ing.grid

RESEARCH ARTICLE

From Ontology to Metadata: A Crawler for Script-based Workflows

HOMER: a tool for extraction and re-use of ontology-based metadata in high-performance measurement and computing workflows

Giuseppe Chiapparino [1], Benjamin Farnbacher [1], Nils Hoppe [1], Radoslav Ralev [2], Vasiliki Sdralia [1,3], Christian Stemmer [1]

1. TUM School of Engineering and Design, Department of Engineering Physics and Computation, Chair of Aerodynamics and Fluid Mechanics, Technical University of Munich, Garching, Germany.
2. TUM School of Computation, Information and Technology, Department of Informatics, Technical University of Munich, Garching, Germany.
3. Munich Data Science Institute (MDSI), Technical University of Munich, Garching, Germany.

Date Submitted:
2023-02-01

Date Received:
2023-02-07

Date Accepted:
2024-06-10

Date Published:
2024-07-12

DOI:
doi.org/10.48694/inggrid.3983

Reviewers:
2 anonymous reviewers

Keywords:
Metadata extraction, HPMC, Ontology, Research Data Management

Data availability:
Data can be found here: https://gitlab.lrz.de/nfdi4ing/crawler/-/tree/master/SimpleApplication_PizzaOntology

Software availability:
Software can be found here: doi:10.14459/2022mp1694401

Corresponding Author:
Giuseppe Chiapparino
giuseppe.chiapparino@tum.de

Abstract. The present work introduces HOMER (**H**igh Performance Measurement and Computing tool for **O**ntology-based **M**etadata **E**xtraction and **R**e-use), a python-written metadata crawler that allows to automatically retrieve relevant research metadata from script-based workflows on HPC systems. The tool offers a flexible approach to metadata collection, as the metadata scheme can be read out from an ontology file. Through minimal user input, the crawler can be adapted to the user's needs and easily implemented within the workflow, enabling to retrieve relevant metadata. The obtained information can be further automatically post-processed. For example, strings may be trimmed by regular expressions or numerical values may be averaged. Currently, data can be collected from text-files and HDF5 files, as well as directly hardcoded by the user. However, the tool has been designed in a modular way, so that it allows straightforward extension of the supported file-types, the instruction processing routines and the post-processing operations.

1 Introduction

Nowadays, scientists are called to handle large amount of generated data, store them in repositories and distribute them among other scientists or the scientific community, something that makes it hard for them to keep track of all the relevant information over time and space. This can lead to the generation of a large quantity of forgotten and unused data, the so-called Dark Data [1], [2]. Although every researcher implements some sort of Research Data Management (RDM), either consciously or unconsciously, to avoid the loss of precious information, standardized RDM approaches, such as the FAIR data principles (Findable, Accessible, Interoperable, and Re-usable), have been proposed in order to provide a more structured and potentially efficient solution to these problems. From the beginning of a project, the scientist should have a data-management plan on how the data will be organized, where they will be stored safely and who should be able to access the data and re-use them. In fact, accompanying the complete data-generation process

with a proper data-management plan will have two benefits. On one side, it will be easier to reproduce old research works for future scientists. On the other side, well-documented data will enable effective secondary research.

For that reason, the German Federal Government funded NFDI (Nationale Forschungsdaten-infrastruktur [National Research Data Infrastructure]), to establish an infrastructure on RDM, providing an environment where scientists can develop solutions to research questions and make their findings and innovations sustainable by implementing the FAIR data principles. NFDI4Ing (NFDI für die Ingenieurwissenschaften [NFDI for Engineering] [3]), one of the consortia funded by the NFDI initiative, brings together the engineering communities to develop, standardise and provide methods and services to make engineering research data FAIR.

One major factor of making data FAIR is the implementation of a controlled vocabulary with common terminology. The use of a controlled vocabulary is essential for findability, interoperability, and consequently, the re-use and the establishment of new user models. Most of the research data in the HPMC domain is neither documented nor are metadata sets available, as common terminologies for HPMC in the engineering sector still need to be developed and established within the community. HOMER allows to automatically retrieve relevant research metadata from script-based workflows on HPC systems and therefore supports researchers to collect and publish their research data within a controlled vocabulary using a standardized workflow. Controlled vocabularies, and the relations and restrictions between their terms, are practically implemented through the use of ontologies. An ontology defines a shared conceptualization of a common vocabulary, semantic relations of data and the syntactic as well as the semantic interoperability, including machine-interpretable definitions of basic concepts in the domain and the relations among them. The NFDI4Ing consortium has developed an ontology as a common classification of engineering data in a taxonomic hierarchy with standardized vocabulary and procedures. Metadata4Ing (Metadata for Engineering [4]) aims at providing a thorough framework for the semantic description of research data, with a particular focus on engineering sciences and neighboring disciplines. Metadata4Ing re-uses elements from the existing terminologies and ontologies, such as DCMI Metadata Terms [5]) or the PROV (Provenance Namespace) ontology [6]), whose terms were imported into Metadata4ing. This ontology allows a thorough description of the whole data-generation process (experiment, observation, simulation), covering aspects such as: the object of investigation, all sample and data manipulation procedures, a summary of the data files and the information contained, and all personal and institutional roles. The NFDI4Ing framework entails many working groups called “archetypes”. Among them, the role of archetype DORIS is twofold: on one side, to create a HPMC-(sub)ontology based on Metadata4Ing in order to establish a consistent terminology for computational fluid-dynamics (CFD) workflows in high performance computing (HPC) systems; on the other side, to develop a metadata crawler, presented in this work, for metadata extraction. The expansion to an HPC-sub-ontology is based on modularity and fits in the primary Metadata4Ing classes of method, tool, object of research. The expansion includes suggestions of unambiguous terms for domain-related metadata expressed in classes, object properties (relations) and data properties. These classes have been developed in a community-based approach and represent common methods and tools for workflows in engineering research on HPMC systems. The crawler named HOMER (**H**PMC tool for **O**ntology-based **M**etadata **E**xtraction and **R**e-use) is intended as a RDM tool to automate

