

Automatic General Metadata Extraction and Mapping in an HDF5 Use-case

Benedikt Heinrichs¹, Nils Preuß², Marius Politze¹, Matthias S. Müller¹ and Peter F. Pelz²

¹*IT Center, RWTH Aachen University, Seffenter Weg 23, Aachen, Germany*

²*Chair of Fluid Systems, Technical University of Darmstadt, Darmstadt, Germany*

Keywords: Research Data Management, Metadata Extraction, Metadata Generation, Metadata Mapping, Linked Data.

Abstract: Extracting interoperable metadata from data entities is not an easy task. A method for this would need to extract non-interoperable metadata values first and then map the extracted metadata to some sensible representation. In the case of HDF5 files, metadata annotation is already an option, making it an easy target for extracting these non-interoperable metadata values. This paper describes a use-case, that utilizes this property to automatically annotate their data. However, the issue arises, that these metadata values are not reusable, due to their missing interoperability, and validatable since they do not follow any defined metadata schema. Therefore, this paper provides a solution for mapping the defined metadata values to interoperable metadata by extracting them first using a general metadata extraction pipeline and then proposing a method for mapping them. This method can receive a number of application profiles and creates interoperable metadata based on the best fit. The method is validated against the introduced use-case and shows promising results for other research domains as well.

1 INTRODUCTION

Research data is a fundamental building block for scientific discovery. Hence, it should be supported by adequate management processes. The field of Research Data Management (RDM) therefore receives more and more attention by researchers and funding agencies. Accordingly, this improvement is driven by adhering to the so-called FAIR principles (Wilkinson et al., 2016) which describe that data and its metadata should be findable, accessible, interoperable and reusable. Metadata is understood as a data record describing the data that serves to make it interpretable. A standardized representation of metadata, furthermore, makes this information machine-readable and interoperable. Especially the creation of metadata produces a major challenge for researchers when they want to adhere to the FAIR principles. Commonly, metadata is only sometimes stored and not easily reusable since no real fitting interopera-

ble schema is defined and the creation of interoperable metadata creates a large overhead. This creates problems when trying to find and reuse research data by looking at the available metadata. A further challenge is that even if metadata is created, there is often no validation which confirms if it adheres to a certain schema. Over the past years several technologies evolved to record metadata. Eventually, the W3C defined RDF as a standardized framework for the exact purpose of describing data. Solutions exist to create schemas and interoperable metadata which can be validated with RDF, but they require a lot of knowledge about ontologies and the semantic web which creates an accessibility problem. Furthermore, creation of these schemas and metadata is often a tedious manual process which is not realistic in a scaling environment. Since there is work being performed to ease the creation of these schemas as so-called application profiles as described by (Grönewald et al., 2021), this paper aims to remove the overhead knowledge necessary for creating the interoperable metadata and utilize these application profiles. Furthermore, the manual nature of metadata creation is put into question and an automatic model is utilized which creates the needed metadata. The approach is assumed to be usable across different research domains and is val-

^a  <https://orcid.org/0000-0003-3309-5985>

^b  <https://orcid.org/0000-0002-6793-8533>

^c  <https://orcid.org/0000-0003-3175-0659>

^d  <https://orcid.org/0000-0003-2545-5258>

^e  <https://orcid.org/0000-0002-0195-627X>

idated on a real-life use-case which deals with the mentioned issues and is described in section 2.

2 USE-CASE

In this use-case, a test rig is utilized for the experimental validation of an efficiency model of positive displacement pumps. The test rig meets the requirements of ISO 4409 that specifies the measurement procedure, the testing methods and the experimental setup of pump efficiency measurements. The setup comprises the following instrumentation: At the inlet and outlet of the pump under investigation, the pressure and temperature are measured by means of a pressure transmitter based on a piezo resistive silicon sensor and a Pt100 resistance thermometer, respectively. Between the electric motor and the pump, a torque meter measures the torque momentum and the rotating speed based on strain gauge technology. Furthermore, a volume flow meter on the high-pressure side of the pump measures the volume flow by means of the screw pump principle. Figure 1 shows a schematic of the pump test rig.

