


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

HOMER: a HPMC tool for Ontology-based Metadata Extraction and Re-use

Giuseppe Chiapparino  ¹Benjamin Farnbacher  ¹Nils Hoppe  ¹Radoslav Ralev  ²Vasiliki Sdralia  ³Christian 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. TUM School of Engineering and Design; Department of Engineering Physics and Computation; Chair of Aerodynamics and Fluid Mechanics; Munich Data Science Institute (MDSI), Technical University of Munich, Garching; Germany.

**Date Received:**

2023-01-20

Licenses:This article is licensed under: **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:10.14459/2022mp16944013](https://doi.org/10.14459/2022mp16944013)

Abstract. The present work introduces HOMER (HPMC tool for Ontology-based Metadata Extraction and Re-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

2 Nowadays, scientists are called to handle large amount of generated data, store them in repositories
3 and distribute them among other scientists or the scientific community, something that makes
4 it hard for them to keep track of them over time and space. This can lead to the generation of
5 so-called Dark Data [1], [2], a large quantity of forgotten and unused data. Here comes Research
6 Data Management to provide an efficient solution to these problems. From the beginning of a
7 project, the scientist should have a data-management plan on how the data will be organized,
8 where they will be stored safely and who should be able to access the data and re-use them.

9 In fact, accompanying the complete data-generation process with a proper data-management plan
10 will have two benefits. On one side, it will be easier to reproduce old research works for future
11 scientists. On the other side, well-documented data will enable effective secondary research.

12 For that reason, the German Federal Government funded NFDI (National Research Data Infras-
13 tructure), to establish an infrastructure on research-data management, providing an environment
14 where scientists can develop solutions to research questions and make their findings and innova-
15 tions sustainable by implementing the FAIR data principles: Findable, Accessible, Interoperable,
16 and Re-usable. NFDI4Ing [3], one of the consortia funded by the NFDI-initiative, brings together
17 the engineering communities to develop, standardise and provide methods and services to make
18 engineering research data FAIR.

19 An ontology defines a common vocabulary and describes the syntactic as well as the semantic
20 interoperability, including machine-interpretable definitions of basic concepts in the domain
21 and the relations among them. The NFDI4Ing consortium has developed an ontology as a
22 common classification of engineering data in a taxonomic hierarchy with standardized vocabulary
23 and procedures. Metadata4Ing [4] aims at providing a thorough framework for the semantic
24 description of research data, with a particular focus on engineering sciences and neighboring
25 disciplines. This ontology allows a thorough description of the whole data-generation process
26 (experiment, observation, simulation), embracing the object of investigation, all sample and
27 data manipulation procedures, a summary of the data files and the information contained, and
28 all personal and institutional roles. Within the NFDI4Ing framework, the role of archetype
29 DORIS is twofold: on one side, to create a HPMC-(sub)ontology based on Metadata4Ing in
30 order to establish a consistent terminology for CFD workflows in HPC systems (HPMC = High
31 Performance Measurement and Computing); on the other side, to develop a metadata crawler,
32 presented in this work, for metadata extraction. The crawler named HOMER (**H**PMC tool for
33 **O**ntology-based **M**etadata **E**xtraction and **R**e-use) is intended as a Research-Data Management
34 tool to automate the retrieval of metadata and is designed to be used in script-based HPMC
35 applications.

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

40 For example, the RDM system at the University of Huddersfield, iCurate [5], provides a tailored
41 solution to HPMC data with the functionalities of metadata retrieval, departmental archiving,
42 workflow management system and metadata validation and self inferencing. This last function-
43 ality requires the metadata to be mapped onto a suitable ontology. iCurate offers support for all
44 aspects of data management, but the actual extraction of metadata is limited to the annotations
45 made by the user in a HPC job file. While this guarantees a non-intrusive integration within the
46 workflow, it doesn't allow the user to retrieve metadata from output files.

