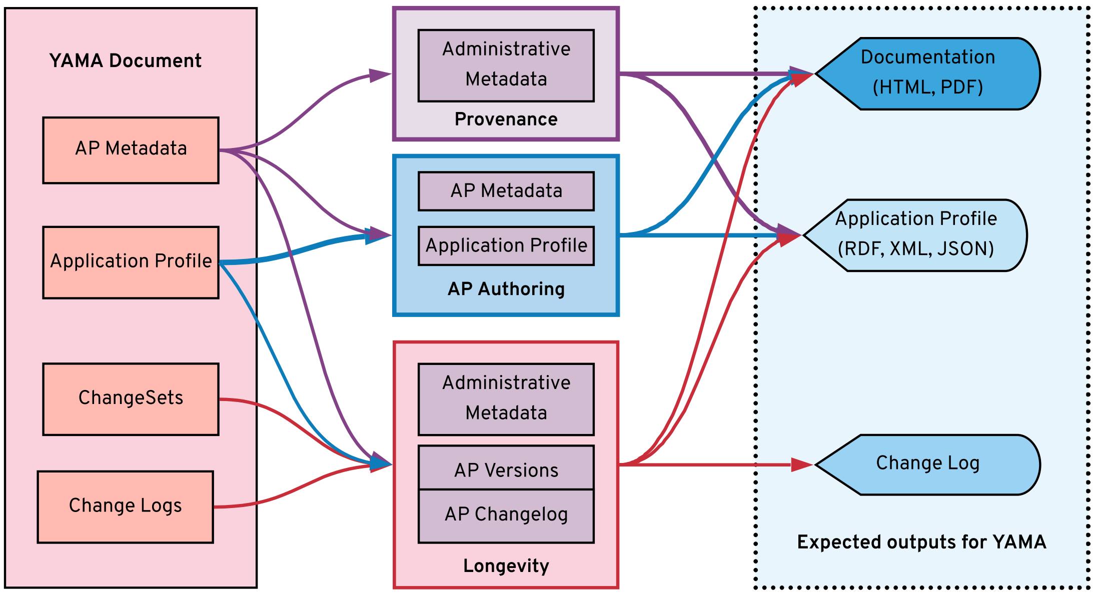


Figure 5.2: Components of YAMA and their expected outputs

Chapter 6

YAMA Syntax and Specifications

6.1 Expressing 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. The 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.

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

Metadata section is intended to express necessary 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 the changelog 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 are 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. Structure of a YAMA document is explained in figure 6.1 and example code for a YAMA document is given in figure 6.2.

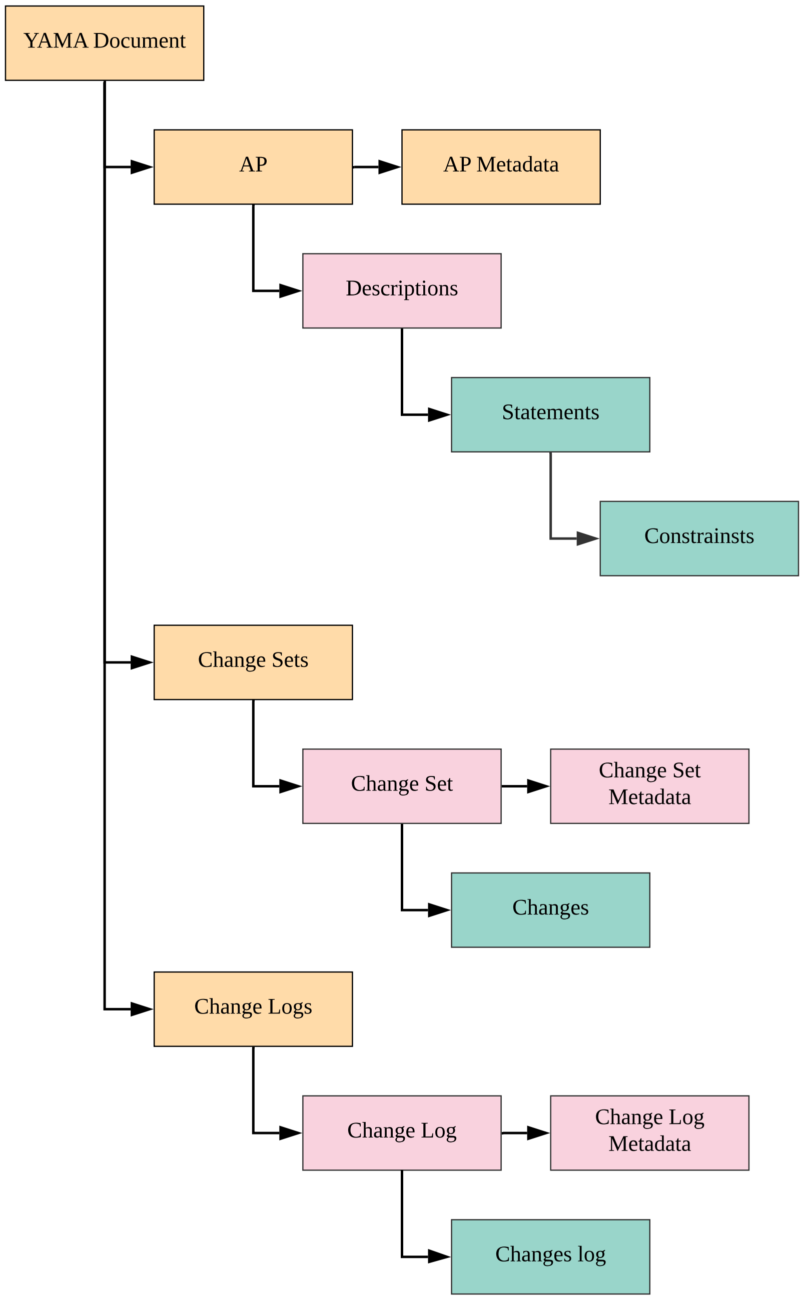


Figure 6.1: Structure of YAMA Document

```
%YAML 1.2
---
#%YAMA 0.8
meta:
  id: dcat-ap
  title: DCAT Application Profile for Data Portals in Europe
  version: 1.2.1
  date: '2019-05-28'
  subject: Application profile for data portals in Europe (DCAT-AP)
  license: ISA Open Metadata Licence v1.1
  creator: DCAT-AP Working Group
  website: https://github.com/SEMICeu/DCAT-AP

# rest of the section is omitted in this example #

namespaces:
  adms: http://www.w3.org/ns/adms#
  dcat: http://www.w3.org/ns/dcat#
  dct: http://purl.org/dc/terms/

# rest of the section is omitted in this example #

constraints:
  voc_dataset_theme_vocabulary: &voc_dataset_theme_vocabulary
    type: vocabulary
    vocabulary_name: Dataset Theme Vocabulary
    URI: http://publications.europa.eu/resource/dataset/data-theme
    notes: The value to be used for this property is the URI of the vocabulary
     ↪  itself,
      i.e. the concept scheme, not the URIs of the concepts in the vocabulary.

# rest of the section is omitted in this example #

statements:
  pr_dataset: &pr_dataset
    label: dataset
    property: dcat:dataset
    range: dcat:Dataset
    description: This property links the Catalogue with a Dataset that is part of
      the Catalogue.
    min: 1
    max: n

# rest of the section is omitted in this example #
```

Figure 6.2: An example of a YAMA document

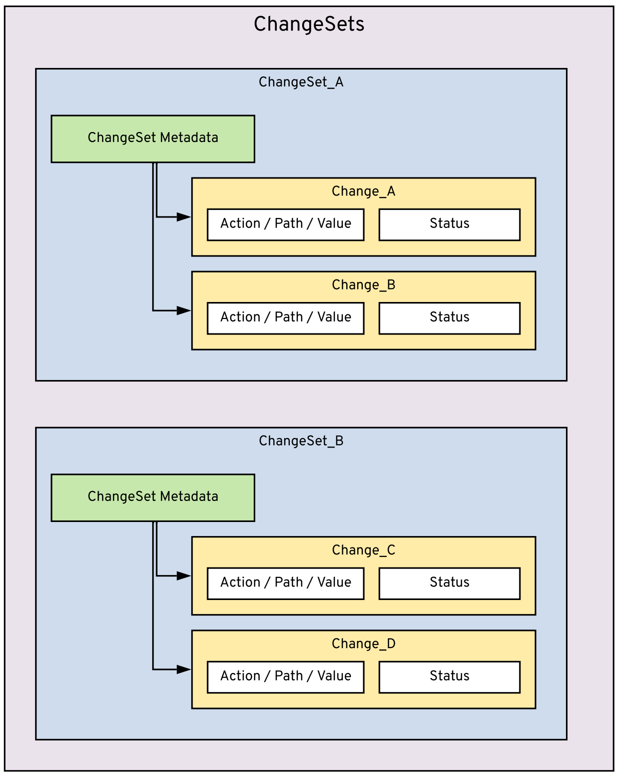


Figure 6.3: Structure of YAMA ChangeSets

6.2 Changesets in YAMA

YAMA also proposes syntax and specifications of Change-Sets. The 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 a modified version or use to record changes of a document over any other versions. Change-Sets is inspired from RFC 6902 JavaScript Object Notation (JSON) Patch [18] , with the changes marked with an action to a path with an additional special reserved value as 'status' to indicate status changes like 'deprecation.' The structure of YAMA changeset is expressed in figure 6.3, and an example code snippet for changeset is shown in figure 6.4.

6.3 YAMA 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 enhances the maintainability of MAPs.

```yaml
changesets:
  cs_20181108_01:
    version: 1.2
    previous_version: 1.1
    date: 2018-11-08
    changes:
      ch_20181108_01:
        op: replace # remove, add, replace, copy, test
        path: /statements/pr_type/max
        value: n
        # Old value; applicable for replace, remove and copy
        previous_value: 0
        # extending a change
        notes: |
          UpdatesCardinality: 0..1 -> 0..n
          This property can be repeated in the
          case that multiple licence types
          apply to a licence document.
        URI: dct:type
        label: Recommended property (Licence Document)
        issue: [DCAT-AP-1, "https://github.com/SEMICeu/DCAT-AP/issues/1"]
```

Figure 6.4: Example of YAMA ChangeSet

3. A structured textual format, which fits well with collaborative development environments and version control systems, and makes 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 as a development roadmap and will permit generation of previous versions, actionable or human-friendly change logs as well as formats for metadata crosswalks and migrations.