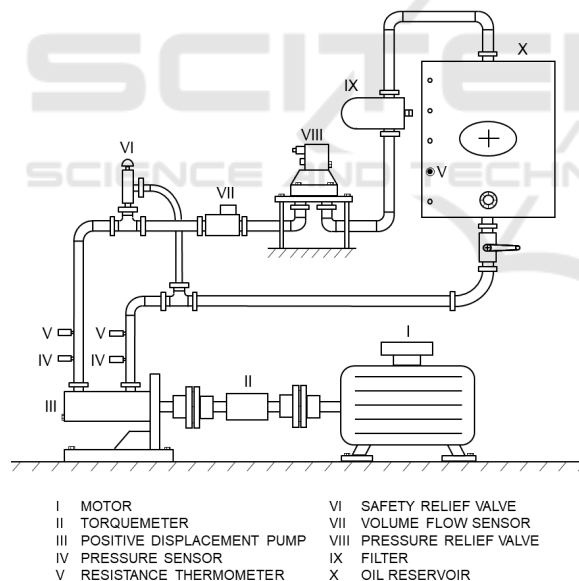


Figure 1: Positive displacement pump test rig utilized for validation of efficiency models.

In the scope of measurements conducted on the introduced test rig, the unit under test, as well as the hydraulic oil, are replaced many times. With these components, the resulting correlation of measured quantities and system state quantities changes as well. For example, the material properties of the hydraulic oil, viscosity and density, depend on the temperature in

the hydraulic system and therefore the operating point of the investigated pump. Similarly, characteristics of the investigated pump are computed dependent on geometric dimensions or other parameters in conjunction with measured quantities. Additionally, measurement equipment needs to be replaced depending on their measuring range in contrast to the range of operating points to be investigated.

It is therefore imperative to organize test rig data of the use-case as self-documenting data products: Complete data packages that contain raw data interlinked with derived result data as well as corresponding metadata, providing all the information needed for the interpretation of the data contents. Section 4.4 describes the data model and file structure utilized to achieve this. Due to the need for frequent adjustments of the test rig setup, this use-case illustrates the need for machine-readable metadata, that is interoperable and applicable across multiple heterogeneous experiment setups. The heterogeneity of data and metadata proliferates further with evolving setup, or when considering a larger scope of multiple test rigs. Therefore, methods and technologies are needed to support the creation of semantically correct metadata which can be validated.

3 CURRENT STATE AND RESEARCH GOAL

The challenge of creating, extracting and mapping metadata is not a new one and has been looked into from different angles. This section therefore describes the current state of the created methods and other use-cases to paint a clearer picture of the identified research gap.

3.1 Metadata Representation

Metadata, in this context, is information that states some fact about a specific (digital) resource. These facts can be used by machines or people to understand the actual dataset. The Resource Description Framework (RDF), described by (Wood et al., 2014), is one of the data models conceived by the W3C for the representation of such information on the web. It originates from the context of the Semantic Web and provides a well-structured way to express metadata in the form of triples (subject-predicate-object) which form a graph. For this reason, RDF offers a reasonable way to map metadata: A resource (subject) is assigned a term or a value (object) within a specific category (predicate). It makes with that, furthermore, a great candidate for describing vocabular-

ies and ontologies with the help of the Web Ontology Language (OWL) described by (W3C, 2012). Additionally, since a need for validated metadata was described, the Shapes Constraint Language (SHACL) is an ideal candidate and was described by (Kontokostas and Knublauch, 2017). By describing the requirements for a given metadataset it can be used as an implementation of metadata schemas as so-called application profiles.