47 Another tool for research-data management is represented by signac [6]. This is a lightweight
48 framework providing all the components to create a searchable and shareable dataspace. The core
49 application is a semi-structured database that allows storing the original files on the file system
50 along with the associated metadata, which is created on-the-fly and saved in human-readable

51 format. The tool also allows for workflow management thanks to the signac-flow application.
52 The framework is implemented in python and is designed to be used in HPC systems. However, in
53 order to perform the metadata annotation, signac requires the user to wrap the original simulation
54 code into a script.

55 The extraction of metadata from output files is possible with Xtract [7], [8]. In general, this
56 powerful serverless middleware provides an effective, flexible and scalable way to retrieve
57 metadata, both centrally and at the edge, from very large data lakes. The service implements
58 several different extractors (written in python or bash), which are run dynamically according
59 to the type of file that needs to be crawled. This allows to handle a vast quantity of different
60 data formats typically employed in scientific applications. Machine learning is used to infer
61 the type of file to be crawled, so as to choose the best extractor(s) for that file in the shortest
62 time possible. Finally, the tool is highly portable as it is wrapped in a docker container and can
63 be linked to Globus. Xtract is a powerful service, which is however more suited to data lakes,
64 rather than to be applied within a workflow. Moreover, the information that can be retrieved is
65 somewhat limited and not entirely customizable by the user.

66 From this point of view, more freedom is given by ExtractIng [9], a generic automated metadata-
67 extraction toolkit. Again suited for HPC systems, ExtractIng is a Java-written standalone tool
68 that needs to be run once simulation outputs have been produced. It is easy to integrate within a
69 workflow and offers both native and parallel implementation of the parsing algorithm, which
70 makes the code scalable for HPC applications. The metadata extraction is based on the metadata
71 scheme provided by EngMeta [10]. The tool is code-independent, in the sense that an external
72 configuration file allows to adapt the metadata extraction to the specific simulation code and
73 computational environment. While this provides a generic extraction tool, the configuration file
74 needs to be manually written and adapted by the user (even though it only needs to be done once
75 per code).

76 Another solution is represented by Brown Dog [11], which consists of two services called DAP
77 and DTS. The first provides file conversion, while the second performs extraction and analysis
78 of metadata. Brown Dog aims at leveraging already existing software, libraries and services in
79 order to provide an automated aid in RDM. The implementation of an elasticity module provides
80 for an optimized auto-scale of the two services based on the system demand. Moreover, a tool
81 catalog points the user to the most suitable option for file conversion and metadata extraction.
82 The extracted metadata is returned as a JSON file. However, this operation is performed by
83 passing the data to the web-based service Clowder. This particular aspect might pose some
84 limitations in the use of Brown Dog.

85 Some of the services that provide metadata extraction might be domain specific. For example,
86 ScienceSearch [12] is a generalized and scalable search infrastructure, which employs machine
87 learning to capture metadata. Information is retrieved not only from regular data, but also from
88 the context and the surroundings artifacts (proposals, publications, file system structures and
89 images) of the data, allowing for an enrichment of the extracted metadata. The service provides
90 a web interface, where users can submit their text queries, and also provides the possibility
91 for the users to give feedback on the collected metadata, so as to improve the search quality.
92 However, the data model is unique to the NCEM dataset, which includes data relevant to the

93 field of electron microscopy.

94 Within the NFDI environment, Swate [13] is an Excel add-in for the annotation of experimental
95 data and computational workflows developed by the consortium NFDI4Plants. The tool is
96 intended for metadata annotation based on the ontology provided by the user. The use of a
97 spreadsheet environment aims at providing an intuitive and low-friction workflow, principally
98 focusing on wet-lab applications, where the user can annotate work-relevant metadata as the
99 experiment is performed. Hence, in this tool, the work is done manually by the user.

100 A second tool within the NFDI infrastructure is presented in this paper. Developed within the
101 NFDI4Ing consortium at the Technical University of Munich (TUM), HOMER is a metadata
102 crawler to be integrated in script-based (HPMC) workflows aiming at retrieving metadata that
103 can be attached to the raw data published by researchers. The tool is designed to be flexible and
104 adjustable to the user's needs in its application and easy to implement in potentially any HPMC
105 workflow. Moreover, it has been written with a modular structure, so it can easily be developed
106 further. The automated metadata extraction is based on the ontology schemes provided by the
107 user and is highly customizable.

