FAIRlead: A Conceptual Framework for a Model Driven Software Development Approach in the Field of FAIR Data Management

Andreas Schmidt[1,2][a], Mohamed Anis Koubaa[1][b], Nan Liu[1][c], Philipp Schmurr[1][d], Karl-Uwe Stucky[1][e] and Wolfgang Süß[1][f]

[1]*Institute for Automation and Applied Computer Science, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*

[2]*Department of Computer Science and Business Information Systems, Karlsruhe University of Applied Sciences, Karlsruhe, Germany*

{*andreas.schmidt, mohamed.koubaa, nan.liu, philipp.schmurr, karl-uwe.stucky, wolfgang.suess*}*@kit.edu*

Keywords: Code Generation, Metadata, Ontology Based Engineering, FAIR.

Abstract: The publication of scientific results together with the underlying experiments is an important source of further research. In 2016, the "FAIR Guiding Principles for scientific data management and stewardship" were published, in which the authors postulate a series of guidelines for improving the (F)indability, (A)ccessibility, (I)nteroperability and (R)eusability of digital information (FAIR). The point (I)nteroperability deals with the prerequisites for the reusability of digital objects. The central point here is the need to have a common understanding of the meaning of digital objects. This understanding is provided by formal languages of knowledge representation (ontologies), which describe the actual data. These descriptions of data are also known as metadata. As part of our current work at the Institute for Automation and Applied Computer Science (IAI) at KIT, we are implementing novel concepts and technologies for the sustainable handling of research data using high-quality metadata. As part of this work, we plan to develop a software tool that can be used to enrich data with suitable metadata and thus automate the process of making research results available. A key requirement is that the tool must be independent of the underlying domain. In order to be able to deal with data from any domain, we have opted for a model-driven approach in which an ontology, and possibly other platform-specific information, are input for a software generator, which then generates an (interactive) tool for specifying the metadata and linking it to the data itself. The generated tool includes the complete software stack, starting with a user interface, programmatic APIs for connecting additional application logic, and a persistence component. How these individual layers are realized is not specified, but defined by the mapping rules of the software generator, which also opens up the possibility of generating and evaluating different variants of the software.

1 INTRODUCTION

In computer science, an ontology is the formal naming and definition of the concepts, categories, properties and relationships between the concepts, data or entities of a particular domain (Ont, 2024). These basic concepts for ontologies are also the basic elements of Conceptual Models (CM), like the ER-Model (Chen, 1976) and UML (OMG, 2011). So, to that extend, languages and tools from both worlds can be used interchangeably. In the ontology context there are e.g. languages like OWL, RDF(S), and SHACL (Shapes Constraint Language) (SHACL, 2017). Having this in mind, the paper on hand generally uses ontology terminology and employs the CM terms only where more suitable for understanding.

[a] https://orcid.org/0000-0002-9911-5881
[b] https://orcid.org/0000-0001-8552-2008
[c] https://orcid.org/0009-0005-8768-7072
[d] https://orcid.org/0009-0004-2324-7839
[e] https://orcid.org/0000-0002-0065-0762
[f] https://orcid.org/0000-0003-2785-7736

1.1 Metadata

Scientific experiments take place in a specific context and it is within this context that data, parameters and results obtained have a practical meaning. In order to be able to interpret the underlying data and results in retrospect, detailed knowledge of this context is required. Data and its context form a unit that can be described by ontologies and their instantiations. This additional data is referred to as metadata, as it provides data about data.

Schmidt, A., Koubaa, M., Liu, N., Schmurr, P., Stucky, K. and Süß, W.
FAIRlead: A Conceptual Framework for a Model Driven Software Development Approach in the Field of FAIR Data Management.
DOI: 10.5220/0013013700003838
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2024) - Volume 3: KMIS*, pages 323-330
ISBN: 978-989-758-716-0; ISSN: 2184-3228

In the context of FAIR (Wilkinson et al., 2016), the present work deals with the aspect of (I)nteroperability, which intends the enrichment of data with rich metadata.

Due to the exponentially growing amount of scientific data, the enrichment with metadata must be automated by using suitable tools.