3.2 Metadata Annotation

Manual metadata annotation based on some kind of schematic is a common reality. Platforms like MetaStore described by (Prabhune et al., 2018) and Coscine described by (Politze et al., 2020) allow users to manually annotate their datasets with metadata using user or machine operable interfaces. Both platforms make use of some kind of schema which defines the structure and requirements for a metadataset. The different schemas used for individual metadata records allow building a semi-structured metadata database. This database, or knowledge graph, then contains metadata from a variety of scientific disciplines as validated subsets. The difference between them is, that MetaStore makes use of JSON and XML schemas while Coscine makes use of the RDF language and SHACL application profiles. While this is a technical hurdle, in the context of metadata, JSON and XML schemas and SHACL are widely equivalent in terms of expressiveness as discussed by (Labra Gayo et al., 2017).

3.3 Metadata Mapping Languages

Since manual metadata annotation might not be feasible and in certain use-cases metadata might already be created, just not in RDF, methods were defined to create a mapping between a specific and standardized representation. R2RML described by (Das et al., 2012) or the extension of it called RML and described by (Ben De Meester et al., 2021) allow the creation of mapping rules for mapping non-interoperable metadata to an interoperable representation. Approaches like the one described by (Iglesias et al., 2020) implement these rules and can automatically interpret and map metadata based on it.

3.4 Ontology Mapping

A familiar field for mapping between concepts is ontology mapping. The idea is that between two different concepts the similarity is determined and it is evaluated how two given ontologies are linked between each other. The researchers in (Harrow et al., 2019)

discuss different methods and the challenges for this field. Since the presented problem and the field of ontology mapping is similar, these kinds of methods can inspire an adoption to this area.

3.5 Direct Metadata Mapping

Moving from defined mapping languages and mapping concepts, software like (AtomGraph, 2019) promises the direct conversion from JSON data to RDF. This kind of conversion maps the given key-value pairs to a static prefix which can be chosen. This however fails to map the data to some ontology for creating some meaning in the generated metadata, but still makes it usable with RDF metadata stores and SPARQL endpoints. Furthermore, in the domain of object-triple mapping libraries, approaches specify their classes with annotations and by that establish a direct mapping as described by (Ledvinka and Kremen, 2020). Additionally, works like the one described in (Verborgh and Taelman, 2020) try to create an abstraction layer for interacting with RDF and produce a mapping between web technologies and RDF and SPARQL. They aim to solve part of the usability problems of the semantic web and achieve that by creating their language called LDflex.

3.6 Direct Domain-based Metadata Mapping

For mapping metadata automatically, several approaches have been pursued, however most of them are very domain-specific. The approach described in (TopQuadrant, 2021) lets users translate their non-interoperable metadata to RDF when the targeted SHACL application profile is provided, however requires their proprietary software to be used and assumes a direct mapping using the RDF term suffix. Furthermore, researchers described in (Souza et al., 2017) a method to automatically map JSON data to domain-dependent vocabularies. This method assumes knowledge of an applicable ontology for the metadata, making the inclusion of a domain-expert necessary. It is aligned to the proposed research goal, however does not consider the use of application profiles for creating interoperable metadata. Lastly, the W3C provides in (W3C, 2021) a list of tools which convert application data to RDF based on specific formats and custom implementations.

3.7 Research Goals

Based on the current state of the art, the following question is being proposed: How can a domain-

independent method be used to convert metadata in domain-dependent formats into RDF based on application profiles without the need of a domain-expert? This question raises the following goals for this paper:

1. Provide a method to extract discipline-specific metadata from HDF5 files
2. Provide an algorithm to map discipline-specific metadata to an application profile

4 APPROACH

Following the proposed research goal, a method has to be created, which can automatically generate interoperable metadata based on some kind of initial application profiles. In the work of (Heinrichs and Politze, 2020) a pipeline was proposed that can deal with the automatic extraction of interoperable metadata, however still leaves the mapping of metadata very open and only has some suggestions for it. Therefore, the aim here is to fill this gap and validate it on the use-case.