108 The code structure is described in section 2, while a simple application is described in section 3.
109 Finally, in section 4, an overview on the future steps in the code development is given.

110 2 Code Description

111 2.1 Characteristics

112 HOMER is a code written in python and, at the moment, its complete extraction workflow
113 consists of 5 steps, as shown in figure 1. The code has been developed as a collection of routines,
114 each performing a different action, rather than as a single script. This guarantees a flexible
115 application of the crawler, as the user doesn't have always to perform all the steps described in the
116 next subsection, especially once the initial setup of the workflow has been done for the first time.
117 The modularity of the code also allows the developers and the users to easily modify/expand the
118 capabilities of the crawler, so that the code can be tailored for specific applications, if needed.
119 HOMER can be employed both locally and on HPC systems. However, it should be noted that,
120 as of now, it does not support a parallel implementation to parse the target files. Conceived as a
121 tool to be integrated in script-based workflows, the crawler should be run after the simulation
122 (or, potentially, any processing step), similarly to ExtractIng. Hence, the metadata is naturally
123 extracted in edge mode (where the data are generated). However, the tool can also be used to
124 retrieve metadata from centrally-stored, previously collected data, similarly to Xtract. In this
125 case, though, the user has to perform some extra steps according to the specific case at hand (see
126 next subsection). Together with metadata extraction, the tool gives the user the opportunity to
127 perform some simple post-processing operations as well, such as trimming strings or calculating
128 the minimum, maximum or mean of a series of values.

129 2.2 Implementation

130 When running the code for the first time, the following five steps (figure 1) are needed, with two
131 of them requiring direct user input. A step-by-step example is shown in section 3.

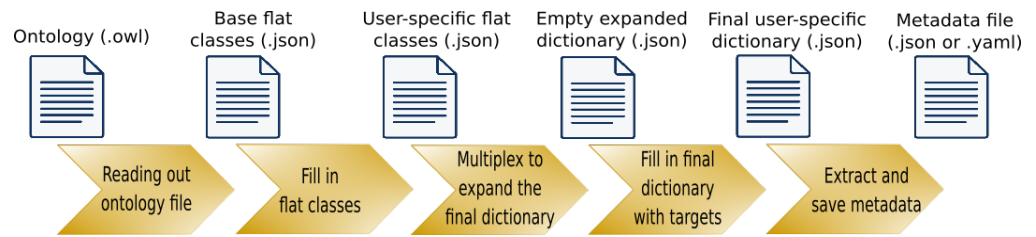


Figure 1: The five steps the crawler is currently composed of and related input/output files.

132 The first step consists of reading out the ontology file and creating an empty dictionary containing
 133 the flat classes as specified in the ontology. The step is performed by the routine `ClassUtils.py`
 134 and takes advantage of the python package `Owlyready2` to work on the ontology. The empty
 135 dictionary is a list of the classes and their attributes as they appear in the ontology file. The
 136 dictionary is a sort of template, where the general classes are simply listed and no specific
 137 instance of a class is created. The file, however, contains the fields that allow the user to specify
 138 how many instances (and related properties) of each class are to be created.

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

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

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

153 The fifth and last step consists in the actual extraction of the metadata and is performed through
 154 the routine `EntityUtils.py`. Once the metadata has been extracted, it is printed out to a file
 155 in a structured human-readable format (JSON or YAML). Currently, metadata can be extracted
 156 from files, such as text files using regular expressions or HDF5 files using `h5py`, from the output
 157 of a operating-system command, or can be directly hardcoded, if needed.

158 As mentioned, the user must first adapt the crawler to the specific simulation code in use
 159 (type and amount of metadata available to be extracted, location and format of the target files,
 160 extraction methods...). This involves the two (lengthy) manual steps just described. However,
 161 once completed, the configuration file (step four) can be re-used with little to no modification
 162 every time the simulation code is run again. Specifically, simply step five needs to be performed
 163 in the subsequent runs. This allows for a seamless integration of the tool within the workflow.