the retrieval of metadata and is designed to be used in script-based HPMC applications.

In the field of RDM, many solutions and tools for metadata extraction have been proposed in the recent years. While all of them share with HOMER the same core concept of automating metadata extraction on HPC systems, they implement different approaches and solutions to the problem, introducing a great variety of capabilities.

For example, the RDM system at the University of Huddersfield, iCurate [7], provides a tailored solution to HPMC data with the functionalities of metadata retrieval, departmental archiving, workflow management system and metadata validation and self inferencing. This last functionality requires the metadata to be mapped onto a suitable ontology. iCurate offers support for all aspects of data management, but the actual extraction of metadata is limited to the annotations made by the user in a HPC job file. While this guarantees a non-intrusive integration within the workflow, it doesn't allow the user to retrieve metadata from output files.

Another tool for research-data management is represented by signac [8]. This is a lightweight framework providing all the components to create a searchable and shareable dataspace (a decentralized infrastructure for data sharing and exchange based on commonly agreed principles [9]). The core application is a semi-structured database that allows storing the original files on the file system along with the associated metadata, which is created on-the-fly and saved in human-readable format. The tool also allows for workflow management thanks to the signac-flow application. The framework is implemented in python and is designed to be used in HPC systems. However, in order to perform the metadata annotation, signac requires the user to wrap the original simulation code into a script.

The extraction of metadata from output files is possible with Xtract [10], [11]. In general, this powerful serverless middleware provides an effective, flexible and scalable way to retrieve metadata from very large data lakes (centralized systems storing data in raw format [12]). With Xtract, metadata can be extracted both centrally ("central mode", i.e. fetching the metadata all at once from the different repositories where the data are generated) and at the edge ("edge mode", i.e. extracting and storing the metadata as soon as the data are created and at the location where they are generated). The service implements several different extractors (written in python or bash), which are run dynamically according to the type of file that needs to be crawled. This allows to handle a vast quantity of different data formats typically employed in scientific applications. Machine learning is used to infer the type of file to be crawled, so as to choose the best extractor(s) for that file in the shortest time possible. Finally, the tool is highly portable as it is wrapped in a docker container [13] and can be linked to Globus [14], [15]. Xtract is a powerful service, which is however more suited to data lakes, rather than to be applied within a workflow. Moreover, the information that can be retrieved is somewhat limited and not entirely customizable by the user.

From this point of view, more freedom is given by ExtractIng [16], a generic automated metadata-extraction toolkit. Again suited for HPC systems, ExtractIng is a Java-written standalone tool that needs to be run once simulation outputs have been produced. It is easy to integrate within a workflow and offers both native and parallel implementation of the parsing algorithm, which makes the code scalable for HPC applications. The metadata extraction is based on the metadata scheme provided by EngMeta [17]. The tool is code-independent, in the sense that an external

configuration file allows to adapt the metadata extraction to the specific simulation code and computational environment. While this provides a generic extraction tool, the configuration file needs to be manually written and adapted by the user (even though it only needs to be done once per code).

Another solution is represented by Brown Dog [18], which consists of two services called DAP and DTS. The first provides file conversion, while the second performs extraction and analysis of metadata. Brown Dog aims at leveraging already existing software, libraries and services in order to provide an automated aid in RDM. The implementation of an elasticity module provides for an optimized auto-scale of the two services based on the system demand. Moreover, a tool catalog points the user to the most suitable option for file conversion and metadata extraction. The extracted metadata is returned as a JSON file. However, this operation is performed by passing the data to the web-based service Clowder. This particular aspect might pose some limitations in the use of Brown Dog.

Some of the services that provide metadata extraction might be domain specific. For example, ScienceSearch [19] is a generalized and scalable search infrastructure, which employs machine learning to capture metadata. Information is retrieved not only from regular data, but also from the context and the surroundings artifacts (proposals, publications, file system structures and images) of the data, allowing for an enrichment of the extracted metadata. The service provides a web interface, where users can submit their text queries, and also provides the possibility for the users to give feedback on the collected metadata, so as to improve the search quality. However, the data model is unique to the NCEM dataset, which includes data relevant to the field of electron microscopy.

Within the NFDI environment, Swate [20] is an Excel add-in for the annotation of experimental data and computational workflows developed by the consortium NFDI4Plants. The tool is intended for metadata annotation based on the ontology provided by the user. The use of a spreadsheet environment aims at providing an intuitive and low-friction workflow, principally focusing on wet-lab applications, where the user can annotate work-relevant metadata as the experiment is performed. Hence, in this tool, the work is done manually by the user.