4.1 General Pipeline Definition

A visual representation of the proposed general pipeline is shown in figure 2 and contains a couple of steps. The first step is some kind of general extraction to retrieve metadata that is already attached to the given data entity like a creation date or a creator. This step is additionally responsible to extract some possible text values and feed this and the data entity itself to the next step. The next step is some kind of MIME-Type-Based extraction method which can be provided by the executing person and is configurable. This kind of method should create some domain-based metadata and deal with the text extraction for specific formats that the general extraction method does not work with. The final step is the text extraction which is based on the idea, that even if no valid metadata has been created yet, at least some kind of text representation of the data entity might exist which can be translated to some facts and represented in RDF as metadata. Finally, all the combined created metadata is pushed through a mapping and validate step which for now is still very open. The results shall then be stored in some kind of metadata store.

4.2 HDF5 Pipeline Definition

Following the general pipeline definition, it can be defined more concretely in the case of HDF5 files. The general extraction step can be performed by the appli-

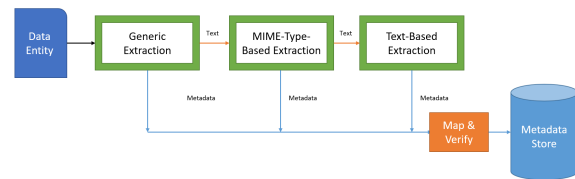


Figure 2: A proposed general metadata extraction pipeline by (Heinrichs and Politze, 2020).

cation Apache Tika described by (Mattmann and Zit-ting, 2011) which supports numerous file types and can extract the attached metadata. The MIME-Type-Based extraction method is here represented by an HDF5 Structure Descriptor which can understand the folder-based structure of an HDF5 file and creates interoperable metadata from this. It furthermore can extract the custom provided non-interoperable metadata from the use-case which will be described in section 4.4. The whole method is described in detail in section 4.3. The mapping step then makes use of provided application profiles and will be discussed more in detail in section 4.5. The resulting metadata is provided to a metadata store.

4.3 HDF5 Structure Descriptor

While the main focus is to derive metadata for a single dataset according to application profiles, HDF5 files provide an internal structure that can be used to break down the single file into multiple datasets and groups. In order to adequately describe the file, it is therefore necessary to find an adequate and interoperable representation of this internal structure in RDF. The vocabulary dcat described by (Perego et al., 2020) can be utilized to describe so-called data catalogs which further can hold some datasets or other data catalogs. This makes it an ideal fit for representing the HDF5 file type, since it can be used to make a direct description of the given structure. The idea is that by describing a group with the property of “dcat:Catalog”, it can be linked to other groups with the term “dcat:catalog”. Different datasets can be described with the property of “dcat:Dataset” and can be referenced by a group with the term “dcat:dataset”.

Furthermore, additional attributes can be extracted from different groups and datasets. Especially the custom specified metadata values are extracted from the attribute key-value pairs and mapped to the respective groups or datasets with a placeholder prefix to label it for a later mapping to an application profile. This additionally synergizes with the later proposed mapping since the custom specified metadata values are all linked to a unique subject representing the specific group or dataset.

4.4 HDF5 Use-case Structure

The general design of the file structure employed in the use-case is the result of the object-oriented modelling method. Relevant entities in the experiment are identified and related to each other, e.g. arbitrary sets of instruments, their configuration and the generated datasets. This structure of objects and their properties is represented as a structure of groups in an HDF5 file, objects are designated by the “CLASS” attribute.

A core example is the class “Pipeline”, extending the concept of I/O-channels in Hardware APIs, which typically represent the configuration of the I/O-hardware per signal as well as the corresponding data. Each “Pipeline” relates a dataset to the set of instruments that it was produced by. Each object of class “Instrument” is in turn described by a set of informational attributes and a model. An object of class “Model” represents the configuration of the software routine that implements the data conversion of the corresponding instrument (e.g. the I/O-Hardware). In the case of a “Pipeline” related to processed data, the “origin” attribute holds a reference (URL or URI) to the “Pipeline” that relates to the raw data. Figure 3 illustrates this data model.