6.4 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. 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.

6.5 Recommendations and Best Practices

YAMA documents should be versioned to utilize its capabilities on versioning, changesets, and changelogs. Semantic versioning (SemVer) [1] is highly recommended. Also, calendar versioning (CalVer) [2] 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. The 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. Changesets specification strictly adheres to version numbers and a semantic logic on versioning. In semantic versioning approach `MAJOR.MINOR.PATCH` is considered in MAP as patch versions do not break any `MAJOR`, `MINOR` definitions with forward and reverse compatibility. It can be used internally as part of the development and for fixes and corrections related to typos and or less significant changes. However, 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.

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
```

[1]https://semver.org/
[2]https://calver.org/

Chapter 7

YAMA Toolkit

A proof of concept toolkit is developed as a part of this research, in order to explain the practicality of the proposed YAMA format. This proof of concept toolkit is not intended to be a full-fledged solution of using YAMA, rather a demonstration of how some of the features proposed in YAMA can be useful. As YAMA is compatible with YAML 1.2, we anticipate that a YAMA document can be completely independent of a specific implementation but a highly adaptable format for any languages or toolkits. YAMA is intended to use or incorporate within existing workflows or workflows which requires highly customised outputs. This toolkit is under continuous development, and enhancements and fixes will be included as the acceptance of the format increases.

7.1 Python module & Command Line Interface

The 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 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 [1] and PyShEx [2] can be used to process the parsed structure to generate or manipulate the formats programmatically.

A command line interface is made available within the Python package. This commandline interface is supposed to be used within congestion of other tools for example linting or formatting the output or included with the git automation. Still this CLI tool is a proof of concept of the actual capability of YAMA format. Some bare minimal functions, such as generating basic formats or rendering using Jinja

[1] https://rdflib.readthedocs.io
[2] https://github.com/hsolbrig/PyShEx

templates. Functionality of this tool will be improved as long as the Python package gets updated.

7.2 Templates and generators

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.

To maintain simplicity, but to ensure extensibility and customization capabilities YAMA Python toolkit can use templates to generate any formats from the parsed structured application profile data. Templates permit to generate virtually any formats without the complexities of dealing with complex libraries. YAMA toolkit uses Jinja2 [3] templates by default. Jinja2 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 [22]. Originally Jinja2 is designed for HTML templates, but it is suitable for all kind of text-based formats. Using Jinja2 templates, not only HTML but 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. An example for a simple Jinja template is given in 7.1.

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.

[3]http://jinja.pocoo.org/

```
<?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>
```

Figure 7.1: A basic Jinja template example

Chapter 8

Related Works

Proposal of YAMA as an extensible authoring format for metadata application profiles, is based on the understanding and analysis of some of the previous and related studies and projects. Some of the important attempts to define application profiles and to help creating application profiles where reviewed as part of this research. These proposals were influenced the idea of YAMA, also we attempted to solve most of the shortcomings of these attempts. YAMA has its own differences and advantageous compared to most of these attempts.

8.1 DCMI Description Set Profiles

The Dublincore Metadata Initiative Description Set Profile describes an information model and XML expression of a Description Set Profile (DSP), based on the DCMI Abstract model. A DSP is a means of describing constraints on a description set. It constrains the resources that may be described by descriptions in the description set, the properties that may be used, and the ways a value surrogate may be given. A DSP can be used as a formal representation of the constraints of a Dublin Core Application Profile, configuration for databases, and metadata editing tools [17].

Even though, DSP does not act as human-readable documentation or definition of vocabularies as well as version control of the metadata instance. A DSP contains only syntactic constraints and need to combine with human-readable information, usage guidelines and version management for using as an application profile. Basic structure of DSP is expressed in figure 8.1.

YAMA built on top of DCDSP, and strictly adhere to any future updates on DCDSP. The basic structure of YAMA is DCDSP and it attempts to extend the DSP to include changes logs, versioning and human readable documentation generation. DCDSP doesn't define any specific authoring environments or formats, but this research is more on defining an extensible authoring format.

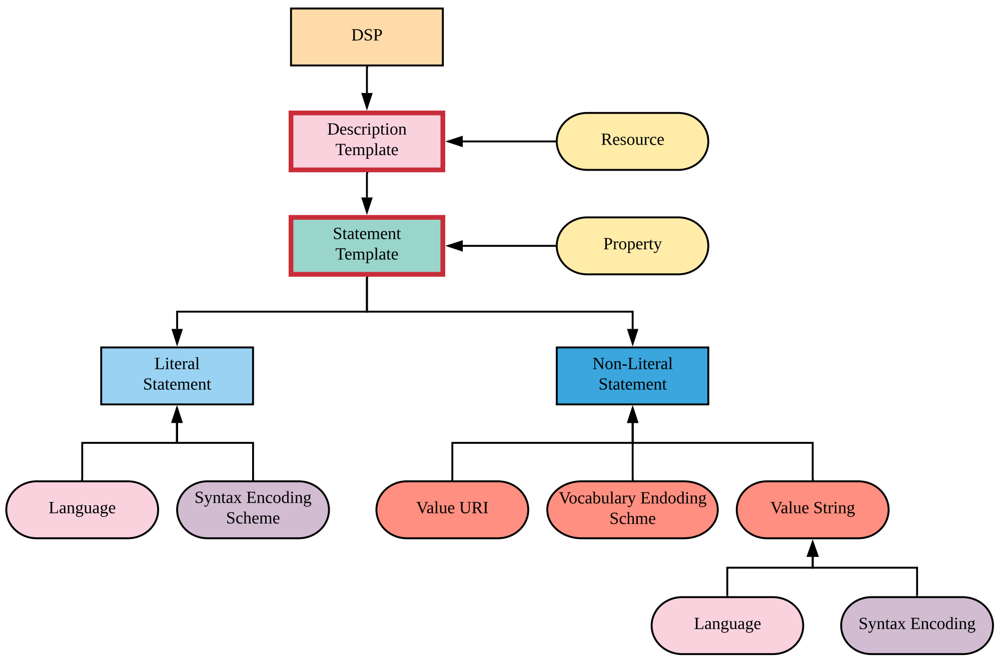


Figure 8.1: Basic structure of DC Description Set Profiles, a constraint language for Dublin Core Application Profiles

8.2 Metabridge Simple DSP (SDSP)

Metabridge [1] is a metadata schema registry to support sharing of metadata schemas and promote reuse of metadata schemas and support metadata interoperability. Metabridge is based on the Singapore Framework of application profiles. Metabrige project defined Simple DSP (SDSP) a simplified DSP authoring format in spreadsheets [2]. Metabridge SDSP can be processed with the metabrigde web service to obtain RDF expression as well as the website can display the application profile in a human friendly manner [15].

YAMA is highly influenced by the simplicity of SDSP and a continuation of attempting more machine actionable and structure format to simplify the authoring of application profiles. This also intended to solve the limitation of a tabular format to express a structured document like application profile. Also YAMA can be parsed without any custom implementation as it is YAML 1.2.

8.3 Me4DCAP

Me4DCAP aims the design of a method for the development of Dublin Core Application Profiles. Me4DCAP can be considered as a starting point to Singapore

[1]https://metabridge.jp
[2]http://www.soumu.go.jp/main_content/000132512.pdf

framework for DCAP and integrates principles from the modes of the metadata community concerning DCAP development and software development processes and techniques. Me4DCAP establishes a way for the development of a DCAP such as, the activities involved in application profile creation, when they should take place, how they interconnect, and which deliverables they will bring about. It also suggests which techniques should be used to build these deliverables [13].

Me4DCAP defines only the methodes involved and deliver a theoretical framework for the application profile creation. The concepts of an extensible authoring format is inspired from the methods from Me4DAP. However, Me4DCAP doesn't resolve the challenges involved in authoring formats or authoring workflows for application profiles.

8.4 IMI Data Model Description (DMD)

Data Model Description (DMD) [3] is a package to explain the data model. DMD is developed as part of IMI(Infrastructure for Multilayer Interoperability) Common Vocabulary Framework to describe common terms and their relationship in order to enhance the interoperability of open data and Digital Government in Japan. DMD is created to share the data model and to unify the data model to ensure data interoperability and utility value. DMD also plays the role of bridging the gap between humans and computers by providing a machine and human readable package of data model definition. A translated diagram [4] to explain the DMD package structure is given in figure 8.2.

DMD is not adhering to Singapore framework; also, it is not a complete application profile. DMD has a well-defined editor and related tools as an authoring environment [5]. The package concept of DMD influences the package concept of YAMA best practices, in delivering Application Profiles as a package with different expression formats.

8.5 Bibframe Profile Editor

Library of Congress Bibframe Profile Editor [6] is a graphical profile editor developed for the specific needs of Bibframe community to modify existing profiles or to create a new profile. Bibframe editor is mainly for the profiles of the BIBFRAME vocabulary, butusable for other profiles as well. See figure 8.3.

[3]https://imi.go.jp/goi/datamodel-about/
[4]https://imi.go.jp/contents/2019/02/DMDSpecification_V301_20190228.pdf
[5]https://imi.go.jp/goi/dmd-editor/
[6]http://bibframe.org/profile-edit/#/profile/list

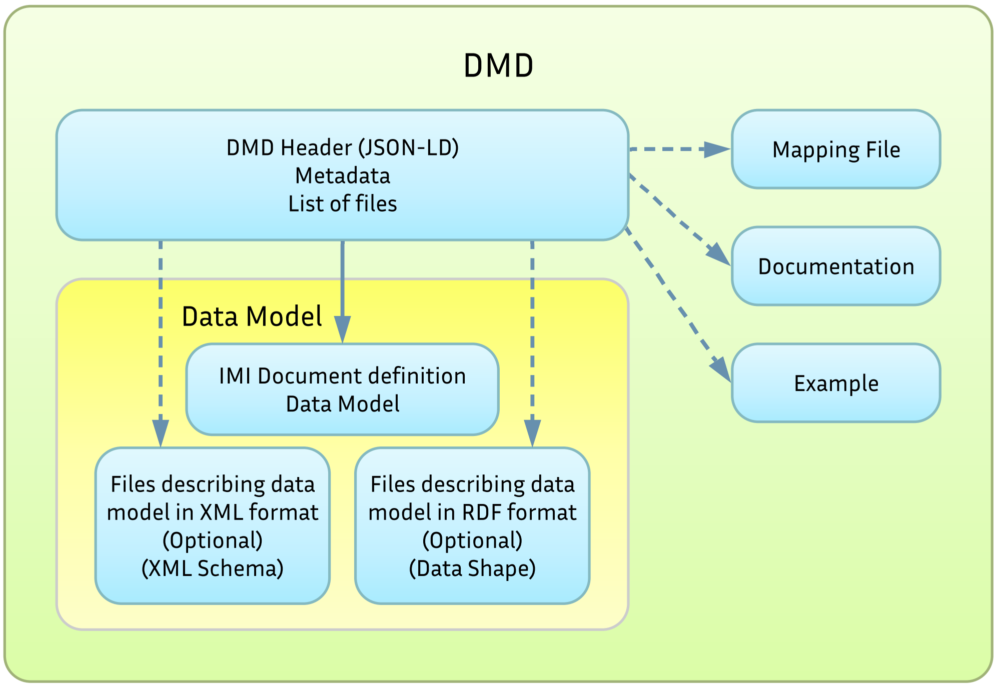
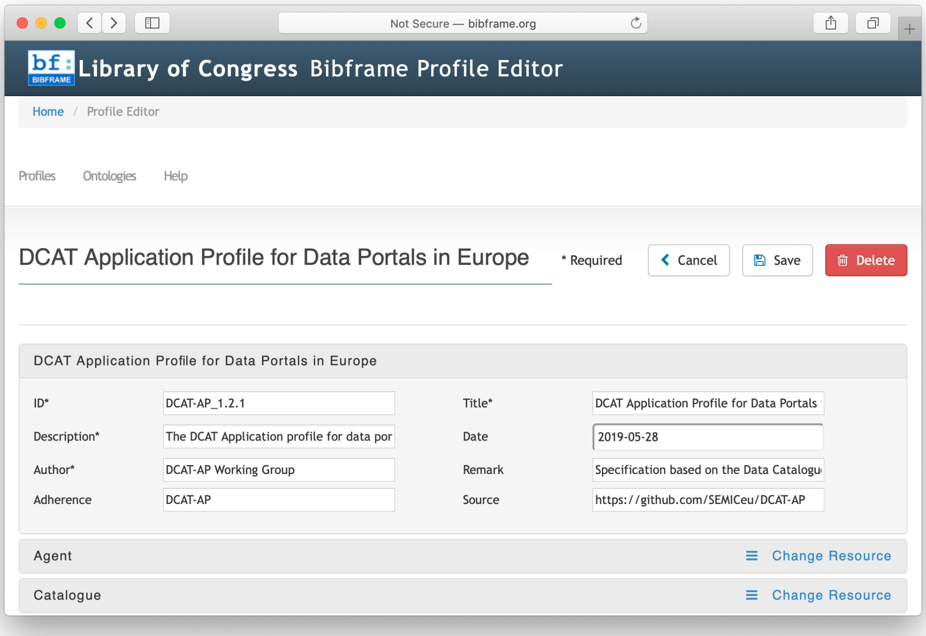

DMD
DMD Header (JSON-LD)
Metadata
List of files
Mapping File
Documentation
Example
Data Model
IMI Document definition
Data Model
Files describing data model in XML format (Optional) (XML Schema)
Files describing data model in RDF format (Optional) (Data Shape)

Figure 8.2: Package structure of DMD (Translated)



DCAT Application Profile for Data Portals in Europe

DCAT Application Profile for Data Portals in Europe

ID*	DCAT-AP_1.2.1	Title*	DCAT Application Profile for Data Portals
Description*	The DCAT Application profile for data por	Date	2019-05-28
Author*	DCAT-AP Working Group	Remark	Specification based on the Data Catalogu
Adherence	DCAT-AP	Source	https://github.com/SEMICeu/DCAT-AP

Agent

Catalogue

Figure 8.3: DCAT-AP in BIBFRAME Profile Editor

Chapter 9

Comparison & Evaluation

To evaluate the advantage of YAMA as an extensible authoring format for application profiles, three different formats were compared. They are Simple DSP [15], Bibframe Editor [3] and YAMA [25]. These three proposals were intended to improve and simplify the process of metadata application profile creation. The three formats use spreadsheets, web interface, and text editor as an authoring environment to express the source. Detailed comparison of these three authoring formats is given in Table 9.1.

One of the major public application profiles, The DCAT Application Profile (DCAT-AP) for data portals in Europe [26] was selected as sample application profiles for the evaluation. The selection is based on its popularity, active maintenance, and the availability of previous versions. DCAT-AP is released as human-readable documentation as well in different machine actionable expression formats such as JSON-LD, RDF and SHACL. Since the source of this application profile is not available, it was assumed that DCAT-AP is created not using any authoring tools but rather manually using word processors and editors for individual formats. To conduct this evaluation, the authors attempted to recreate DCAT-AP using all three authoring formats and used the created source to generate standard application profile expression formats with corresponding tools. This recreation process used human-friendly documentation of the application profiles and tried to include maximum information as the corresponding authoring format permits.

An analysis of the outputs was conducted by comparing the generated expression formats to its originally published versions and documentation. This analysis aimed to identify if an extensible authoring format can express real-world application profiles better than its non-extensible counterparts. The evaluation is performed by comparing the authoring process with these formats, and the extent of information these formats could reproduce from the documentation.

Table 9.1: Feature comparison of three authoring formats used for evaluation.

	SDSP	**Bibframe**	**YAMA**
Based on	DC-DSP	Bibframe	DC-DSP
Format	CSV	JSON	YAML
Standard	RFC-4180	RFC-8259, ECMA-404	YAML 1.2
Strict to standard	No	Yes	Yes
Comments	Yes	No	Yes
Blank lines	Yes	Yes	Yes
Line Diff	Partial	Yes	Yes
Standard library compatibility	Partial	Yes	Yes
Native logic	No	Yes	No
Text editor compatibility	Partial	Yes	Yes
Native Syntax highlighting	No	Yes	Yes
Custom Editor	No	Yes	No
Proposed Editor	Spreadsheet	Bibframe Profile Editor	Text Editor
Schema Validation	No	Possible	Possible
Readability	Low	Low	High
Extensible	No	No	Yes

9.1 Results

Three authoring formats were used in creating a known application profile (DCAT-AP) sources from the documentation and their capability to include the documented application profile features and details were evaluated. The results shown from the investigation can be categorized into a) the capability of including details from the documentation, b) convenience as an authoring format, c) capabilities of producing relevant expression formats.

Three of these formats could represent part of the documentation but mostly the textual explanations and general sections form the documentation is ignored for the convenience, with an assumption that it is manually maintained. All three authoring formats are initially designed for the Dublin Core Application Profile (DCAP), and the application profile evaluated was not entirely Dublin Core Application Profiles. However, for the evaluation, the authors attempted to include maximum information without altering the DCAP structure, wherever possible.

A general observation from the expression capability can be summarized as all three formats could express the essential elements from the documentation and simple DSP could express the minimal information of the AP, and could not express most of the components from the documentation. BibFrame Profile editor could express the full application profile, but semantics including the classes and its statements had to be represented as resources and statements. Also, specific hierarchical structures such as 'mandatory', 'recommended', and 'optional' status of the classification of the statements was skipped and represented the application profile structure more specific to BibFrame. YAMA is extensible due to its straight adaption of YAML, so every section from the application profile documentation