164 3 Example Application

165 In this section, a simple usage of the tool is shown. HOMER is intended to be used, potentially,
 166 for any script-based HPMC application. For example in the field of computational fluid dynamics
 167 (CFD), it could be used to extract the initial conditions of a simulation from the input file(s),
 168 some relevant flow-field quantities from the output file(s) of the CFD code and information on
 169 the specific hardware used to perform the calculation provided by the HPC system. Taking the
 170 case of an airplane's wing simulation as an example, the user could be interested in retrieving
 171 the freestream pressure and temperature, the Reynolds number, the drag coefficient of the wing
 172 and the ID-number of the node(s) where the calculation was performed. All the extracted pieces
 173 of information are to be classified according to the specific metadata scheme chosen by the user.
 174 For example, using the Metadata4Ing scheme, for the simulation step the user can define the
 175 class "Processing_Step", give it a name (such as "Running simulation") and store all the
 176 relevant parameters (pressure, temperature, Reynolds, and so on) with their values and units of
 177 measures as properties of the main class. If a different ontology scheme is used, the classification
 178 of the extracted metadata will be of course different, just like different metadata will need to be
 179 extracted if the crawler is employed in a different research field, such as structural mechanics or
 180 particle physics.

181 Hence, in order to show the working principles of HOMER in a simple, clear and application-
 182 independent way, the crawler is applied to extract metadata from a pizzeria menu using a
 183 showcase ontology. This simplified, limited example helps in laying out the steps described in
 184 subsection 2.2. A brief outlook of how the code could be employed in a real CFD workflow is
 185 given at the end of this section.

186 3.1 Showcase Pizza Ontology

187 This application is based on a simple "Pizza ontology". All the files referenced hereafter can
 188 be found in the directory /SimpleApplication_PizzaOntology on the GitLab repository of
 189 the code. Assume the following scenario: three persons always have lunch at the same pizzeria
 190 from Monday to Saturday and always choose the same pizza each. One goes for the first special
 191 pizza in the menu, the second always goes for the second special, while the third one always
 192 chooses the first option listed in the White-Pizzas¹ section of the daily menu. The menus vary
 193 during the week, so that every day each person has a different pizza compared to the previous
 194 day. In this example, HOMER is used to retrieve from the daily menu two pieces of information
 195 for each person: the price ("Price") and the name of the pizza ("Has_main_topping").

196 To draw a comparison with the extraction of metadata for a HPMC application using the Meta-
 197 data4Ing scheme, each person represents a processing step and the two properties name and price
 198 exemplify a computational variable and its unit of measure, respectively.

199 The information to be retrieved is stored in text files, as shown below, and metadata are extracted
 200 using regular expressions.

```
201 *****-- White Pizzas --*****
202 White First choice: Monterosa
```

1. A "white" pizza does not contain tomato sauce, as opposed to a "red" one

```
203 White First choice price: 7.5
204 ...
205 *****-- Special Pizzas --*****
206 Special First choice: Carbonara
207 Special First choice price: 7.5
208 ...
209 Special Second choice: Vegana
210 Special Second choice price: 7
211 ...
```

212 After setting up the (optional) python virtual environment and having installed the crawler, the
213 first command to run is `ClassUtils.py`, which retrieves the ontology and creates a dictionary
214 in the form of a `.json` file as shown here.

```
215 {
216     ...
217     "Vegetarian Pizza": {
218         "__count__": 1,
219         "__is_subclass_of__": ["Pizza"],
220         "__restrictions__": ["ontology.main_topping_of.exactly(1,
221 owl.Thing)"],
222         "Price": [1],
223         "Has_main_topping": [1],
224         "Has_topping": [1]
225     },
226     ...
227 }
```

228 The empty dictionary has to be manually adjusted to the specific case by the user. In this exam-
229 ple, the classes have been trimmed, so that the "Special Pizza" class will be repeated twice
230 ("__count__": 2) and each of the two instances will have one "Price" and one "Has_main_top-
231 ping" attributes ([1,1]). The result of the manual file manipulation for the "Special Pizza"
232 class is shown below. Only one instance of the "White Pizza" class is needed, instead, so the
233 keyword to use is "__count__": 1.

```
234 {
235     ...
236     "Special Pizza": {
237         "__count__": 2,
238         "__is_subclass_of__": ["Pizza"],
239         "__restrictions__": ["ontology.main_topping_of.exactly(1,
240 owl.Thing)"],
241         "Price": [1,1],
242         "Has_main_topping": [1,1]
243     },
244     ...
```