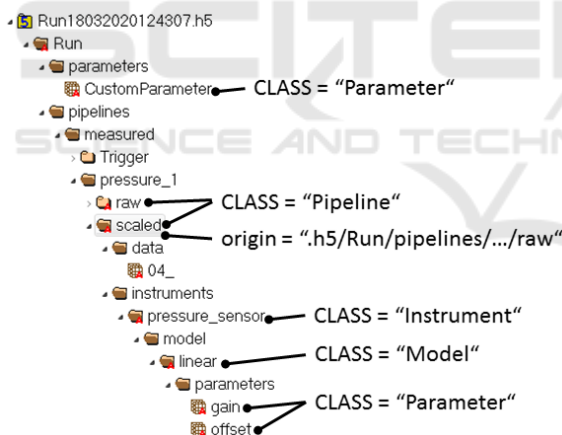


Figure 3: Example of the use-case's class-based structure.

Based on this generic base model, highly diverse experimental setups can be represented. This covers different test rigs of different projects as well as adjustments of one and the same test rig over time. Based on the embedded information, data processing actions corresponding to utilized instruments and their calibration can be configured and re-run.

4.5 Proposed Mapping Algorithm

The proposed mapping algorithm makes use of the previous steps, which include the creation of custom

provided metadata values. After they have been extracted, they have to be mapped to a given number of application profiles. For this, an algorithm is proposed which shall deal with such a mapping. The algorithm is assumed to be used for any range of use-cases and not just the HDF5 use-case, even though it will be validated on it. It receives an application profile as an input and the not yet mapped metadata values. The closeness of each application profile is then determined by the algorithm and the closest output is then chosen to be used for mapping the values.

It is to note that the algorithm can only be as good as the input is. If the input metadata (seen as key-value pairs) is obscured to the point that a mapping is not really possible, or the application profiles are not fitting, then the output will not yield great results. However, the assumption is, if the input is not that obscured and the application profiles contain fitting terms, then the output will yield a good representation.

4.6 Mapping Metadata Values to an Application Profile

The idea to map a metadata value to an application profile is presented in Algorithm 1. The base is a function that receives as input values an application profile as a graph and a metadata value as an RDF triple “t” with a subject “t.sub”, predicate “t.pred” and object “t.obj”. This metadata value is coming from the previous metadata extraction and is in the structure of an identifying subject pointing with the custom mapped attribute key to its value. An example of this would be: `?s custom:CLASS "Pipeline"`. The provided graph can have any number of triples and shall offer some filter functionality. The triples of the application profile graph all have a subject represented as “sub”, a predicate represented as “pred” and an object represented as “obj”. Based on the input, the algorithm then tries to see if the given triple contains the term “class” in the predicate. The assumption is that such a term will indicate that the given triple's object will contain the specific class value, making a mapping to certain classes possible. In the application profile graph, every triple with the “sh:targetClass” property is then iterated through. Note that this iteration has to furthermore support implicit class targets, which are defined by being an instance of “sh:NodeShape” and “rdfs:Class”. If a similar match between the iterated triple class identifier and the triple class identifier can be made, a definition describing that the triple subject is an instance of (“a”) the iterated triple class identifier is returned. This similarity is further being extended by looking

at the label definition of the iterated triple subject and matching this to the triple class identifier. If a similar match can be made, the instance definition is also being returned.