A second tool within the NFDI infrastructure is presented in this paper. Developed within the NFDI4Ing consortium at the Technical University of Munich (TUM), HOMER is a metadata crawler to be integrated in script-based (HPMC) workflows aiming at retrieving metadata that can be attached to the raw data published by researchers. The tool is designed to be flexible and adjustable to the user's needs in its application and easy to implement in potentially any HPMC workflow. This development approach tries to overcome all the shortcomings highlighted for the RDM solutions and tool reviewed in this section, and to allow HOMER to be suitable for a wide range of applications. In fact, the crawler can retrieve metadata from text and binary (HDF5) files, as well as from user's annotations and terminal commands, at any stage of the workflow without interfering with the other processes composing the workflow. The automated extraction of metadata can be performed both in edge as well as in central mode, making the tool suitable for extracting information also from central repositories (such as data lakes). The metadata extraction is based on the ontology schemes chosen by the users. However, the users do not need to strictly adhere to a fixed scheme, but can adjust and customize it according to

their needs. Moreover, although developed primarily keeping engineering sciences as the main use application, HOMER can be employed to retrieve metadata from HPMC workflows applied to a wide variety of research fields. Finally, the tool has been written with a modular structure, so it can easily be developed further to include new features. Hence, HOMER is proposed as a flexible and consistent RDM tool that can be used in a wide variety of applications and fields with limited user inputs in order to easily promote the FAIR principles and enrich the data created by the user.

The code structure is described in section 3, while a simple application is described in section 4. Finally, in section 5, an overview on the future steps in the code development is given.

2 Characterization of the problem

Many numerical applications, such as optimization problems or parametric studies, require the user to solve essentially the same problem with slightly different inputs each time. To make an example in the field of CFD, assume that it is necessary to assess the aerodynamic characteristics of an aircraft wing during different phases of the flight (take-off, climb, cruise and so on). Hence, the user will perform a certain number of simulations employing the same geometry of the wing while varying the freestream conditions (pressure, density, temperature, Reynolds number and so on) provided as input to the simulation. In such a case, especially when the number of simulations to be performed is large, automating the workflow (or parts of it) by means of script-based processes enables an efficient use of the available computing resources. For example, the user could create a script to change the freestream-input parameters as soon as a simulation ends so that the following one with new conditions would start immediately. Together with the data generation, the researcher should also aim at retrieving and storing the relevant metadata for all the computations performed, in order to comply with the FAIR principles and add value to the data gathered. In the wing-study example, the most obvious relevant metadata would be the different input freestream conditions associated to each simulation result, but the user could be also interested in storing information on the specific hardware or software (version of the code, version of the compiler, ID of the computational node, and so on) used for the simulations, for example. The information that needs to be extracted might be scattered across the different files that are usually generated during numerical calculations, such as the input and output files associated with the simulation, as well as files generated by the HPC system. Therefore, the user will have to go through all these files for each simulation and recursively extract and store the metadata. Doing such a job by hand would be certainly time consuming and would look as a viable option only if the number of simulations is very limited. In HPMC applications where hundreds of simulations are performed, this approach would be prohibitive to say the least and, therefore, the use of a dedicated extraction routine would be the preferential and most efficient choice. At this point, for the user it would be a matter of either writing an extraction routine from scratch, which would guarantee a perfect compatibility even for very specific codes but requires time and resources directly spent by the user, or employ an already available tool, at the price of possible overheads to properly implement the tool within the workflow. In the latter case, HOMER would come in handy as a valid support for the researchers. In fact, HOMER is intended to be used, potentially, for any script-based application and is designed to be easily

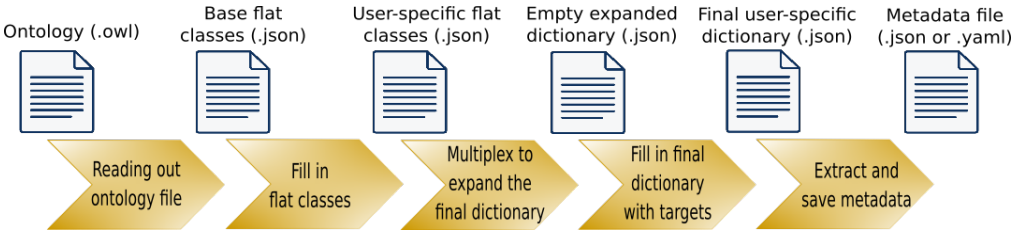

Figure 1: The five steps the crawler is currently composed of and related input/output files.

adjusted to different research fields.

3 Code Description

3.1 Characteristics

HOMER is a code written in python and, at the moment, its complete extraction workflow consists of 5 steps, as shown in figure 1. The code has been developed as a collection of modular routines, each performing a different action, rather than as a single script. This guarantees a flexible application of the crawler, as the user doesn't have always to perform all the steps described in the next subsection, especially once the initial setup of the workflow has been done for the first time. The modularity of the code also allows the developers and the users to easily modify and expand the capabilities of the crawler, so that the code can be tailored for specific applications, if needed. HOMER can be employed both locally and on HPC systems. However, it should be noted that, as of now, it does not support a parallel implementation to parse the target files. Moreover, the tool only covers the stages of planning, creating/collecting and processing/analyzing within the data life cycle (figure 2).