245 }
 246 Running `Multiplexer.py` expands the classes according to the parameters indicated in the
 247 previous step. The output is shown below. The new empty dictionary containing all the instances
 248 and corresponding attributes the crawler will use in the creation of the metadata file.

```

249 {
250   "Special Pizza_1": {
251     "__restrictions__": ["ontology.main_topping_of.exactly(1, owl.
252   Thing)"],
253     "Price": {
254       "path": [""],
255       "type": [""],
256       "pattern": [""],
257       "postprocessor": [{
258         "type": "",
259         "args": ""
260       }]
261     },
262     "Has_main_topping": {
263       "path": [""],
264       "type": [""],
265       "pattern": [""],
266       "postprocessor": [{
267         "type": "",
268         "args": ""
269       }]
270     }
271   },
272   {
273     "Special Pizza_2": {
274       ...
275     },
276     ...

```

277 The multiplexed dictionary has to be filled in manually again by the user, as shown below.
 278 How to fill in the dictionary depends on how the user wants to retrieve the data and where the
 279 information is stored. In this example, data are all extracted from plain text files and the crawler
 280 uses regular expressions to locate and read the data. This is done by specifying the keywords:
 281 "path", "type" and "pattern". The entries in "postprocessor" can be left empty for the
 282 sake of this example. The complete keywords for "Special Pizza_1" are shown here.

```

283 {
284   "Special Pizza_1": {
285     "__restrictions__": "ontology.main_topping_of.exactly(1, owl.
286   Thing)",

```



```

2874     "Price": {
2885         "path": "PizzaExample/Menus/RandomMenu_1.txt",
2895         "type": "regex",
2907         "pattern": "Special First choice price:\\s(.*)\\n",
2913         "postprocessor": {
2920             "type": "",
2930             "args": ""
2941         }
2952     },
2963     "Has_main_topping": {
2974         "path": "PizzaExample/Menus/RandomMenu_1.txt",
2985         "type": "regex",
2995         "pattern": "Special First choice:\\s(.*)\\n",
3007         "postprocessor": {
3013             "type": "",
3020             "args": ""
3030         }
3041     }
3052 },
3063 ...
3074 }

```

308 Finally, `EntityUtils.py` is used to run the actual extraction routine, which retrieves the meta-
309 data according to the parameters specified in the previous step. The output file is shown below
310 and could be either a `.json` or a `.yaml` file.

```

3111 {
3122     "Special Pizza_1": {
3133         "Price": "7.5",
3144         "Has_main_topping": "Carbonara"
3155     },
3166     "Special Pizza_2": {
3177         "Price": "7",
3188         "Has_main_topping": "Vegana"
3199     },
3200     "White Pizza": {
3211         "Price": "7.5",
3222         "Has_main_topping": "Monterosa"
3233     }
3244 }

```

325 At this point, the menu for Monday (`RandomMenu_1.txt`) has been crawled. The next steps
326 depend on the intended use for the crawler by the user, showing again the flexibility of the tool.

327 In this example, all data files to be crawled have already been generated (menus from 1 to 6).
328 Therefore, the crawler can be used in centralized mode. This means that, using a script (for

329 example `CyclingRandomMenu.sh` provided in the folder), it is possible to easily automate the
 330 metadata extraction by simply adapting the filled multiplexed dictionary from step 4 (that means
 331 changing `./RandomMenu_1.txt` to `./RandomMenu_N.txt`, with suitable N, in "path") and
 332 re-run step 5.

333 The other option would be to integrate the crawler within the workflow right after the data files
 334 are generated (for example, once a generic `RandomMenu.txt` has been created) and use the tool
 335 in edge mode. In such a case, the filled multiplexed dictionary wouldn't need any modification,
 336 so that simply the command for step 5 would have to be invoked.