Algorithm 1: An algorithm for mapping metadata values to an application profile.

```

1: function APMAPPING
2:   applicationProfile  $\leftarrow$  RDF as Graph
3:   t  $\leftarrow$  RDF Triple
4:   if t.pred contains “class” then
5:     for tar  $\leftarrow$  Triples with “sh:targetClass”
6:     in applicationProfile do
7:       if t.obj similar to tar.obj then
8:         return (t.sub, a, tar.obj)
9:       for label  $\leftarrow$  Triples with tar.sub
10:      and “rdfs:label” in applicationProfile do
11:        if t.obj similar to label.obj then
12:          return (t.sub, a, tar.obj)
13:     else
14:       for tar  $\leftarrow$  Triples with “sh:path”
15:       in applicationProfile do
16:         if t.pred similar to tar.obj then
17:           return (t.sub, tar.obj, t.obj)
18:         for label  $\leftarrow$  Triples with tar.sub
19:         and “sh:name” in applicationProfile do
20:           if t.pred similar to label.obj then
21:             return (t.sub, tar.obj, t.obj)
22:   return Null

```

If the given triple did not contain the term class, the matching continues by iterating through all property definitions in an application profile and listing every triple with a “sh:path” predicate. For every property definition, the object of “sh:path” is compared to the predicate of the input triple and on similarity, a definition of the input triple subject, “sh:path” object and input triple object is returned since a fitting property has been found. When this is not the case, the “sh:name” definitions of the “sh:path” subject are iterated through and a comparison between the resulted objects and the input triple predicate is made. If a similar match can be made, the previous definition is returned. The final return value, when no similarity could be determined, is null.

This algorithm is built to scale and therefore pre-existing solutions for providing the application profiles can be used as a data-source. Therefore, services like application profile databases could be integrated to provide the data. Furthermore, since application profiles have a user-defined nature which is not possible with ontologies, the mapping can be improved by application profiles which fit into the specific use-

cases (e.g. fitting names for the to be mapped metadata).

5 RESULTS

Following the approach, the proposed pipeline and mapping algorithms were programmed and applied to the described HDF5 use-case. The source code can be found under this URL: <https://git.rwth-aachen.de/coscine/research/semanticmapper>.

5.1 Outcomes

For the mapping, it was important to further define a fitting application profile, since the terms being used in the use-case were quite specialized so that given ontologies did not really fit most of the time. Only sometimes, general terms like “version” could be mapped to ontologies like “schema.org” (defined with <https://schema.org/version>). Therefore, the created application profile followed the in section 4.4 defined use-case class structure and contains a number of definitions which describe the class relations and the properties. In figure 4 part of the class-based structure is shown. The main focus here is the “Pipeline” class. Some properties like “origin”, “units” and “variable” are defined for it and everything is mapped to a custom prefix, since no fitting ontology could be found. Such application profile and the proposed use-case terms could, however, be published in an application profile database, ensuring the interoperability.

Following the definition for the “Pipeline” class, the mapped metadataset utilizing this application profile can be seen in figure 5. The similarity method used here was just a plain string comparison, which returns true if the values are equal and false if they are not. The resulting metadataset describes a group of the HDF5 file which is represented with “dcat:Catalog”, maps the previous defined custom metadata attributes to the metadata properties in the application profile and links to other groups using “dcat:catalog”. Furthermore, the class “Pipeline” is linked to this part of the metadataset using the predicate and object “a ns3:Pipeline” where “ns3” is just the prefix linking to the same base URL as the one defined in the application profile. Lastly, the subject is a unique identifier describing the unique part of the HDF5 file, utilizing a so-called persistent identifier (PID).

It is important to note that these results are also retrieved on a dataset level, where the entry is assigned a “dcat:Dataset” class. For these datasets, further appli-

```

custom:PipelineShape
  a sh:NodeShape ;
  sh:targetClass custom:Pipeline ;
  sh:property [
    sh:path custom:origin ;
    sh:datatype xsd:string ;
    sh:name "Origin" ;
    sh:maxCount 1 ;
    sh:minCount 1 ;
  ] ;
  sh:property [
    sh:path qudt:unit ;
    sh:datatype xsd:string ;
    sh:name "Units" ;
    sh:maxCount 1 ;
    sh:minCount 1 ;
  ] ;
  sh:property [
    sh:path custom:variable ;
    sh:datatype xsd:string ;
    sh:name "Variable" ;
    sh:maxCount 1 ;
    sh:minCount 1 ;
  ] .