Conceived as a tool to be integrated in script-based workflows, the crawler should be run after the simulation (or, potentially, any processing step), similarly to ExtractIng. Hence, the metadata are naturally extracted in edge mode (where the data are generated). However, the tool can also be used to retrieve metadata from centrally-stored, previously collected data, similarly to Xtract. In this case, though, the user has to perform some extra steps according to the specific case at hand (an example is given at the end of section 4). Together with metadata extraction, the tool gives the user the opportunity to perform some simple post-processing operations as well, such as trimming strings or calculating the minimum, maximum or mean of a series of values.

3.2 Implementation

When running the code for the first time, five steps (figure 1) are needed, with two of them requiring direct user input. The overview of this five-steps workflow is given in the next paragraphs, while an application on a CFD-based example is shown in section 4.

The first step consists in reading the ontology file and creating an empty dictionary containing the flat classes as specified in the ontology. A "flat class" in this context is the initial state of the class in which the properties have not been specified, yet. The step is performed by the routine `ClassUtils.py` and takes advantage of the python package `Owlready2` to work on the ontology. The empty dictionary is a list of the classes and their attributes as they appear in

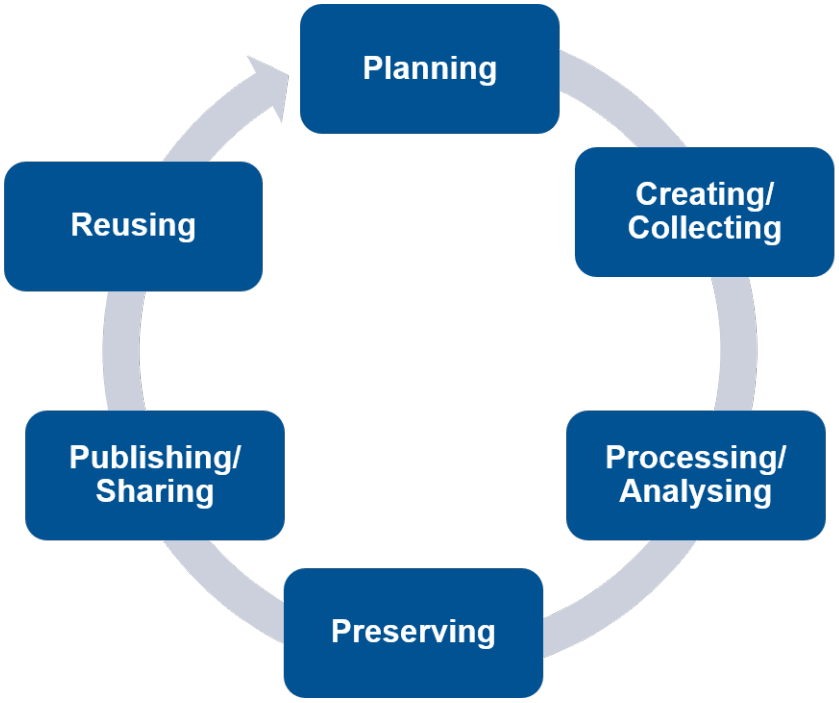

Figure 2: Data Life Cycle [21].

the ontology file. The dictionary acts as a template, where all the flat classes are listed but not filled-in, yet. Hence, no specific instance of a class is created at this point. The file, however, contains the fields that allow the user to specify how many instances and related properties of each class are to be created.

The second step has to be performed manually and serves the purpose of preparing the dictionary file to be used by the "Multiplexer", as explained in the third step. The user needs to specify how many instances of each class need to be retrieved. This is done by giving a numerical value to the keyword `__count__` in the corresponding class. Similarly, the user can specify how many properties each instance of a class needs to have by indicating the numerical value in the corresponding property list.

The third step consists in the Multiplexer and is performed by the routine `Multiplexer.py`. The output is the original flat-class dictionary which has been now expanded according to the needs of the user, so that the new file contains a list of all the needed instances for each class and all the properties for each instance of a class.

The fourth step again has to be performed manually. The user has to fill in the multiplexed dictionary by specifying, for each instance and property, where the crawler should look for the data and how it should retrieve them. This information is to be provided by specifying the three keywords `"path"`, `"type"` and `"pattern"`. The filled expanded dictionary works as a configuration file for the final step.

The fifth and last step consists in the actual extraction of the metadata and is performed through the routine `EntityUtils.py`. Once the metadata have been extracted, it is printed out to a file in a structured human-readable format (JSON or YAML). Currently, metadata can be extracted

from files, such as text files using regular expressions or HDF5 files using `h5py`, from the output of a operating-system command, or can be directly hardcoded during the fourth step, if needed.

As mentioned, the user must first configure the crawler to the specific simulation code in use (type and amount of metadata available to be extracted, location and format of the target files, extraction methods...). This involves the two (lengthy) manual steps just described. However, once the first setup has been completed, the configuration file (created at the end of step four) can be re-used with little to no modification every time a new simulation is performed by the user and new metadata need to be extracted. Specifically, step five simply needs to be performed in the subsequent runs. This allows for a seamless integration of the tool within the workflow.

4 Example Application

In this section, a simple usage of the tool is shown for an application within the CFD field. Namely, the same test problem of a wing aerodynamic optimization mentioned in 2 is considered, in order to show the capabilities of the tool directly applied to an engineering application.