337 3.2 Simple CFD-like Application

338 Within a script-based workflow, the natural use of the crawler would be the edge mode. The first
 339 usage requires, as mentioned, to perform all the five steps. After that, HOMER can be seamlessly
 340 integrated in the workflow without further modifications. The snippet below shows a short
 341 example of metadata extracted from a simulation input and output files for the code NSMB [14]
 342 and classified according to the Metadata4Ing ontology. In this scheme, the reference classes are
 343 "Processing_Step", "Tool" and "Method", with information such as parameters name and
 344 numerical value being considered as properties of those classes. Some entries have been hard-
 345 coded by providing a full string (using "type": "string" and "pattern": "<string>"),
 346 others have been extracted from command-line outputs (using "type": "os" and "pattern":
 347 "<command>"), while others are extracted from the text files as previously shown.

```

348 {
349     "Processing_Step": {
350         "Name": "Running Simulation",
351         "Parameter_1": "Freestream Mach number",
352         "Parameter_2": "Freestream pressure",
353         "Parameter_3": "Freestream temperature",
354         "Parameter_4": "Freestream unit Reynolds number",
355         "Parameter_5": "Start time",
356         "Has_numerical_value_1": "9.1",
357         "Has_numerical_value_2": "730",
358         "Has_numerical_value_3": "160",
359         "Has_numerical_value_4": "3.22E6",
360         "Has_numerical_value_5": "10:13:31"
361     },
362     "Tool_1": {
363         "Type": "Hardware",
364         "Name": "lrz-coolmuc2-linux-cluster-2022"
365         "Computational node": "i22r07c05s05"
366     },
367     "Tool_2": {
368         "Type": "Software",
369         "Name": "NSMB",
370         "Version": " 6.09.21    Date: 28 - January - 2021    "
  
```

```
3724     },  
3725     "Method": {  
3735         "Name": "LU-SGS"  
3747     }  
3753 }
```

376 4 Conclusion and Future Developments

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

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

395 As of now, HOMER can be already implemented in HPMC workflows, so as to enrich each
396 processing step (e.g. mesh generation, simulation, post-processing, report) by adding the corre-
397 sponding metadata. This capability allows for the collection of valuable data (such as the energy
398 consumption for a set of simulations) to enable secondary research and the development of new
399 methodologies in HPC systems. In the current state, the tool would provide the best performance
400 when used to extract metadata at the edge, as no parallel implementation for file parsing is
401 present, yet. In its five-steps implementation described in this work, HOMER has mainly been
402 utilized in the data life cycle (figure 2) for the processing stages of planning, creating/collecting
403 and processing/analyzing. A future step would be to cover the complete data life cycle in a
404 holistic approach, by providing the possibility to automatically preserve and publish/share the
405 extracted metadata along with the research dataset.

406 Through publishing not just the data but also administrative (preservation) metadata, third
407 party users will be able to retrieve crucial information about accessibility, access rights and
408 licenses among others. Bibliographic (author, identifiers) and descriptive (research domain,
409 tools, methods, processing steps) metadata can be published in repositories together with the
410 referenced research data, or be linked to the research data by persistent links and identifiers, if



Figure 2: Data Life Cycle [15].

411 technical or organizational reasons impede a joint provisioning (e.g. if the research data is too
412 large to be stored in a common repository).

413 One main factor of making data FAIR is a controlled vocabulary / common terminology. This
414 is guaranteed by the fact that HOMER supports the usage of semantic ontologies as metadata
415 schemes. These schemes have to be matched somehow with searchable metadata fields in the
416 corresponding repositories. Only few repositories offer such publishing options, like DaRUS
417 (University of Stuttgart) [16], which uses predefined metadata blocks, or Coscine (RWTH)
418 [17], which provides the possibility to use standardized or self-created metadata application
419 profiles [18]. These schemes still have to be parsed with the corresponding metadata fields in
420 the extracted metadata file, to provide the metadata in a standardized, searchable and indexable
421 front end. The NFDI4Ing consortium is simultaneously working on a generic interface which
422 combines different kinds of metadata and data repositories with one standard-based interface.
423 This enables the linking between all data and metadata of the research data life cycle, including
424 experiments, raw data, software, subject-specific metadata sets, and the tracking of usage and
425 citations. Standardized and automatically extracted metadata files can easily be made findable
426 and accessible by this new generic interface [19]. Therefore, HOMER can be a crucial piece
427 within the metadata toolchain from using common vocabularies and automatized extracting to
428 FAIR publishing. The already mentioned Metadata4Ing ontology has been used as the reference
429 during the early stages of the development of HOMER. In the meantime, a HPMC-sub-ontology
430 has been developed within Metadat4Ing. Hence, one of the next steps will be to further adapt
431 HOMER to this new sub-ontology, allowing the tool to be more effective in the complete data
432 life cycle of a CFD workflow on HPC systems.