```

Figure 4: A part of the application profile representing a “Pipeline” class.

```

hdl:{identifier} a ns3:Pipeline,
  dcat:Catalog ;
  ns5:pipelineVersion "1.0" ;
  ns3:origin "this" ;
  qudt:unit "volts" ;
  ns3:variable "voltage" ;
  dct:identifier "{identifier}" ;
  dcat:catalog hdl:{identifier2},
  hdl:{identifier3} .

```

Figure 5: A part of the metadataset which represents a catalog.

cation profiles are defined, which detail the respective properties and classes.

5.2 Discussion

Utilizing the presented mapping approach, the in the use-case described entities could be mapped to application profiles. The resulted metadata is of course only as good as the provided application profiles are and this can be seen in the resulting figures pretty well. Furthermore, the results show that the mapping between application profiles has the advantage that the requirements can be easily described across different ontologies, opening up a larger variability and additionally, because of the customizable nature of application profiles, they can be adjusted or created with the to be mapped metadata in mind. This

can be seen in figure 4 where multiple ontologies are targeted in an application profile and the names are used to fit with the to be mapped metadata. Such a possible combination eliminates part of the need to define a full custom ontology for the use-case. However, there are some downsides since the mapping for now only supports structures, where there is a key-value structure that does not need any further linking. When the values specify an instance, the mapping currently is too limited to represent this structure. As an example, a definition like “qudt:unit” expecting a “qudt:Unit” instance will not be mapped correctly, however a future extension of the application profile mapping could have the potential to tackle such an issue. Lastly, the current dcat structure is defined manually by the HDF5 Structure Descriptor by linking every key-value pair to a specific group or dataset and adding the necessary terms. This, however, could also be defined automatically by the proposed mapping algorithms if the fitting key-value pairs are present. Suppose the HDF5 Structure Descriptor would provide the keys of “identifier”, “dataset”, “catalog” and “class” tied with the values of “Catalog” and “Dataset”, the application profile shapes provided for the dcat ontology would map these entries to their intended representation. These shapes can be found here: <https://github.com/SEMICEu/DCAT-AP>.

6 CONCLUSION

In this paper a method for automatically extracting and mapping metadata values was presented and validated on an HDF5 use-case. The presented method makes use of previous work which created a general pipeline definition for extracting metadata and extended it by giving a concrete implementation of the metadata mapping part. This novel concrete metadata mapping implementation method can receive application profiles and based on them produces interoperable metadata by looking at the closest matches. The results of this approach show promise and validate the mapping’s operationality. Therefore, this approach has implications for generally mapping metadata values when a number of fitting application profiles are present.

7 FUTURE WORK

Following on this work, a couple of open questions are still left which can be answered by some future work. First, this method assumes to be generally usable when given fitting application profiles, however

validating this on a couple of other use-cases with e.g. more detailed application profiles would be interesting to see the limitations and the computational efficiency. Comparing the “FAIRness” of datasets by utilizing or not utilizing this method could additionally show the effectiveness. Furthermore, the current similarity determination is very simple. Therefore, it might be interesting how different similarity measures compare against each other in the approach. Things like checking if a label only contains certain words or going further and determining similarity based on a created word embedding might all be valid options which could improve the method and ease its use. The method could furthermore be extended to additionally utilize RML mapping definitions, this was in this work however a bit out of scope. Lastly, the current implementation and results utilize labels which are fitting based on the defined name in the HDF5 file which could be improved by a custom mapping property.

ACKNOWLEDGEMENTS

The work was partially funded with resources granted by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 432233186 – AIMS.