Nonetheless, the reader is also invited to try out the more generic step-by-step test case based on a simplified “Pizza ontology” available in the GitLab code repository (all the relevant files are in the directory `/SimpleApplication_PizzaOntology`). This purposely generic example is intended to provide a complete overview of the implementation of HOMER within a script-based workflow in a simple, clear and application-independent way. The ontology file used in the initial step is loosely based on the “Pizza ontology” provided by Stanford University in their Protégé tutorial [22].

4.1 Application to a CFD case

Taking the case of simulations on an airplane wing at different freestream conditions as an example, the user could be interested in extracting values such as freestream pressure, temperature, Mach and Reynolds number, as well as the ID-number of the node(s) where the calculation was performed and the software version used at the time of the calculation. This metadata information would then be attached to the results of the corresponding simulations to help making the data compliant with the FAIR principles.

In this example, the code NSMB [23] is employed to perform the simulations, and all the relevant pieces of information are extracted from the input and output files of the code and are classified according to a simplified version of the Metadata4Ing ontology for sake of simplicity. Therefore, in this example, the reference classes are limited to `"Processing_Step"`, `"Tool"` and `"Method"`, with information such as parameters name and numerical value being considered as properties of those classes.

In this example, it is assumed that the crawler is employed in edge mode, meaning that the HOMER is invoked right after each simulation has finished to immediately extract the corresponding metadata. This means that all the file paths specified by the keyword `"path"` are relative paths, as the crawler runs from the same directory where the simulations are performed.

The example shows all the five steps described in section 3, which would be ideally required

for the first usage of the tool. After that, HOMER can be seamlessly integrated in the workflow without further modifications.

After setting up the (optional) python virtual environment and having installed the crawler, the first command to run is `ClassUtils.py`, which retrieves the ontology and creates a dictionary with the flat classes (i.e. classes to be filled-out by the user and where, right after step 1, the properties have simply placeholder values) in the form of a `.json` file. The content of such a file would look like the one shown in the next lines.

```
{
    "Processing_Step": {
        "__count__": 1,
        "__restrictions__": "",
        "Name": [1],
        "Parameter": [1],
        "Has_numerical_value": [1],
    },
    "Tool": {
        "__count__": 1,
        "__restrictions__": "",
        "System_component": [1],
        "Name": [1],
        "ID": [1],
    },
    ...
}
```

Listing 1: Example of the content inside the file produced after step 1.

This empty dictionary has to be manually adjusted to the specific case by the user. In this example, only one processing step is foreseen (running the simulation), for which five parameters are going to be extracted (pressure, temperature, Mach, Reynolds and starting time of the simulation). Therefore, the `"Processing_Step"` class will be repeated once (`"__count__": 1`) and will contain one `"Name"` and five `"Parameter"` and `"Has_numerical_value"` properties. Regarding the `"Tool"` class, assume to separate between “Hardware” and “Software”. Hence, two instances of such a class (`"__count__": 2`) need to be created, each of them with its own properties `"System_component"`, `"Name"` and `"ID"`. The result of this manual file manipulation is shown below.

```
{
    "Processing_Step": {
        "__count__": 1,
        "__restrictions__": "",
        "Name": [1],
        "Parameter": [5],
        "Has_numerical_value": [5],
    },
```

```
    "Tool": {
        "__count__":2,
        "__restrictions__": "",
        "System_component": [1,1],
        "Name": [1,1],
        "ID": [1,1],
    },
    ...
}
```

Listing 2: Filled-in .json file after step 2.

Running `Multiplexer.py` expands the classes according to the parameters indicated in the previous step. The output is shown below. The new empty dictionary contains all the instances and corresponding properties the crawler will use in the creation of the metadata file.

```
{
    "Processing_Step": {
        "__restrictions__": "",
        "Name": {
            "path": "",
            "type": "",
            "pattern": "",
            "postprocessor": {
                    "type": "",
                    "args": ""
                }
        },
        "Parameter_1": {
            "path": "",
            "type": "",
            "pattern": "",
            "postprocessor": {
                    "type": "",
                    "args": ""
                }
            }
        },
        ...
        "Has_numerical_value_1": {
            "path": "",
            "type": "",
            "pattern": "",
            "postprocessor": {
                    "type": "",
```

```
                    "args": ""
                }
        },
        ...
    },
    "Tool_1": {
        "__restrictions__": "",
        "Software_component": {
            "path": "",
            "type": "",
            "pattern": "",
            "postprocessor": {
                    "type": "",
                    "args": ""
                }
        },
        ...
    },
    ...
}
```

Listing 3: Dictionary with all the classes and their properties expanded by the multiplexer in step 3.