```
classes:
  cl_agent: &cl_agent
    label: Agent
    property: foaf:Agent
    requirement: mandatory
    reference: [http://xmlns.com/foaf/spec/#term_Agent, http://www.w3.org/TR/vocab-org/]
    notes: An entity that is associated with Catalogues and/or Datasets. If the Agent
      is an organisation, the use of the Organization Ontology  is recommended. See
      section 7 for a discussion on Agent roles.
    mandatory_properties:
      - *pr_name
    recommended_properties:
      - *pr_licence_type
  cl_catalogue: &cl_catalogue
    label: Catalogue
    property: dcat:Catalog
    requirement: mandatory
    reference: http://www.w3.org/TR/2013/WD-vocab-dcat-20130312/#class-catalog
    notes: A catalogue or repository that hosts the Datasets being described.
    mandatory_properties:
      - *pr_dataset
      - *pr_description
```

Figure 9.1: Classes from DCAT-AP shows extensibility of YAMA

could be included, and custom names could be used for the keys from the profile's own naming conventions. For an example, part of the document is shown in Figure 9.1.

With respect to the editing process involved, SDSP is found to be relatively easy due to its spreadsheet-based nature which is expressed in CSV, and a spreadsheet editor was used in the process. However, BibFrame Profile Editor is a direct application profile editor with an interactive web-based user interface and advanced suggestion systems to help the profile creation process. Also, BibFrame profile editor supports selecting vocabularies for pre-populating the fields as well as it permits to edit the application profile in a well-structured way. Both in SDSP and YAMA, the structure of the profile is logically built through the syntax specification, but in BibFrame editor the structure is built through visual interaction. A screen shot of the BIBFRAME profile editor is shown in Figure 8.3.

As part of the evaluation, two different sets of use-cases were identified. The first set of use-cases, which are general for an application profile authoring format, and the second set is about extensible authoring formats. All three authoring formats in this study were compared against these use cases. Detailed results for the comparison of the first set is given in Table 9.2, and the comparison for extensible authoring format use cases are in Table 9.3.

Table 9.2: Comparison of general use cases for application profile authoring formats.

Use-cases	**BIBFRAME Profile Editor**	**Simple DSP (SDSP)**	**YAMA**
Acts as a meaningful and actionable authoring tool for MAP.	Yes (Web UI)	Yes (CSV)	Yes (YAML is actionable and human readable.)
A preprocessor format acts as a source for MAP, which enhance the maintainability of MAPs.	Yes (Import and Edit Bibframe JSON)	Yes (CSV)	Yes (YAML is a maintainable format)
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.	Yes (JSON is less human friendly, but structured and maintainable)	Limited (Limitations of tabular format is applicable in version control systems.)	Yes (YAML is structured and suitable with version control systems
A structured authoring format makes it easy to validate the MAP and helps to eliminate errors and logical complexities.	Yes (Bibframe Editor is suitable to limit errors. And JSON files can be validated.)	No (CSV formats has limited support with schema validation.)	Yes (Any YAML schema validator can be used.)
An authoring format 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.	No	No	Yes (ChangSet format permits maintenance of actionable changelogs.)

Table 9.3: Comparison of use cases for extensible MAP authoring formats.

Use-cases	**BIBFRAME Profile Editor**	**Simple DSP (SDSP)**	**YAMA**
A authoring format to generate human friendly documentation.	No	No	Yes (Using templates suitable for required human friendly formats.)
A template driven system, to customize the output formats.	No(But the JSON file can be programmatically parsed to use with templating systems.)	No (Separate logical parser and a templating system are required.)	Yes (YAMA toolkit natively supports Jinja2 template system. As well as the structure of YAMA is optimized for template based output.)
Extensibility of the format to customize it to suit for different community/use-cases.	No(Limited to the scope of BibFrame community.)	Limited (Communities can use it but extensibility is not supported.)	Yes (YAMA permits extending or adding custom key-values in the YAML structure, as per the template requirements)
Inter-operable with various editors/tools	Limited (The output JSON can be edited in any text editors, but intended to work with the web-ui.)	Limited (Editable as a spreadsheet, but difficult to use with text editors.)	Yes (YAML is editable in any text editor and all popular text editors support, highlighting and proper indenting.)
Adherence to Singapore Framework	No (Only basic application profile.)	No (MetaBridge can generate owl format.)	Partial (Templates and generators can cover most of the Singapore Framework aspects.)

Table 9.4: Support for provenance description

Feature	SDSP	BIBFRAME	YAMA
Administrative Information	No	Limited	Yes
Version Details	No	No	Yes
Release Date	No	Yes	Yes
Change Records	No	No	Yes
Change Logs	No	No	Yes

9.2 Versioning, Changelogs and Provenance

Provenance information is vital for application profiles. Maintaining change logs of application profile versions help to assure the longevity of the metadata. The longevity of the schema is essential for metadata longevity. Metadata schema provenance should be documented and maintained for the preservation of metadata [12].Application Profile should provide sufficient administrative information, such as creator, date of release, version, and usage rights. Versioning of the application profile is crucial as it is a record of an application profile as well as metadata changes. Keeping changelogs might help to migrate data-sets to new profiles or create crosswalks to upgrade the instances. For Linked Open Data (LOD), changelogs help to update linked datasets as well.

As an extensible format, YAMA can express version details with custom administrative information of the application profile. Also, YAMA records changes in a machine-actionable way, as well as readable changelogs. From the evaluation, it is clear that YAMA has the advantage of expressing, versioning, administrative details, release dates, change logs as well as actionable change records in the form of changesets. A tabular representation of a comparison of provenance among the tree authoring formats are provided in Table 9.4.

Chapter 10

Discussion

YAMA is developed to be a direct adaption of DC-DSP. It is heavily inspired by the Simple DSP (SDSP) format developed for the MetaBridge project [15]. YAMA is an attempt to promote the acceptance of application profile concepts to various communities with less technical expertise. As a simplified format, it is not free from the limitations of 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 some of the standard libraries to provide 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.

The evaluation suggests that authoring formats can significantly reduce the overall efforts in application profile creation as well as in compelling production of different expression formats. Three different types of authoring formats where evaluated and a considerably larger application profile is recreated using them. Compared to the other two constrained formats, as an extensible authoring tool for application profile, YAMA could express most of the documentation in an actionable manner. Limitations from the other two formats lead in dropping out most of the elements from the documentation and thus, permitting only a part of the human-readable documentation to be recreated. In YAMA, custom key-value pairs were used to include some aspects of the documentation, but in SDSP the structure is constrained so that it cannot be treated as an extensible option to add any custom information other than those specified in the SDSP structure. Similarly, the BIBFRAME profile editor does not permit to interact with the underlying structure and limits the user to follow a specific input process which the GUI is designed for.

The authors could programmatically obtain a structured application profile definition from the YAMA document with native YAML processing libraries. However, SDSP required MetaBridge service to generate RDF expression of the application profile. BIBFRAME editor's output is JSON, and it could be parsed using standard libraries; however, to create different formats, advanced processing of the data is required, and it is possible but tedious.

10.1 Continuation and future plans

YAMA specification is a living standard, and it is getting updated as per the available use-cases. Continues updates and maintenance of YAMA specification as well as improving YAMA toolkit is planned. Activities to develop more examples, templates, and toolkit extensions will be continued.

Sample files created as part of this study is published at:
`http://purl.org/yama/examples`.

Chapter 11

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 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 [1] give application profile more use cases and functions.

The developments in LOD promote data exchange and interoperability of data among communities. Application profiles are the most suitable means for ensuring interoperability and bringing in better use cases for data. Introducing easy to create and maintainable authoring formats will help different communities to adopt application profiles to express their data. There are evolving use cases for application profiles other than just explaining the metadata instance. Extensible authoring formats are expected to provide much more coverage for these emerging use cases, and it will also help communities to generate different formats or type of documentation from a single source which may serve much more purposes than just creating application profiles. The scope of an extensible format is futuristic, and also it can cater to various other demands related to metadata instances. Easy-to-use tools will also promote domain experts to create and publish application profiles in different machine-actionable expressions as well.