REFERENCES

- AtomGraph (2019). JSON2RDF. <https://github.com/AtomGraph/JSON2RDF>.
- Ben De Meester, Pieter Heyvaert, and Thomas Delva (2021). RDF Mapping Language (RML).
- Das, S., Sundara, S., and Cyganiak, R. (2012). R2RML: RDB to RDF mapping language. W3C recommendation, W3C. <https://www.w3.org/TR/2012/REC-r2rml-20120927/>.
- Grönewald, M., Mund, P., Bodenbrenner, M., Fuhrmans, M., Heinrichs, B., Müller, M. S., Pelz, P. F., Marius, P., Preuß, N., Schmitt, R. H., and Stäcker, T. (in press 2021). Mit AIMS zu einem Metadatenmanagement 4.0: FAIRe Forschungsdaten benötigen interoperable Metadaten.
- Harrow, I., Balakrishnan, R., Jimenez-Ruiz, E., Jupp, S., Lomax, J., Reed, J., Romacker, M., Senger, C., Splendiani, A., Wilson, J., and Woollard, P. (2019). Ontology mapping for semantically enabled applications. *Drug Discovery Today*, 24(10):2068–2075.
- Heinrichs, B. and Politze, M. (2020). Moving Towards a General Metadata Extraction Solution for Research Data with State-of-the-Art Methods. 12th International Conference on Knowledge Discovery and Information Retrieval.
- Iglesias, E., Jozashoori, S., Chaves-Fraga, D., Collarana, D., and Vidal, M.-E. (2020). SDM-RDFizer. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*.
- Kontokostas, D. and Knublauch, H. (2017). Shapes Constraint Language (SHACL). W3C recommendation, W3C. <https://www.w3.org/TR/2017/REC-shacl-20170720/>.
- Labra Gayo, J., Prud’hommeaux, E., Boneva, I., and Kontokostas, D. (2017). Validating rdf data. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 7:1–328.
- Ledvinka, M. and Kremen, P. (2020). A comparison of object-triple mapping libraries. *Semantic Web*, 11:483–524.
- Mattmann, C. and Zitting, J. (2011). Tika in Action.
- Perego, A., Beltran, A. G., Albertoni, R., Cox, S., Brown-ing, D., and Winstanley, P. (2020). Data Catalog Vocabulary (DCAT) - Version 2. W3C recommendation, W3C. <https://www.w3.org/TR/2020/REC-vocab-dcat-2-20200204/>.
- Politze, M., Claus, F., Brenger, B. D., Yazdi, M. A., Heinrichs, B., and Schwarz, A. (2020). How to Manage IT Resources in Research Projects? Towards a Collaborative Scientific Integration Environment. *European journal of higher education IT*, 1(2020/1):5.
- Prabhune, A., Stotzka, R., Sakharkar, V., Hesser, J., and Gertz, M. (2018). MetaStore: an adaptive metadata management framework for heterogeneous metadata models. *Distributed and Parallel Databases*, 36(1):153–194.
- Souza, D., Freire, F., and Freire, C. (2017). Enhancing JSON to RDF Data Conversion with Entity Type Recognition.
- TopQuadrant (2021). Importing Data using Active Data Shapes. <http://www.datashapes.org/active/import.html#json>.
- Verborgh, R. and Taelman, R. (2020). Ldflex: A read/write linked data abstraction for front-end web developers. In Pan, J. Z., Tamma, V., d’Amato, C., Janowicz, K., Fu, B., Polleres, A., Seneviratne, O., and Kagal, L., editors, *The Semantic Web – ISWC 2020*, pages 193–211, Cham. Springer International Publishing.
- W3C (2012). OWL 2 Web Ontology Language Document Overview (Second Edition). Technical report, W3C. <https://www.w3.org/TR/2012/REC-owl2-overview-20121211/>.
- W3C (2021). ConverterToRdf - W3C Wiki. <https://www.w3.org/wiki/ConverterToRdf>.
- Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., and Evelo, Chris T. ... Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data*, 3:160018.
- Wood, D., Lanthaler, M., and Cyganiak, R. (2014). RDF 1.1 Concepts and Abstract Syntax. W3C recommendation, W3C. <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>.