The multiplexed dictionary has to be filled in manually again by the user. How to fill in the dictionary depends on how the user wants to retrieve the data and where the information is stored. In this example, data are all extracted from plain text files and the crawler uses regular expressions to locate and read the data. This is done by specifying the keywords: `"path"`, `"type"` and `"pattern"`. The entries in `"postprocessor"` can be left empty for the sake of this example. The lines below show how to hardcode metadata by providing a string in `"type"` (for the property `"Parameter_1"`, where the user directly provides the name of the parameter), retrieve information from a file using regular expressions (`"Has_numerical_value_1"`, retrieved from the file specified in `"path"`) and from the output of a terminal command (`"ID"`, where the terminal command is given in `"pattern"`).

```
{
    "Processing_Step": {
        ...
        "Parameter_1": {
            "path": "",
            "type": "string",
            "pattern": "Freestream Mach number",
            "postprocessor": {
                    "type": "",
                    "args": ""
                }
```

```
            }
        },
        ...
        "Has_numerical_value_1": {
            "path": "input.dat",
            "type": "regex",
            "pattern": "Mach :\\s(.*)\\n",
            "postprocessor": {
                    "type": "",
                    "args": ""
                }
        },
        ...
    },
    "Tool_1": {
        ...
        "ID": {
            "path": "",
            "type": "os",
            "pattern": "hostname",
            "postprocessor": {
                    "type": "",
                    "args": ""
                }
        },
        ...
    },
    ...
}
```

Listing 4: Filled-in dictionary in step 4.

Finally, `EntityUtils.py` is used to run the actual extraction routine, which retrieves the metadata according to the parameters specified in the previous step. The output file is shown below and could be either a `.json` or a `.yaml` file, according to the user needs.

```
{
    "Processing_Step": {
        "Name": "Wing simulation",
        "Parameter_1": "Freestream Mach number",
        "Parameter_2": "Freestream pressure",
        "Parameter_3": "Freestream temperature",
        "Parameter_4": "Freestream unit Reynolds number",
        "Parameter_5": "Start time",
        "Has_numerical_value_1": "0.35",
```

```
        "Has_numerical_value_2": "61640",
        "Has_numerical_value_3": "262",
        "Has_numerical_value_4": "5.607E6",
        "Has_numerical_value_5": "10:13:31"
    },
    "Tool_1": {
        "System_component": "Hardware",
        "Name": "lrz-coolmuc2-linux-cluster-2022"
        "ID": "i22r07c05s05"
    },
    "Tool_2": {
        "System_component": "Software",
        "Name": "NSMB",
        "ID": " 6.09.21    Date:  28 - January - 2021     "
    },
    "Method": {
        "Name": "LU-SGS"
    }
}
```

Listing 5: Final .json file containing the extracted metadata after step 5.

At this point, the generated metadata file can be stored together with the output data from the simulation. Whenever the user performs a new simulation and wants to extract the same type of metadata, there is no need to repeat all five steps of the process. In fact, the filled multiplexed dictionary created in step 4 will not change and acts as a configuration file that can be directly re-used in step 5. This means that the user needs only to add the command that runs step 5 in the script-based workflow. This corresponds to using HOMER in edge mode, which means invoking the crawler each time new data is generated at the end of a CFD simulation.

The other option would be to use the crawler after all the simulations have been run in order to retrieve all the metadata at once, which corresponds to using the crawler in central mode. In this case, the user will need to specify absolute file paths (via the keyword `"path"` in the multiplexed `.json` file) to point to the files containing the information. This means that the user needs to create an extra script that allows the crawler to search all the relevant folders and files. Such a script would be specialized according to the user's simulation environment and workflow. Hence, no example of such a usage can be given in the context of the generic CFD-showcase described in this work. However, an example script is provided in the GitLab folder for the Pizza-ontology tutorial. It must be noted that central and edge modes are not features of the tool itself, but are different ways of using HOMER. It's up to the users to decide which is the best usage based on their needs and preferences. This, however, shows again the flexibility of the tool. Another remark is that the user doesn't need to adhere strictly to the chosen ontology file, nor does the user have to use an ontology based on Metada4Ing. At any of the steps where manual input is needed, the user can adjust the classes and properties according to the case-specific needs. For example, the user could rename the property `"Parameter"` to `"Variable"` in the class

`"Processing_Step"` by manually amending the `".json"` file while filling out the flat-class template during step 2. In fact, an ontology file is not even necessary for the actual extraction of the metadata, in principle. The user could even create it's own `.json` file with its own classes and properties skipping the first two steps altogether. On one hand, of course, this approach requires a certain amount of overhead from the researcher side in terms of planning and preparing the `.json` files. On the other hand, it gives much more freedom to the user when it comes to adapt the crawler to the specific case at hand.

Regarding the limitations of HOMER, the tool works best within standardized workflows, where the structure of the files containing the metadata to be extracted changes very little or not at all over time. Although, as shown, it would be possible to adapt the crawler, and in particular the multiplexed dictionary, to new file structures thanks to the flexibility of the tool, such an operation could take a considerable amount of time and effort from the user side if performed for every new application of a (changing) work flow. Hence, it appears sensible to limit the use of the crawler to cases where well-known and relatively fixed data structures are employed as it is common in most numerical and experimental research projects. The second limitation is the range of data formats the crawler can currently extract metadata from, which is limited to text and HDF5 files, together with outputs of terminal commands and hardcoded lines. Although the regular-expression parser allows to retrieve information from virtually any text file regardless of its extension, commonly used formats such as `.xml` have not been implemented, yet. As the crawler is designed flexible, this would be a straight forward process.

5 Conclusion and Future Developments

In this work, HOMER (**H**PMC tool for **O**ntology-based **M**etadata **E**xtraction and **R**e-use), a tool to automate metadata extraction in script-based workflows, has been presented. The crawler, a python-written code, allows for a flexible approach to metadata retrieval. As starting point, the user can provide an ontology file, whose metadata scheme represents the backbone of the extracted information. The classes and attributes from the ontology can be tailored to the specific case at hand and expanded by means of the multiplexer. Once the user has filled in the final dictionary, the actual metadata extraction is executed. This can happen both in edge mode (natural application for script-based workflows) or, with some further user input, in central mode. Then, the extracted metadata can be further post-processed by some routines included in the code. The use of the tool requires some user input and tuning for the first application, but after that, it can be seamlessly integrated in potentially any workflow.