[1]https://github.com/dcmi/dcap

Acknowledgement

First and foremost I want to thank my wonderful advisors Tetsuo Sakaguchi and Mitsuharu Nagamori. It has been a great experience to be their student. I appreciate all their contributions of time, ideas, expertise, care, and patience to make my master's program productive and enjoyable and also for granting me unconditional academic and personal freedom. They perfectly shaped my random thoughts into actionable concepts and nurtured them to meaningful research. I am very thankful to the two of them for undertaking this venture of leading me through this thesis.

I am wholeheartedly thankful to Prof. Shigeo Sugimoto for his guidance, helpful comments, informative pointers, and especially for his constructive criticisms.

I express my sincere gratitude to Tom Baker of DCMI for his support, feedbacks, and time, which significantly helped me in improving this research project. I am also indebted to Karen Coyle for her discussions, and ongoing continues efforts to improve DC-DSP, which gave a solid future and direction for this study.

The members of the MDLab have contributed immensely to my personal and academic time at Tsukuba. The group has been a source of friendships as well as sound advice and collaboration. I would like to mention specially and thank both of my tutors, Ryota Kinjo and Yuki Yamanaka for their understanding and helps in the last two years of my student life. I am also thankful to Shohei Toyota for sharing his valuable expertise, which helped a lot in developing some of the fundamental knowledge required for this research. My hearty regards for various helps from Mihara Tetsuya and Chiranthi Wijesundara.

I sincerely thank Prof. Morishima Atsuyuki and other members of the combined lab seminars for their comments and involvements during the presentation sessions.

This acknowledgment won't be complete without mentioning Dr. Beena Cherukuth for being my mentor over a decade and her influences in developing keen academic interests in Library and Information Science. My awesome dear people - Vijoy, Jollyettan, Rahman, Rajenish, Vava and my mother - for being at my side. In the absence of them, I would not be where I am today.

I dedicate this thesis and the time I spent for this Master's program to my late father - the coolest teacher I ever knew - for always encouraging me to pursue higher education and forced me to believe in 'better late than never.' I love you, you are the rock-star of my life ;).

At last but not least, my personal achievements entitled to this research are credited to my partner Deepa for her love, support, encouragement, companionship, and debates, which helped me to overcome all hardships and finally, made this thesis possible.

Nishad, T R
University of Tsukuba
July 2019

References

[1] Murtha Baca. *Introduction to Metadata*. en-US. InteractiveResource. July 2016. URL: http://www.getty.edu/publications/intrometadata (visited on 04/10/2019).

[2] Oren Ben-Kiki, Clark Evans, and Ingy döt Net. *YAML Ain't Markup Language (YAML™) Version 1.2*. Oct. 2009. URL: https://yaml.org/spec/1.2/spec.html (visited on 04/10/2019).

[3] *BIBFRAME Profile Editor*. 2018. URL: http://bibframe.org/profile-edit/ (visited on 06/14/2019).

[4] Karen Coyle. *RDF-AP*. original-date: 2017-01-12T15:38:41Z. Jan. 2017. URL: https://github.com/kcoyle/RDF-AP (visited on 05/15/2019).

[5] Fredrik Enoksson. *DCMI: A MoinMoin Wiki Syntax for Description Set Profiles*. Oct. 2008. URL: http://www.dublincore.org/specifications/dublin-core/dsp-wiki-syntax/ (visited on 03/10/2019).

[6] John Gruber. *Daring Fireball: Markdown*. URL: https://daringfireball.net/projects/markdown/ (visited on 05/14/2019).

[7] Rachel Heery and Manjula Patel. "Application Profiles: Mixing and Matching Metadata Schemas". In: *Ariadne* 25 (2000). ISSN: 1361-3200. URL: http://www.ariadne.ac.uk/issue/25/app-profiles/ (visited on 04/03/2018).

[8] Pieter Heyvaert et al. "Declarative Rules for Linked Data Generation at Your Fingertips". en. In: *The Semantic Web: ESWC 2018 Satellite Events*. Ed. by Aldo Gangemi et al. Vol. 11155. Cham: Springer International Publishing, 2018, pp. 213–217. DOI: 10.1007/978-3-319-98192-5_40. URL: http://link.springer.com/10.1007/978-3-319-98192-5_40 (visited on 05/17/2019).

[9] Diane Hillmann. *Metadata standards and applications*. publisher: Metadata Management Associates LLC. 2006. URL: http://managemetadata.com/ (visited on 04/25/2019).

[10] OpenAPI Initiative. *OpenAPI Specification*. Aug. 2011. URL: https://swagger.io/specification/ (visited on 05/14/2019).

[11] Richard Jones. *A ReStructuredText Primer Docutils 3.0 documentation*. URL: https://docutils.readthedocs.io/en/sphinx-docs/user/rst/quickstart.html (visited on 05/14/2019).

[12] Chunqiu Li and Shigeo Sugimoto. "Provenance description of metadata application profiles for long-term maintenance of metadata schemas". en. In: *Journal of Documentation* 74.1 (Jan. 2018), pp. 36–61. ISSN: 0022-0418. DOI: 10.1108/JD-03-2017-0042. URL: http://www.emeraldinsight.com/doi/10.1108/JD-03-2017-0042 (visited on 05/17/2019).

[13] Mariana Curado Malta and Ana Alice Baptista. "A Method for the Development of Dublin Core Application Profiles (Me4DCAP V0.2): Detailed Description". en. In: (2013), p. 14.

[14] Mariana Curado Malta and Ana Alice Baptista. "A panoramic view on metadata application profiles of the last decade". en. In: *International Journal of Metadata, Semantics and Ontologies* 9.1 (2014), p. 58. ISSN: 1744-2621, 1744-263X. DOI: 10.1504/IJMSO.2014.059124. URL: http://www.inderscience.com/link.php?id=59124 (visited on 05/17/2019).

[15] Mitsuharu Nagamori et al. "Meta-Bridge: A Development of Metadata Information Infrastructure in Japan". en. In: (2011), p. 6.

[16] Mikael Nilsson. *DCMI: Description Set Profiles: A constraint language for Dublin Core Application Profiles*. Mar. 2008. URL: http://www.dublincore.org/specifications/dublin-core/dc-dsp/ (visited on 03/01/2019).

[17] Mikael Nilsson, Tom Baker, and Pete Johnston. *DCMI: The Singapore Framework for Dublin Core Application Profiles*. Jan. 2008. URL: http://dublincore.org/specifications/dublin-core/singapore-framework/ (visited on 05/17/2019).

[18] Mark Nottingham and Paul Bryan. *JavaScript Object Notation (JSON) Patch*. en. Apr. 2013. URL: https://tools.ietf.org/html/rfc6902 (visited on 03/17/2019).

[19] David Osumi-Sutherland et al. "Dead simple OWL design patterns". In: *Journal of Biomedical Semantics* 8.1 (June 2017), p. 18. ISSN: 2041-1480. DOI: 10.1186/s13326-017-0126-0. URL: https://doi.org/10.1186/s13326-017-0126-0 (visited on 05/17/2019).

[20] Andy Powell et al. *DCMI: DCMI Abstract Model*. 2007. URL: http://www.dublincore.org/specifications/dublin-core/abstract-model/ (visited on 05/01/2019).

[21] Linked Data for Production 2 (LD4P2). *Sinopia Profile Editor*. 2019. URL: https://profile-editor.sinopia.io/ (visited on 06/14/2019).

[22] Armin Ronacher. "Jinja2 Documentation". In: *Welcome to Jinja2—Jinja2 Documentation (2.8-dev)* (2008). URL: http://jinja.pocoo.org/.

[23] L. and R. Verborgh Svensson. *Negotiating Profiles in HTTP*. en. Mar. 2017. URL: https://profilenegotiation.github.io/I-D-Accept--Schema/I-D-accept-schema (visited on 05/17/2019).

[24] Lars G Svensson, Rob Atkinson, and Nicholas J Car. *Content Negotiation by Profile*. Apr. 2019. URL: https://www.w3.org/TR/dx-prof-conneg/ (visited on 05/15/2019).

[25] Nishad Thalhath, Mitsuharu Nagamori, and Tetsuo Sakaguchi. *YAMA: Yet Another Metadata Application Profile*. 2019. URL: https://purl.org/yama/spec/latest (visited on 05/09/2019).

[26] *The DCAT Application profile for data portals in Europe (DCAT-AP)*. original-date: 2017-09-13T07:53:27Z. Apr. 2019. URL: https://github.com/SEMICeu/DCAT-AP (visited on 06/14/2019).

Appendices

Appendix A : YAMA Specification

YAMA: Yet Another Metadata Application Profile

Finding, 20 May

This version: `http://purl.org/yama/spec/latest`
Issue Tracking: `https://github.com/nishad/yama/issues/`
Version: 0.1.4

---

1. YAMA Specification

This document defines textual syntax and specifications for writing YAMA documents in a natural text form. YAMA syntax is based on [YAML 1.2 specification](https://yaml.org/spec/1.2/spec.html). YAMA is parsable with any YAML 1.2 parser, but processing capabilities of YAMA documents are limited to YAMA specific implementations.

2. Status of This Document

This document is a part of the YAMA documentation. This document defines textual syntax and specifications. This is a working draft.

3. Introduction

3.1. Philosophy of YAMA

Yet Another Metadata Application Profile (YAMA) is not defined as a new standard of metadata application profiles; but YAMA is defined as an easy to use preprocessor to create standard metadata application profile formats. YAMA intended to be simple enough that it can be used by domain experts without extensive knowledge on metadata application profiles.

3.2. Syntax Compatibility

Instead of introducing its own syntax, YAMA adapts popular YAML format to avoid reinventing the wheel. Being a well proven data serialization format, YAML is widely accepted and various implementations are available for different programming languages. YAMA intended to get benefited from YAML's readability and human friendliness. As a superset of JSON, YAML is a comfortable choice to express complex structures in a human readable yet machine friendly way.

3.3. Extendability

YAMA is extendable with custom elements and structure. The only restriction is 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.

3.4. Specification Versioning

YAMA specifications adhere to [Semantic Versioning 2.0.0](https://semver.org/spec/v2.0.0.html), where are the MAJOR.MINOR stands for specification versions and PATCH for corrections and changes of the documentation, which doesn't break any implementations.

MAJOR version changes will affect the core specification, and MINOR version changes will be backward compatible and does not affect any previously implemented specifications.

4. Document Structure

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.