1.2 Example Scenario

In order to get a clearer picture of the requirements for such software, a concrete scenario will be presented here. The Institute for Automation and Applied Informatics (IAI) at KIT operates an experimental photovoltaic system (PV), consisting of a large number of panels that are connected together as an array (in parallel) or string (in series). Additional components of this PV system include power inverters, batteries, measuring equipment, and so on. An overview of the concepts and their relationships are shown in Figure 1. By reconfiguring the components, a large number of experiments can be carried out in order to achieve certain goals (e.g. maximum average electricity yield), or to observe the behavior under certain effects (e.g. partial shading).

As part of an experiment that is carried out with a specific interconnection of the components over a specific time interval and under specific weather conditions, a series of result data is generated that is stored in a time series database. In order to be able to interpret the measurement results, it is necessary to know the components' connectivity and the weather condition at this time. This is an example for meta information that has to be managed by our tool, together with the information on where the result values are stored.

1.3 Requirements for a Metadata Management Component

In order to fulfill this task, the software must have a persistence component that allows the specific test setup (the metadata) to be saved. Since we are talking about ontology-based metadata the database schema will be derived from the ontology. In addition, the component must have an interface through which the information can be entered. This can be, for example, a simple web-based CRUD (Create, Read, Update, and Delete) interface via which the individual components of the system can be specified, or a dedicated graphical editor with which the panels and other components can be graphically created and linked together.

The software must also maintain the connection between data and metadata. Data can be available in a variety of formats, such as measurement data in a time series database, relational data, parameter sets of a learned neural network, as well as a variety of proprietary data formats. Therefore, a component that establishes this data-metadata connection is needed.

Such a tool can provide the data of the experiment as well as the concrete setup (the metadata). By preparing this information according to existing standard formats, it is now able to export the data together with its metadata, or directly write it into a previously specified repository like the *databus* (Hoyer-Klick et al., 2023). This repository than covers the FAIR aspects (F)indable and (A)ccessible.

In contrast, in today's reality, the information about a specific experimental setup frequently is only implicitly available in configuration files, installation scripts or makefiles, which makes it almost impossible to extract this meta information.

The general functionality of the component just described is therefore not only useful for the publication of semantically enriched FAIR data, but also offers valuable services as an electronic notebook of the experiments carried out.

In addition, it is not limited to the PV domain used here as an example. The statements made here are valid for any domain. While the general functionality of the application is the same for all domains, the structure of the metadata will be different. It depends on the specific entities or concepts that describe the specific application. These are described by CMs or by the ontologies that describe the applications' domains.

1.4 Approach

And this brings us to the core idea of our research approach, the generation of an application, as described in the previous section, on the basis of the available domain information. We use a Model-Driven Software Development (MDSD) approach (see Section 2 for details). The application to be realized is generated from a model description (the ontology) and a number of transformation rules, which map the model information to source code for a specific target platform. In addition to the model information in the form of an ontology, the generator can process further information such as a specific GUI layout or information on the underlying software platform during the generation process.

The rest of the paper is structured as follows: Next, in Section 2 the basic terms and the methodology of MDSD are presented. Then the concept of our *FAIRlead* generator is presented in Section 3. Having