Currently, metadata can be retrieved from text and HDF5 files, from outputs of console commands or can be directly hardcoded in the configuration file. This limitation can be easily overcome in the future, as the code is designed in a modular way, thus allowing for a simple integration of new building blocks. According to the user's needs, new readers/writers of other file formats can be added. The same applies for the post-processing capabilities on the extracted metadata. Moreover, work to increase the amount of readable file formats is planned, at first focusing on the most common formats in CFD applications.

As of now, HOMER can be already implemented in HPMC workflows, so as to enrich each processing step (e.g. mesh generation, simulation, post-processing, report) by adding the cor-

responding metadata. This capability allows for the collection of valuable data (such as the energy consumption for a set of simulations) to enable secondary research and the development of new methodologies in HPC systems. In the current state, the tool would provide the best performance when used to extract metadata right after the creation of the data, as no parallel implementation for file parsing is present, yet. In its five-steps implementation described in this work, HOMER was mainly employed in the data life cycle for the processing stages of planning, creating/collecting and processing/analyzing. A future development of the tool would be to cover the complete data life cycle in a holistic approach, by providing the possibility to automatically preserve and publish/share the extracted metadata along with the research dataset. Through publishing not just the data but also administrative (preservation) metadata, third party users will be able to retrieve crucial information about accessibility, access rights and licenses among others. Bibliographic (author, identifiers) and descriptive (research domain, tools, methods, processing steps) metadata can be published in repositories together with the referenced research data, or be linked to the research data by persistent links and identifiers, if technical or organizational reasons impede a joint provisioning (for example, if the research data are too large to be stored in a common repository).

One main factor of making data FAIR is the use of a controlled vocabulary with common terminology. This is guaranteed by the fact that HOMER supports the usage of semantic ontologies as metadata schemes. These schemes have to be matched somehow with searchable metadata fields in the corresponding repositories. Only few repositories offer such publishing options, like DaRUS (University of Stuttgart) [24], which uses predefined metadata blocks, or Coscine (RWTH) [25], which provides the possibility to use standardized or self-created metadata application profiles [26]. These schemes still have to be parsed with the corresponding metadata fields in the extracted metadata file, to provide the metadata in a standardized, searchable and indexable front end. The NFDI4Ing consortium is simultaneously working on a generic interface which combines different kinds of metadata and data repositories with one standard-based interface. This enables the linking between all data and metadata of the research data life cycle, including experiments, raw data, software, subject-specific metadata sets, and the tracking of usage and citations. Standardized and automatically extracted metadata files can easily be made findable and accessible by this new generic interface [27]. Therefore, HOMER can be a crucial piece within the metadata toolchain from using common vocabularies and automatized extracting to FAIR publishing. The already mentioned Metadata4Ing ontology has been used as the reference during the early stages of the development of HOMER. In the meantime, a HPMC-sub-ontology has been developed within Metadat4Ing. Hence, one of the next steps will be to further adapt HOMER to this new sub-ontology, allowing the tool to be more effective in the complete data life cycle of a CFD workflow on HPC systems.

6 Acknowledgements

The authors would like to thank the Federal Government and the Heads of Government of the Länder, as well as the Joint Science Conference (GWK), for their funding and support within the framework of the NFDI4Ing consortium. Funded by the German Research Foundation (DFG) - project number 442146713. Moreover, the authors gratefully acknowledge the Gauss

Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (www.lrz.de).

7 Roles and contributions

Giuseppe Chiapparino: Conceptualization; Investigation; Methodology; Software - testing; Validation; Writing – original draft

Benjamin Farnbacher: Data curation; Investigation; Writing – original draft (Introduction and Conclusions)

Nils Hoppe: Conceptualization; Investigation; Methodology; Software - development and design

Radoslav Ralev: Software - design, development, implementation and testing

Vasiliki Sdralia: Writing – original draft (Introduction)

Christian Stemmer: Funding acquisition; Resources; Supervision; Writing – review and editing of original

References

[1] P. B. Heidorn, “Shedding Light on the Dark Data in the Long Tail of Science,” *Library Trends*, vol. 57, no. 2, pp. 280–299, 2008. DOI: doi:10.1353/lib.0.0036.

[2] B. Schembera and J. M. Duràn, “Dark Data as the New Challenge for Big Data Science and the Introduction of the Scientific Data Officer,” *Philosophy & Technology*, vol. 33, pp. 93–115, 2020. DOI: https://doi.org/10.1007/s13347-019-00346-x.

[3] NFDI4Ing Consortium. “Website.” (2022), [Online]. Available: https://nfdi4ing.de.

[4] Metadata4Ing Workgroup. “Metadata4ing: An ontology for describing the generation of research data within a scientific activity.” (2022), [Online]. Available: https://nfdi4ing.pages.rwth-aachen.de/metadata4ing/metadata4ing/index.html#ref.

[5] DCMI Usage Board. “Dcmi metadata terms.” (2020), [Online]. Available: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/.