```
%YAML 1.2
---
YAMA : 1.0
```

4.1. Metadata of the Application Profile

Metadata section is intended to express basic information of the specific metadata application profile. 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 metadata application profile.

YAMA documents MUST be versioned. [Semantic versioning (SemVer)](https://semver.org/) is highly recommended optionally [calendar versioning (CalVer)](https://calver.org/) 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. See table 1 for list of elements in YAMA application profile metadata.

```
description_set :
  ID                  :           # (R) A unique ID for the Description Set. eg :
  ↪ MyBookCaseDS.
  title               :           # Name of the AP. eg : My Book Case Application
  ↪ Profile.
  version             :           # Version following semver.org Semantic
  ↪ Versioning eg. X.Y.Z or X.Y or X.
  date                :           # Release date of AP. Any valid ISO-8601
  ↪ string.
  subject             :           # Subject or topic.
  creator             :           # Person, URL or more contact information. Can
  ↪ also be {name: Person, email: Email, org: Org}
  open                : true      # Open or closed DSP, will be respected in
  ↪ Application Profile curation services. Default true.
  license             :           # License of the DSP default CC ?
  descriptions        : [a,b]     # If given as a list, only those descriptions
  ↪ will be included, else all descriptions with this DSP-ID will be used
```

4.2. Name Spaces

Name spaces are key value pair of prefix and URI. YAMA follows XML specification for prefixes and URI. URI should be as recommended in RFC3986 (https://tools.ietf.org/html/rfc3986). during the document processing, YAMA can generate Qnames from this namespace key values.

Table 1: YAMA Metadata for the Application Profile

Key	Type	Default	Description	Sample	Required
id	Text	-	A unique ID for the Description Set	MyBookCaseDS	R
title	Text	-	Title of the MAP	My Book Case	R
version	Text	-	Version following semver.org Semantic Versioning	X.Y.Z or X.Y or X	R
date	Text	-	Release date of AP. Any valid ISO–8601 string.	2018–12–29	
subject	Text	-	Subject or topic		
creator	Text		Person, URL or more contact information, Free text		Creator can be repeated if there are multiple creators
homepage					
publisher					
keyword					
open	Boolean	true	Open or closed MAP, will be respected in Application Profile curation services.		
license	Free Text	CC	License of the MAP		
descriptions	Array	[a,b]	If given as a list, only those descriptions will be included, else all descriptions with this MAP-ID will be used		

Table 2: YAMA Elements for Descriptions

KEY	TYPE	DEFAULT	DESCRIPTION	RE-QUIRED
label	Text	-	Label of the Element	R
name	Text	Value from label	Human Friendly Name	
min	Intiger	0		
max	Intiger	n		
standalone	Boolean	true		
X class			The class of a description	
description	Text		Short Description	
long_description	Text		Detailed Description	
statements	Array		[a,b] Statements belongs to this description	R

```
namespaces     :
  prefix_1          : uri_1
  prefix_2          : uri_2
```

4.3. Descriptions

Single resources are described under descriptions with a unique ID for each descriptions. Every unique descriptions ID can have multiple key value pairs to describe that resource. See table 2 for list of elements in YAMA description scheme.

```
descriptions    :
  example_description_01:
    label             :          # (R) Label of the Element
    name              :          # Human Friendly Name
    min               :          #
    max               :          #
    standalone        : true     # Default, true
    X class             :          #
    X subclass          :          #
    description       :          # Short Description
    long_description  :          # Detailed Description
    statements        : [a,b]    #
```

4.4. Statements

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. See table 3 for list of elements in YAMA statements.