433 5 Acknowledgements

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

438 6 Roles and contributions

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

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

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

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

446 **Vasiliki Sdralia:** Writing – original draft (Introduction)

447 **Christian Stemmer:** Funding acquisition; Resources; Supervision; Writing – review and editing

448 References

- 449 [1] P. B. Heidorn, “Shedding Light on the Dark Data in the Long Tail of Science,” *Library*
 450 *Trends*, vol. 57, no. 2, pp. 280–299, 2008. DOI: [doi:10.1353/lib.0.0036](https://doi.org/10.1353/lib.0.0036).
- 451 [2] B. Schembera and J. M. Duràn, “Dark Data as the New Challenge for Big Data Science
 452 and the Introduction of the Scientific Data Officer,” *Philosophy & Technology*, vol. 33,
 453 pp. 93–115, 2020. DOI: <https://doi.org/10.1007/s13347-019-00346-x>.
- 454 [3] NFDI4Ing Consortium. “Website.” (2022), [Online]. Available: <https://nfdi4ing.de>.
- 455 [4] Metadata4Ing Workgroup. “Metadata4ing: An ontology for describing the generation of
 456 research data within a scientific activity.” (2022), [Online]. Available: [https://nfdi4i
 457 ng.pages.rwth-aachen.de/metadata4ing/metadata4ing/index.html#ref](https://nfdi4ing.pages.rwth-aachen.de/metadata4ing/metadata4ing/index.html#ref).
- 458 [5] S. Liang, V. Holmes, G. Antoniou, and J. Higgins, “Icurate: A research data management
 459 system,” in *Multi-disciplinary Trends in Artificial Intelligence*, A. Bikakis and X. Zheng,
 460 Eds., Cham: Springer International Publishing, 2015, pp. 39–47, ISBN: 978-3-319-26181-
 461 2. DOI: [10.1007/978-3-319-26181-2_4](https://doi.org/10.1007/978-3-319-26181-2_4).
- 462 [6] C. S. Adorf, P. M. Dodd, V. Ramasubramani, and S. C. Glotzer, “Simple data and workflow
 463 management with the signac framework,” *Computational Materials Science*, vol. 146,
 464 pp. 220–229, 2018, ISSN: 0927-0256. DOI: [https://doi.org/10.1016/j.com
 465 matsci.2018.01.035](https://doi.org/10.1016/j.commatsci.2018.01.035).