[6] Lebo, Timothy and Satya, Sahoo and Deborah, McGuinness. “Prov-o: The prov ontology.” (2013), [Online]. Available: https://www.w3.org/TR/prov-o/.

[7] S. Liang, V. Holmes, G. Antoniou, and J. Higgins, “Icurate: A research data management system,” in *Multi-disciplinary Trends in Artificial Intelligence*, A. Bikakis and X. Zheng, Eds., Cham: Springer International Publishing, 2015, pp. 39–47, ISBN: 978-3-319-26181-2. DOI: 10.1007/978-3-319-26181-2_4.

[8] C. S. Adorf, P. M. Dodd, V. Ramasubramani, and S. C. Glotzer, “Simple data and workflow management with the signac framework,” *Computational Materials Science*, vol. 146, pp. 220–229, 2018, ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2018.01.035.

[9] L. Nagel and D. Lycklama, “Design principles for data spaces - position paper,” version 1.0, 2021. DOI: 10.5281/zenodo.5105744.

[10] T. J. Skluzacek, “Dredging a data lake: Decentralized metadata extraction,” in *Proceedings of the 20th International Middleware Conference Doctoral Symposium*, ser. Middleware ’19, Davis, California: Association for Computing Machinery, 2019, pp. 51–53, ISBN: 9781450370394. DOI: https://doi.org/10.1145/3366624.3368170.

[11] T. J. Skluzacek, R. Chard, R. Wong, *et al.*, “Serverless workflows for indexing large scientific data,” in *Proceedings of the 5th International Workshop on Serverless Computing*, ser. WOSC ’19, Davis, CA, USA: Association for Computing Machinery, 2019, pp. 43–48, ISBN: 9781450370387. DOI: https://doi.org/10.1145/3366623.3368140.

[12] J. Dixon. “Pentaho, hadoop, and data lakes.” (2010), [Online]. Available: https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/.

[13] D. Merkel, “Docker: Lightweight linux containers for consistent development and deployment,” *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[14] I. Foster, “Globus online: Accelerating and democratizing science through cloud-based services,” *IEEE Internet Computing*, vol. 15, no. 3, pp. 70–73, 2011. DOI: 10.1109/MIC.2011.64.

[15] B. Allen, J. Bresnahan, L. Childers, *et al.*, “Software as a service for data scientists,” *IEEE Internet Computing*, vol. 55, no. 2, pp. 81–88, 2012. DOI: 10.1145/2076450.2076468.

[16] B. Schembera, “Like a rainbow in the dark: Metadata annotation for HPC applications in the age of dark data,” *Journal of Supercomputing*, vol. 77, pp. 8946–8966, 2021. DOI: https://doi.org/10.1007/s11227-020-03602-6.

[17] B. Schembera and D. Iglezakis, “The Genesis of EngMeta - A Metadata Model for Research Data in Computational Engineering,” in *Metadata and Semantic Research*, Cham: Springer International Publishing, 2019, pp. 127–132, ISBN: 978-3-030-14401-2. DOI: https://doi.org/10.1007/978-3-030-14401-2_12.

[18] S. Padhy, G. Jansen, J. Alameda, *et al.*, “Brown dog: Leveraging everything towards autocuration,” in *2015 IEEE International Conference on Big Data (Big Data)*, 2015, pp. 493–500. DOI: 10.1109/BigData.2015.7363791.

[19] G. P. Rodrigo, M. Henderson, G. H. Weber, C. Ophus, K. Antypas, and L. Ramakrishnan, “ScienceSearch: Enabling search through automatic metadata generation,” in *2018 IEEE 14th International Conference on e-Science (e-Science)*, 2018, pp. 93–104. DOI: 10.1109/eScience.2018.00025.

[20] K. Frey, K. Schneider, O. Maus, and T. Mühlhaus. “Swate: A swate workflow annotation tool for excel.” (2022), [Online]. Available: https://github.com/nfdi4plants/Swate.

[21] UK Data Service, modified by TUM University Library (UB). “Data life cycle - icons.” (2022).

[22] M. A. Musen, “The protégé project: A look back and a look forward,” *AI Matters*, vol. 1, no. 4, pp. 4–12, 2015. DOI: 10.1145/2757001.2757003. [Online]. Available: https://doi.org/10.1145/2757001.2757003.

[23] J. Vos, N. Duquesne, and H. J. Lee, “Shock wave boundary layer interaction studies using the NSMB flow solver,” in *3rd European Symposium on Aerothermodynamics for Space Vehicles, ESA SP-426*, 1999.

[24] University of Stuttgart. “Darus.” (2022), [Online]. Available: https://www.izus.uni-stuttgart.de/en/fokus/darus/.

[25] RWTH Aachen University. “Coscine.” (2022), [Online]. Available: https://coscine.rwth-aachen.de.

[26] RWTH Aachen University. “Aims – applying interoperable metadata standards.” (), [Online]. Available: https://www.wzl.rwth-aachen.de/cms/wzl/Forschung/Forschungsumfeld/Forschungsprojekte/Projekte/~ivong/ProMiDigit-Process-Mining-fuer-No-Code/.

[27] NFDI4Ing Consortium. “Metadata hub.” (2022), [Online]. Available: https://git.rwth-aachen.de/nfdi4ing/s-3/s-3-3/metadatahub.