```
statements     :
  example_statement_01 :
    label             :          # (R) Label of the Element
    name              :          # Human Friendly Name
    min               :          #
    max               :          #
    type              :          #
    description       :          # Short Description
    long_description  :          # Detailed Description
    constraint        : x or [x,y]    # IDs of constraints
```

Table 3: YAMA elements for statements

KEY	TYPE	DEFAULT	DESCRIPTION	REQUIRED
label			[R] Label of the Element	
name			Human Friendly Name	
min				
max				
type				
description			Short Description	
long_description			Detailed Description	
constraint			x or [x,y] IDs of constraints	

Appendix B : Example YAMA Document

```yaml
%YAML 1.2
---
#%YAMA 0.8
meta:
  id: dcat-ap
  title: DCAT Application Profile for Data Portals in Europe
  version: 1.2.1
  date: '2019-05-28'
  subject: Application profile for data portals in Europe (DCAT-AP)
  description: The DCAT Application profile for data portals in Europe (DCAT-AP)
   ↪  is
    a specification based on the Data Catalogue vocabulary (DCAT) for describing
     ↪  public
    sector datasets in Europe. Its basic use case is to enable a cross-data
     ↪  portal
    search for data sets and make public sector data better searchable across
     ↪  borders
    and sectors. This can be achieved by the exchange of descriptions of data
     ↪  sets
    among data portals.
  license: ISA Open Metadata Licence v1.1
  license-url: https://joinup.ec.europa.eu/licence/isa-open-metadata-licence-v11
  creator: DCAT-AP Working Group
  website: https://github.com/SEMICeu/DCAT-AP
  logo:
   ↪  https://joinup.ec.europa.eu/sites/default/files/imagecache/community_logo/DCAT_application_p

namespaces:
  adms: http://www.w3.org/ns/adms#
  dcat: http://www.w3.org/ns/dcat#
  dct: http://purl.org/dc/terms/
  foaf: http://xmlns.com/foaf/0.1/
  owl: http://www.w3.org/2002/07/owl#
  rdfs: http://www.w3.org/2000/01/rdf-schema#
  schema: http://schema.org/
  skos: http://www.w3.org/2004/02/skos/core#
  spdx: http://spdx.org/rdf/terms#
  xsd: http://www.w3.org/2001/XMLSchema#
  vcard: http://www.w3.org/2006/vcard/ns#

```

```
constraints:
  voc_iana_media_types: &voc_iana_media_types
    type: vocabulary
    vocabulary_name: IANA Media Types
    URI: http://www.iana.org/assignments/media-types/media-types.xhtml
    notes: ''
  voc_dataset_theme_vocabulary: &voc_dataset_theme_vocabulary
    type: vocabulary
    vocabulary_name: Dataset Theme Vocabulary
    URI: http://publications.europa.eu/resource/dataset/data-theme
    notes: The value to be used for this property is the URI of the vocabulary
     ↪  itself,
      i.e. the concept scheme, not the URIs of the concepts in the vocabulary.
  voc_eu_vocabularies_frequency_named_authority_list:
   ↪  &voc_eu_vocabularies_frequency_named_authority_list
    type: vocabulary
    vocabulary_name: EU Vocabularies Frequency Named Authority List
    URI: http://publications.europa.eu/resource/authority/frequency
    notes: ''
  voc_eu_vocabularies_file_type_named_authority_list:
   ↪  &voc_eu_vocabularies_file_type_named_authority_list
    type: vocabulary
    vocabulary_name: EU Vocabularies File Type Named Authority List
    URI: http://publications.europa.eu/resource/authority/file-type
    notes: ''
  voc_eu_vocabularies_languages_named_authority_list:
   ↪  &voc_eu_vocabularies_languages_named_authority_list
    type: vocabulary
    vocabulary_name: EU Vocabularies Languages Named Authority List
    URI: http://publications.europa.eu/resource/authority/language
    notes: ''
  voc_eu_vocabularies_corporate_bodies_named_authority_list:
   ↪  &voc_eu_vocabularies_corporate_bodies_named_authority_list
    type: vocabulary
    vocabulary_name: EU Vocabularies Corporate bodies Named Authority List
    URI: http://publications.europa.eu/resource/authority/corporate-body
    notes: The Corporate bodies NAL must be used for European institutions and a
     ↪  small
      set of international organisations. In case of other types of
       ↪  organisations,
      national, regional or local vocabularies should be used.
  voc_geonames: &voc_geonames
    type: vocabulary
    vocabulary_name: Geonames
    URI: http://sws.geonames.org/
    notes: The EU Vocabularies Name Authority Lists must be used for continents,
     ↪  countries
      and places that are in those lists; if a particular location is not in one
       ↪  of
      the mentioned Named Authority Lists, Geonames URIsmust be used.
  voc_eu_vocabularies_places_named_authority_list:
   ↪  &voc_eu_vocabularies_places_named_authority_list
    type: vocabulary
    vocabulary_name: EU Vocabularies Places Named Authority List
    URI: http://publications.europa.eu/resource/authority/place/
    notes: The EU Vocabularies Name Authority Lists must be used for continents,
     ↪  countries
      and places that are in those lists; if a particular location is not in one
       ↪  of
      the mentioned Named Authority Lists, Geonames URIsmust be used.
  voc_eu_vocabularies_countries_named_authority_list:
   ↪  &voc_eu_vocabularies_countries_named_authority_list
    type: vocabulary
    vocabulary_name: EU Vocabularies Countries Named Authority List
    URI: http://publications.europa.eu/resource/authority/country
    notes: The EU Vocabularies Name Authority Lists must be used for continents,
     ↪  countries
      and places that are in those lists; if a particular location is not in one
       ↪  of
      the mentioned Named Authority Lists, Geonames URIsmust be used.
  voc_eu_vocabularies_continents_named_authority_list:
   ↪  &voc_eu_vocabularies_continents_named_authority_list
    type: vocabulary
    vocabulary_name: EU Vocabularies Continents Named Authority List
    URI: http://publications.europa.eu/resource/authority/continent/
    notes: The EU Vocabularies Name Authority Lists must be used for continents,
     ↪  countries
      and places that are in those lists; if a particular location is not in one
       ↪  of
      the mentioned Named Authority Lists, Geonames URIsmust be used.
  voc_adms_change_type_vocabulary: &voc_adms_change_type_vocabulary
    type: vocabulary
    vocabulary_name: ADMS change type vocabulary
    URI: http://purl.org/adms/changetype/
    notes: :created, :updated, :deleted
  voc_adms_status_vocabulary: &voc_adms_status_vocabulary
    type: vocabulary
    vocabulary_name: ADMS status vocabulary
    URI: http://purl.org/adms/status/
    notes: The list of terms in the ADMS status vocabulary is included in the
     ↪  ADMS
      specification
  voc_adms_publisher_type_vocabulary: &voc_adms_publisher_type_vocabulary
    type: vocabulary
    vocabulary_name: ADMS publisher type vocabulary
    URI: http://purl.org/adms/publishertype/
    notes: The list of terms in the ADMS publisher type vocabulary is included in
      the ADMS specification
  voc_adms_licence_type_vocabulary: &voc_adms_licence_type_vocabulary
    type: vocabulary
    vocabulary_name: ADMS licence type vocabulary
    URI: http://purl.org/adms/licencetype/
    notes: The list of terms in the ADMS licence type vocabulary is included in
     ↪  the
      ADMS specification

statements:
  pr_dataset: &pr_dataset
    label: dataset
    property: dcat:dataset
    range: dcat:Dataset
    description: This property links the Catalogue with a Dataset that is part of
      the Catalogue.
    min: 1
    max: n
  pr_description: &pr_description
    label: description
    property: dct:description
    range: rdfs:Literal
    description: This property contains a free-text account of the Distribution.
     ↪  This
      property can be repeated for parallel language versions of the description.
    min: 0
    max: n
  pr_publisher: &pr_publisher
    label: publisher
    property: dct:publisher
    range: foaf:Agent
    description: This property refers to an entity (organisation) responsible for
      making the Dataset available.
    min: 0
    max: 1
    constraints: *voc_eu_vocabularies_corporate_bodies_named_authority_list
  pr_title: &pr_title
    label: title
    property: dct:title
    range: rdfs:Literal
    description: This property contains a name of the category scheme. May be
     ↪  repeated
      for different versions of the name
    min: 1
    max: n
  pr_homepage: &pr_homepage
    label: homepage
    property: foaf:homepage
    range: foaf:Document
    description: This property refers to a web page that acts as the main page
     ↪  for
      the Catalogue.
    min: 0
    max: 1
  pr_language: &pr_language
    label: language
    property: dct:language
    range: dct:LinguisticSystem
    description: This property refers to a language used in the Distribution.
     ↪  This
      property can be repeated if the metadata is provided in multiple languages.
    min: 0
    max: n
    constraints: *voc_eu_vocabularies_languages_named_authority_list
  pr_licence: &pr_licence
    label: licence
    property: dct:license
    range: dct:LicenseDocument
    description: This property refers to the licence under which the Distribution
      is made available.
    min: 0
    max: 1
  pr_release_date: &pr_release_date
    label: release date
    property: dct:issued
    range: rdfs:Literal
    description: This property contains the date of formal issuance (e.g.,
     ↪  publication)
      of the Distribution.
    min: 0
    max: 1
  pr_themes: &pr_themes
    label: themes
    property: dcat:themeTaxonomy
    range: skos:ConceptScheme
    description: This property refers to a knowledge organization system used to
     ↪  classify
      the Catalogue's Datasets.
    min: 0
    max: n
    constraints: *voc_dataset_theme_vocabulary
  pr_update_modification_date: &pr_update_modification_date
    label: update/ modification date
    property: dct:modified
    range: rdfs:Literal
    description: This property contains the most recent date on which the
     ↪  Distribution
      was changed or modified.
    min: 0
    max: 1
  pr_has_part: &pr_has_part
    label: has part
    property: dct:hasPart
    range: dcat:Catalog
    description: This property refers to a related Catalogue that is part of the
     ↪  described
      Catalogue
    min: 0
    max: n
  pr_is_part_of: &pr_is_part_of
    label: is part of
    property: dct:isPartOf
    range: dcat:Catalog
    description: This property refers to a related Catalogue in which the
     ↪  described
      Catalogue is physically or logically included.
    min: 0
    max: 1
  pr_record: &pr_record
    label: record
    property: dcat:record
    range: dcat:CatalogRecord
    description: This property refers to a Catalogue Record that is part of the
     ↪  Catalogue
    min: 0
    max: n
  pr_rights: &pr_rights
    label: rights
    property: dct:rights
    range: dct:RightsStatement
    description: This property refers to a statement that specifies rights
      ↪  associated
      with the Distribution.
    min: 0
    max: 1
  pr_spatial_geographic: &pr_spatial_geographic
    label: spatial / geographic
    property: dct:spatial
    range: dct:Location
    description: This property refers to a geographical area covered by the
      ↪  Catalogue.
    min: 0
    max: n
  pr_primary_topic: &pr_primary_topic
    label: primary topic
    property: foaf:primaryTopic
    range: dcat:Dataset
    description: This property links the Catalogue Record to the Dataset
      ↪  described
      in the record.
    min: 1
    max: 1
  pr_application_profile: &pr_application_profile
    label: application profile
    property: dct:conformsTo
    range: rdfs:Resource
    description: This property refers to an Application Profile that the
      ↪  Dataset’s
      metadata conforms to
    min: 0
    max: 1
  pr_change_type: &pr_change_type
    label: change type
    property: adms:status
    range: skos:Concept
    description: This property refers to the type of the latest revision of a
      ↪  Dataset's
      entry in the Catalogue. It MUST take one of the values :created, :updated
        ↪  or
      :deleted depending on whether this latest revision is a result of a
        ↪  creation,
      update or deletion.
    min: 0
    max: 1
  pr_listing_date: &pr_listing_date
    label: listing date
    property: dct:issued
    range: rdfs:Literal
    description: This property contains the date on which the description of the
      ↪  Dataset
      was included in the Catalogue.
    min: 0
    max: 1
  pr_source_metadata: &pr_source_metadata
    label: source metadata
    property: dct:source
    range: dcat:CatalogRecord
    description: This property refers to the original metadata that was used in
     ↪  creating
      metadata for the Dataset
    min: 0
    max: 1
  pr_contact_point: &pr_contact_point
    label: contact point
    property: dcat:contactPoint
    range: vcard:Kind
    description: This property contains contact information that can be used for
     ↪  sending
      comments about the Dataset.
    min: 0
    max: n
  pr_dataset_distribution: &pr_dataset_distribution
    label: dataset distribution
    property: dcat:distribution
    range: dcat:Distribution
    description: This property links the Dataset to an available Distribution.
    min: 0
    max: n
  pr_keyword_tag: &pr_keyword_tag
    label: keyword/ tag
    property: dcat:keyword
    range: rdfs:Literal
    description: This property contains a keyword or tag describing the Dataset.
    min: 0
    max: n
  pr_theme_category: &pr_theme_category
    label: theme/ category
    property: dcat:theme
    range: skos:Concept
    description: This property refers to a category of the Dataset. A Dataset may
      be associated with multiple themes.
    min: 0
    max: n
    constraints: *voc_dataset_theme_vocabulary
  pr_access_rights: &pr_access_rights
    label: access rights
    property: dct:accessRights
    range: dct:RightsStatement
    description: This property refers to information that indicates whether the
     ↪  Dataset
      is open data, has access restrictions or is not public. A controlled
       ↪  vocabulary
      with three members (:public, :restricted, :non-public) will be created and
       ↪  maintained
      by the Publications Office of the EU.
    min: 0
    max: 1
  pr_conforms_to: &pr_conforms_to
```