- 466 [7] T. J. Skluzacek, “Dredging a data lake: Decentralized metadata extraction,” in *Proceedings*
467 *of the 20th International Middleware Conference Doctoral Symposium*, ser. Middleware
468 ’19, Davis, California: Association for Computing Machinery, 2019, pp. 51–53, ISBN:
469 9781450370394. DOI: <https://doi.org/10.1145/3366624.3368170>.
- 470 [8] T. J. Skluzacek, R. Chard, R. Wong, *et al.*, “Serverless workflows for indexing large
471 scientific data,” in *Proceedings of the 5th International Workshop on Serverless Computing*,
472 ser. WOSC ’19, Davis, CA, USA: Association for Computing Machinery, 2019, pp. 43–48,
473 ISBN: 9781450370387. DOI: <https://doi.org/10.1145/3366623.3368140>.
- 474 [9] B. Schembera, “Like a rainbow in the dark: Metadata annotation for HPC applications in
475 the age of dark data,” *Journal of Supercomputing*, vol. 77, pp. 8946–8966, 2021. DOI:
476 <https://doi.org/10.1007/s11227-020-03602-6>.
- 477 [10] B. Schembera and D. Iglezakis, “The Genesis of EngMeta - A Metadata Model for
478 Research Data in Computational Engineering,” in *Metadata and Semantic Research*,
479 Cham: Springer International Publishing, 2019, pp. 127–132, ISBN: 978-3-030-14401-2.
480 DOI: https://doi.org/10.1007/978-3-030-14401-2_12.
- 481 [11] S. Padhy, G. Jansen, J. Alameda, *et al.*, “Brown dog: Leveraging everything towards
482 autocuration,” in *2015 IEEE International Conference on Big Data (Big Data)*, 2015,
483 pp. 493–500. DOI: [10.1109/BigData.2015.7363791](https://doi.org/10.1109/BigData.2015.7363791).
- 484 [12] G. P. Rodrigo, M. Henderson, G. H. Weber, C. Ophus, K. Antypas, and L. Ramakrishnan,
485 “ScienceSearch: Enabling search through automatic metadata generation,” in *2018 IEEE*
486 *14th International Conference on e-Science (e-Science)*, 2018, pp. 93–104. DOI: [10.1109](https://doi.org/10.1109/9/eScience.2018.00025)
487 [9/eScience.2018.00025](https://doi.org/10.1109/9/eScience.2018.00025).
- 488 [13] K. Frey, K. Schneider, O. Maus, and T. Mühlhaus. “Swate: A swate workflow annotation
489 tool for excel.” (2022), [Online]. Available: [https://github.com/nfdi4plants/Swa](https://github.com/nfdi4plants/Swate)
490 [te](https://github.com/nfdi4plants/Swate).
- 491 [14] J. Vos, N. Duquesne, and H. J. Lee, “Shock wave boundary layer interaction studies using
492 the NSMB flow solver,” in *3rd European Symposium on Aerothermodynamics for Space*
493 *Vehicles, ESA SP-426*, 1999.
- 494 [15] UK Data Service, modified by TUM University Library (UB). “Data life cycle - icons.”
495 (2022).
- 496 [16] University of Stuttgart. “Darus.” (2022), [Online]. Available: [https://www.izus.uni-](https://www.izus.uni-stuttgart.de/en/fokus/darus/)
497 [stuttgart.de/en/fokus/darus/](https://www.izus.uni-stuttgart.de/en/fokus/darus/).
- 498 [17] RWTH Aachen University. “Coscine.” (2022), [Online]. Available: [https://coscine](https://coscine.rwth-aachen.de)
499 [.rwth-aachen.de](https://coscine.rwth-aachen.de).
- 500 [18] RWTH Aachen University. “Aims – applying interoperable metadata standards.” (), [On-
501 line]. Available: [https://www.wzl.rwth-aachen.de/cms/wzl/Forschung/Forsch](https://www.wzl.rwth-aachen.de/cms/wzl/Forschung/Forschungsumfeld/Forschungsprojekte/Projekte/~ivong/ProMiDigit-Process-Mining-fuer-No-Code/)
502 [ungsumfeld/Forschungsprojekte/Projekte/~ivong/ProMiDigit-Process-Mi](https://www.wzl.rwth-aachen.de/cms/wzl/Forschung/Forschungsumfeld/Forschungsprojekte/Projekte/~ivong/ProMiDigit-Process-Mining-fuer-No-Code/)
503 [ning-fuer-No-Code/](https://www.wzl.rwth-aachen.de/cms/wzl/Forschung/Forschungsumfeld/Forschungsprojekte/Projekte/~ivong/ProMiDigit-Process-Mining-fuer-No-Code/).
- 504 [19] NFDI4Ing Consortium. “Metadata hub.” (2022), [Online]. Available: [https://git.rwt](https://git.rwth-aachen.de/nfdi4ing/s-3/s-3-3/metadatahub)
505 [h-aachen.de/nfdi4ing/s-3/s-3-3/metadatahub](https://git.rwth-aachen.de/nfdi4ing/s-3/s-3-3/metadatahub).