```
    label: conforms to
    property: dct:conformsTo
    range: dct:Standard
    description: This property refers to an implementing rule or other
     ↪  specification.
    min: 0
    max: n
  pr_documentation: &pr_documentation
    label: documentation
    property: foaf:page
    range: foaf:Document
    description: This property refers to a page or document about this
     ↪  Distribution.
    min: 0
    max: n
  pr_frequency: &pr_frequency
    label: frequency
    property: dct:accrualPeriodicity
    range: dct:Frequency
    description: This property refers to the frequency at which the Dataset is
     ↪  updated.
    min: 0
    max: 1
    constraints: *voc_eu_vocabularies_frequency_named_authority_list
  pr_has_version: &pr_has_version
    label: has version
    property: dct:hasVersion
    range: dcat:Dataset
    description: This property refers to a related Dataset that is a version,
     ↪  edition,
      or adaptation of the described Dataset.
    min: 0
    max: n
  pr_identifier: &pr_identifier
    label: identifier
    property: dct:identifier
    range: rdfs:Literal
    description: This property contains the main identifier for the Dataset, e.g.
      the URI or other unique identifier in the context of the Catalogue.
    min: 0
    max: n
  pr_is_version_of: &pr_is_version_of
    label: is version of
    property: dct:isVersionOf
    range: dcat:Dataset
    description: This property refers to a related Dataset of which the described
      Dataset is a version, edition, or adaptation.
    min: 0
    max: n
  pr_landing_page: &pr_landing_page
    label: landing page
    property: dcat:landingPage
    range: foaf:Document
    description: This property refers to a web page that provides access to the
     ↪  Dataset,
      its Distributions and/or additional information. It is intended to point to
      a landing page at the original data provider, not to a page on a site of a
       ↪  third
      party, such as an aggregator.
    min: 0
    max: n
  pr_other_identifier: &pr_other_identifier
    label: other identifier
    property: adms:identifier
    range: adms:Identifier
    description: This property refers to a secondary identifier of the Dataset,
     ↪  such
      as MAST/ADS[1], DataCite[2], DOI[3], EZID[4] or W3ID[5].
    min: 0
    max: n
  pr_provenance: &pr_provenance
    label: provenance
    property: dct:provenance
    range: dct:ProvenanceStatement
    description: This property contains a statement about the lineage of a
     ↪  Dataset.
    min: 0
    max: n
  pr_related_resource: &pr_related_resource
    label: related resource
    property: dct:relation
    range: rdfs:Resource
    description: This property refers to a related resource.
    min: 0
    max: n
  pr_sample: &pr_sample
    label: sample
    property: adms:sample
    range: dcat:Distribution
    description: This property refers to a sample distribution of the dataset
    min: 0
    max: n
  pr_source: &pr_source
    label: source
    property: dct:source
    range: dcat:Dataset
    description: This property refers to a related Dataset from which the
     ↪  described
      Dataset is derived.
    min: 0
    max: n
  pr_spatial_geographical_coverage: &pr_spatial_geographical_coverage
    label: spatial/ geographical coverage
    property: dct:spatial
    range: dct:Location
    description: This property refers to a geographic region that is covered by
     ↪  the
      Dataset.
    min: 0
    max: n
    constraints:
      << : *voc_eu_vocabularies_continents_named_authority_list
      URI: hdhakjsdkjhas
  pr_temporal_coverage: &pr_temporal_coverage
    label: temporal coverage
    property: dct:temporal
    range: dct:PeriodOfTime
    description: This property refers to a temporal period that the Dataset
     ↪  covers.
    min: 0
    max: n
  pr_type: &pr_type
    label: type
    property: dct:type
    range: skos:Concept
    description: This property refers to a type of the agent that makes the
     ↪  Catalogue
      or Dataset available
    min: 0
    max: 1
  pr_version: &pr_version
    label: version
    property: owl:versionInfo
    range: rdfs:Literal
    description: This property contains a version number or other version
     ↪  designation
      of the Dataset.
    min: 0
    max: 1
  pr_version_notes: &pr_version_notes
    label: version notes
    property: adms:versionNotes
    range: rdfs:Literal
    description: This property contains a description of the differences between
     ↪  this
      version and a previous version of the Dataset. This property can be
       ↪  repeated
      for parallel language versions of the version notes.
    min: 0
    max: n
  pr_access_url: &pr_access_url
    label: access URL
    property: dcat:accessURL
    range: rdfs:Resource
    description: This property contains a URL that gives access to a Distribution
      of the Dataset. The resource at the access URL may contain information
       ↪  about
      how to get the Dataset.
    min: 1
    max: n
  pr_format: &pr_format
    label: format
    property: dct:format
    range: dct:MediaTypeOrExtent
    description: This property refers to the file format of the Distribution.
    min: 0
    max: 1
    constraints: *voc_eu_vocabularies_file_type_named_authority_list
```

```
pr_byte_size: &pr_byte_size
  label: byte size
  property: dcat:byteSize
  range: rdfs:Literal
  description: This property contains the size of a Distribution in bytes.
  min: 0
  max: 1
pr_checksum: &pr_checksum
  label: checksum
  property: spdx:checksum
  range: spdx:Checksum
  description: This property provides a mechanism that can be used to verify
   ↪  that
    the contents of a distribution have not changed
  min: 0
  max: 1
pr_download_url: &pr_download_url
  label: download URL
  property: dcat:downloadURL
  range: rdfs:Resource
  description: This property contains a URL that is a direct link to a
   ↪  downloadable
    file in a given format.
  min: 0
  max: n
pr_linked_schemas: &pr_linked_schemas
  label: linked schemas
  property: dct:conformsTo
  range: dct:Standard
  description: This property refers to an established schema to which the
   ↪  described
    Distribution conforms.
  min: 0
  max: n
pr_media_type: &pr_media_type
  label: media type
  property: dcat:mediaType
  range: dct:MediaTypeOrExtent
  description: This property refers to the media type of the Distribution as
   ↪  defined
    in the official register of media types managed by IANA.
  min: 0
  max: 1
  constraints: *voc_iana_media_types
pr_status: &pr_status
  label: status
  property: adms:status
  range: skos:Concept
  description: This property refers to the maturity of the Distribution
  min: 0
  max: 1
  constraints: *voc_adms_status_vocabulary
pr_name: &pr_name
  label: name
  property: foaf:name
  range: rdfs:Literal
```

```
    description: This property contains a name of the agent. This property can be
      repeated for different versions of the name (e.g. the name in different
       ↪  languages)
    min: 1
    max: n
  pr_preferred_label: &pr_preferred_label
    label: preferred label
    property: skos:prefLabel
    range: rdfs:Literal
    description: This property contains a preferred label of the category. This
     ↪  property
      can be repeated for parallel language versions of the label.
    min: 1
    max: n
  pr_algorithm: &pr_algorithm
    label: algorithm
    property: spdx:algorithm
    range: spdx:checksumAlgorithm_sha1
    description: This property identifies the algorithm used to produce the
     ↪  subject
      Checksum. Currently, SHA-1 is the only supported algorithm. It is
       ↪  anticipated
      that other algorithms will be supported at a later time.
    min: 1
    max: 1
  pr_checksum_value: &pr_checksum_value
    label: checksum value
    property: spdx:checksumValue
    range: rdfs:Literal
    description: This property provides a lower case hexadecimal encoded digest
     ↪  value
      produced using a specific algorithm.
    min: 1
    max: 1
  pr_notation: &pr_notation
    label: notation
    property: skos:notation
    range: rdfs:Literal
    description: This property contains a string that is an identifier in the
     ↪  context
      of the identifier scheme referenced by its datatype.
    min: 0
    max: 1
  pr_licence_type: &pr_licence_type
    label: licence type
    property: dct:type
    range: skos:Concept
    description: This property refers to a type of licence, e.g. indicating
     ↪  ‘public
      domain’ or ‘royalties required’.
    min: 0
    max: n
    constraints: *voc_adms_licence_type_vocabulary
  pr_start_date_time: &pr_start_date_time
    label: start date/time
    property: schema:startDate
    range: rdfs:Literal
    description: This property contains the start of the period
    min: 0
    max: 1
  pr_end_date_time: &pr_end_date_time
    label: end date/time
    property: schema:endDate
    range: rdfs:Literal
    description: This property contains the end of the period
    min: 0
    max: 1

classes:
  cl_agent: &cl_agent
    label: Agent
    property: foaf:Agent
    requirement: mandatory
    reference: http://xmlns.com/foaf/spec/#term_Agent
     ↪  http://www.w3.org/TR/vocab-org/
    notes: An entity that is associated with Catalogues and/or Datasets. If the
     ↪  Agent
      is an organisation, the use of the Organization Ontology  is recommended.
       ↪  See
      section 7 for a discussion on Agent roles.
    mandatory_properties:
      - *pr_name
    recommended_properties:
      - *pr_licence_type
  cl_catalogue: &cl_catalogue
    label: Catalogue
    property: dcat:Catalog
    requirement: mandatory
    reference: http://www.w3.org/TR/2013/WD-vocab-dcat-20130312/#class-catalog
    notes: A catalogue or repository that hosts the Datasets being described.
    mandatory_properties:
      - *pr_dataset
      - *pr_description
      - *pr_publisher
      - *pr_title
    recommended_properties:
      - *pr_homepage
      - *pr_language
      - *pr_licence
      - *pr_listing_date
      - *pr_themes
      - *pr_update_modification_date
    optional_properties:
      - *pr_has_part
      - *pr_is_part_of
      - *pr_record
      - *pr_rights
      - *pr_spatial_geographical_coverage
  cl_dataset: &cl_dataset
    label: Dataset
    property: dcat:Dataset
    requirement: mandatory
    reference: http://www.w3.org/TR/2013/WD-vocab-dcat-20130312/#class-dataset
    notes: A conceptual entity that represents the information published.
    mandatory_properties:
      - *pr_description
      - *pr_title
    recommended_properties:
      - *pr_contact_point
      - *pr_dataset_distribution
      - *pr_keyword_tag
      - *pr_publisher
      - *pr_theme_category
    optional_properties:
      - *pr_other_identifier
      - *pr_sample
      - *pr_version_notes
      - *pr_landing_page
      - *pr_access_rights
      - *pr_frequency
      - *pr_linked_schemas
      - *pr_has_version
      - *pr_is_version_of
      - *pr_identifier
      - *pr_listing_date
      - *pr_language
      - *pr_update_modification_date
      - *pr_provenance
      - *pr_related_resource
      - *pr_source
      - *pr_spatial_geographical_coverage
      - *pr_temporal_coverage
      - *pr_licence_type
      - *pr_documentation
      - *pr_version
  cl_literal: &cl_literal
    label: Literal
    property: rdfs:Literal
    requirement: mandatory
    reference: http://www.w3.org/TR/rdf-concepts/#section-Literals
    notes: A literal value such as a string or integer; Literals may be typed,
     ↪  e.g.
      as a date according to xsd:date. Literals that contain human-readable text
       ↪  have
      an optional language tag as defined by BCP 47 .
  cl_resource: &cl_resource
    label: Resource
    property: rdfs:Resource
    requirement: mandatory
    reference: http://www.w3.org/TR/rdf-schema/#ch_resource
    notes: Anything described by RDF.
  cl_category: &cl_category
    label: Category
    property: skos:Concept
    requirement: recommended
    reference:
     ↪  http://www.w3.org/TR/2013/WD-vocab-dcat-20130312/#class-category-and-category-scheme
    notes: A subject of a Dataset.
    mandatory_properties:
      - *pr_preferred_label
  cl_category_scheme: &cl_category_scheme
    label: Category scheme
    property: skos:ConceptScheme
    requirement: recommended
    reference:
     ↪  http://www.w3.org/TR/2013/WD-vocab-dcat-20130312/#class-category-and-category-scheme
    notes: A concept collection (e.g. controlled vocabulary) in which the
     ↪  Category
      is defined.
    mandatory_properties:
      - *pr_title
  cl_distribution: &cl_distribution
    label: Distribution
    property: dcat:Distribution
    requirement: recommended
    reference:
     ↪  http://www.w3.org/TR/2013/WD-vocab-dcat-20130312/#class-distribution
    notes: A physical embodiment of the Dataset in a particular format.
    mandatory_properties:
      - *pr_access_url
    recommended_properties:
      - *pr_description
      - *pr_format
      - *pr_licence
    optional_properties:
      - *pr_status
      - *pr_byte_size
      - *pr_download_url
      - *pr_media_type
      - *pr_linked_schemas
      - *pr_listing_date
      - *pr_language
      - *pr_update_modification_date
      - *pr_rights
      - *pr_title
      - *pr_documentation
      - *pr_checksum
  cl_licence_document: &cl_licence_document
    label: Licence document
    property: dct:LicenseDocument
    requirement: recommended
    reference:
     ↪  http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=terms#LicenseDocument
    notes: A legal document giving official permission to do something with a
     ↪  resource.
    recommended_properties:
      - *pr_licence_type
  cl_catalogue_record: &cl_catalogue_record
    label: Catalogue Record
    property: dcat:CatalogRecord
    requirement: optional
    reference:
     ↪  http://www.w3.org/TR/2013/WD-vocab-dcat-20130312/#class-catalog-record
    notes: A description of a Dataset's entry in the Catalogue.
    mandatory_properties:
      - *pr_update_modification_date
      - *pr_primary_topic
    recommended_properties:
      - *pr_linked_schemas
      - *pr_status
      - *pr_listing_date
    optional_properties:
      - *pr_description
      - *pr_language
      - *pr_source
      - *pr_title
  cl_checksum: &cl_checksum
    label: Checksum
    property: spdx:Checksum
    requirement: optional
    reference: http://spdx.org/rdf/terms#Checksum
    notes: A value that allows the contents of a file to be authenticated. This
     ↪  class
      allows the results of a variety of checksum and cryptographic message
       ↪  digest
      algorithms to be represented.
    mandatory_properties:
      - *pr_algorithm
      - *pr_checksum_value
  cl_document: &cl_document
    label: Document
    property: foaf:Document
    requirement: optional
    reference: http://xmlns.com/foaf/spec/#term_Document
    notes: A textual resource intended for human consumption that contains
     ↪  information,
      e.g. a web page about a Dataset.
  cl_frequency: &cl_frequency
    label: Frequency
    property: dct:Frequency
    requirement: optional
    reference: http://dublincore.org/documents/dcmi-terms/#terms-Frequency
    notes: A rate at which something recurs, e.g. the publication of a Dataset.
  cl_identifier: &cl_identifier
    label: Identifier
    property: adms:Identifier
    requirement: optional
    reference: http://www.w3.org/TR/vocab-adms/#identifier
    notes: An identifier in a particular context, consisting of the string that
     ↪  is
      the identifier; an optional identifier for the identifier scheme; an
       ↪  optional
      identifier for the version of the identifier scheme; an optional identifier
      for the agency that manages the identifier scheme
    mandatory_properties:
      - *pr_notation
  cl_kind: &cl_kind
    label: Kind
    property: vcard:Kind
    requirement: optional
    reference: http://www.w3.org/TR/2014/NOTE-vcard-rdf-20140522/#d4e181
    notes: A description following the vCard specification, e.g. to provide
     ↪  telephone
      number and e-mail address for a contact point. Note that the class Kind is
       ↪  the
      parent class for the four explicit types of vCards (Individual,
       ↪  Organization,
      Location, Group).
  cl_linguistic_system: &cl_linguistic_system
    label: Linguistic system
    property: dct:LinguisticSystem
    requirement: optional
    reference: http://dublincore.org/documents/dcmi-terms/#terms-LinguisticSystem
    notes: A system of signs, symbols, sounds, gestures, or rules used in
     ↪  communication,
      e.g. a language
  cl_location: &cl_location
    label: Location
    property: dct:Location
    requirement: optional
    reference: http://dublincore.org/documents/dcmi-terms/#terms-Location
    notes: A spatial region or named place. It can be represented using a
     ↪  controlled
      vocabulary or with geographic coordinates. In the latter case, the use of
       ↪  the
      Core Location Vocabulary  is recommended, following the approach described
       ↪  in
      the GeoDCAT-AP specification.
  cl_media_type_or_extent: &cl_media_type_or_extent
    label: Media type or extent
    property: dct:MediaTypeOrExtent
    requirement: optional
    reference:
     ↪  http://dublincore.org/documents/dcmi-terms/#terms-MediaTypeOrExtent
    notes: A media type or extent, e.g. the format of a computer file
  cl_period_of_time: &cl_period_of_time
    label: Period of time
    property: dct:PeriodOfTime
    requirement: optional
    reference: http://dublincore.org/documents/dcmi-terms/#terms-PeriodOfTime
    notes: An interval of time that is named or defined by its start and end
     ↪  dates.
    optional_properties:
      - *pr_start_date_time
      - *pr_end_date_time
  cl_publisher_type: &cl_publisher_type
    label: Publisher type
    property: skos:Concept
    requirement: optional
    reference: http://www.w3.org/TR/vocab-adms/#dcterms-type
    notes: A type of organisation that acts as a publisher
  cl_rights_statement: &cl_rights_statement
    label: Rights statement
    property: dct:RightsStatement
    requirement: optional
    reference: http://dublincore.org/documents/dcmi-terms/#terms-RightsStatement
    notes: A statement about the intellectual property rights (IPR) held in or
    ↪  over
      a resource, a legal document giving official permission to do something
      ↪  with
      a resource, or a statement about access rights.
  cl_standard: &cl_standard
    label: Standard
    property: dct:Standard
    requirement: optional
    reference: http://dublincore.org/documents/dcmi-terms/#terms-Standard
    notes: A standard or other specification to which a Dataset or Distribution
    ↪  conforms
  cl_status: &cl_status
    label: Status
    property: skos:Concept
    requirement: optional
    reference: http://www.w3.org/TR/vocab-adms/#status
    notes: An indication of the maturity of a Distribution or the type of change
    ↪  of
      a Catalogue Record.
  cl_provenance_statement: &cl_provenance_statement
    label: Provenance Statement
    property: dct:ProvenanceStatement
    requirement: optional
    reference:
    ↪  http://dublincore.org/documents/dcmi-terms/#terms-ProvenanceStatement
    notes: A statement of any changes in ownership and custody of a resource
    ↪  since
      its creation that are significant for its authenticity, integrity, and
      ↪  interpretation
```