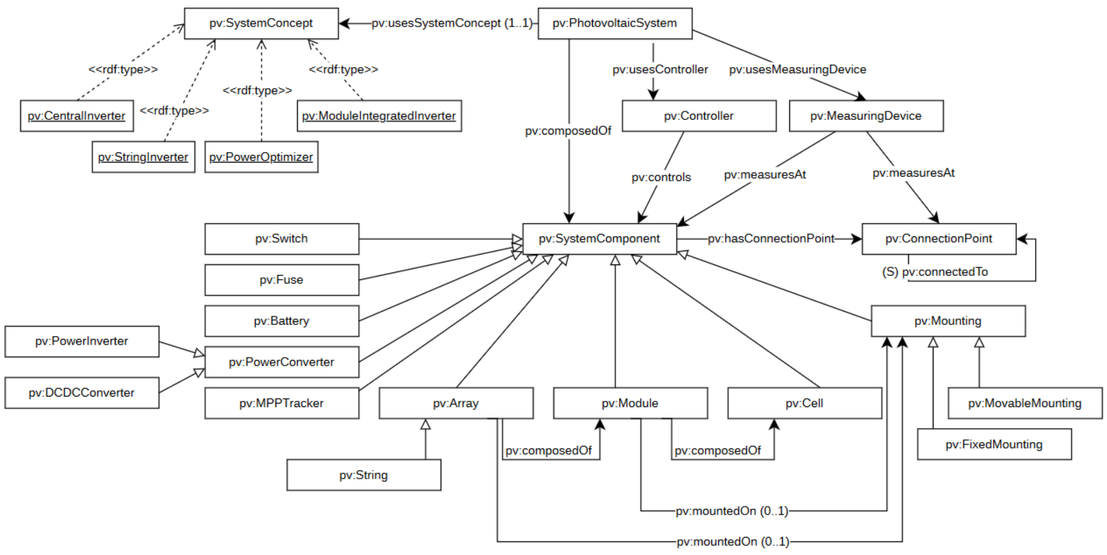


Figure 1: PV ontology (from (Schweikert et al., 2023)).

developed the concept so far, the following steps that we plan to take will be explained in Section 3. In Section 4, we take a closer look at the benefits we expect from the use of our generator, before giving a brief summary and outlook in Section 5.

2 MODEL DRIVEN SOFTWARE DEVELOPMENT

The basic idea of MDSD is the generation of source code from model information that describe the software to be developed (see Figure 2). Model descriptions are compact, abstract, formal, and platform-independent. As an example, the following is an abstract model description of the class Person:

```
<class Person(name: string(40),
              birthday: date,
              mother: Person,
              father: Person)>
```

Transformation rules are needed to map this abstract representation of the problem space onto programming code. These are usually provided in the form of templates and add the platform-specific information to the model. The following code fragment shows an example of a transformation rule that transforms the above model description into executable PHP code (specifically: a class description with constructor and setter methods). To do this, a template language (language elements shown in red) is used to integrate the variable parts from the model into the static code framework (in black).

```
<? foreach ($model->classes as $class) { ?>
  // class generated, do not edit !!!
  // timestamp: <?= date(DATE_RFC2822); ?>
  class <?= $class->name; ?>  {
  <? foreach ($class->properties as $p) { ?>
         protected $<?= $p ?>;
  <? } ?>
    function __construct() { }

    <? foreach ($class->properties as $p) { ?>
         function set<?= ucfirst($p) ?>($v) {
             $this-><?= $p ?> = $v;
         }
    <? } ?>
  }
<? } ?>
```

The code generated from the model and the transformation rule with the help of the generator then, after an additional code formatting step (not shown in Figure 2), looks like this:

```
// class generated, do not edit !!!
// timestamp: Tue, 24 Sep 2024 14:49:37 +0200
class Person {
    protected $name;
    protected $birthday;
    protected $mother;
    protected $father;

    function __construct() { }

    function setName($value) {
        $this->name = $value;
    }

    function setBirthday($value) {
        $this->birthday = $value;
    }
// more code folows here ...
}
```

Typically, 60% to 80% of an application's code can be generated (Stahl and Völter, 2006). The basic functionality of the intended application can even be generated almost by 100%. In addition to the higher development speed, this code typically has a higher quality, as the transformation rules are centrally defined in the generator templates and are consistently applied to the generated platform code. Even the quality of the software architecture is usually higher, as more thought is given to the underlying architecture during the template development. Furthermore, there is a clear separation of functional and technological aspects, so that the transition from one technological platform to another only requires an adaptation of the templates, but the model remains untouched.

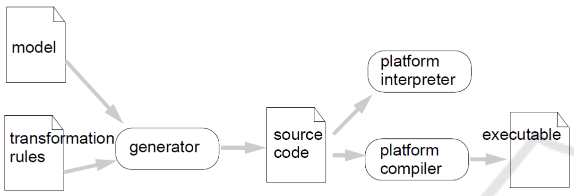


Figure 2: Principle function of a software generator. The input for the generator is the abstract, platform-neutral description of the application to be realized, as well as the platform-specific transformation rules. The result is the generated source code, which is then compiled in a further step or executed directly by an interpreter.

Starting point of an MDSD project is a reference implementation that is as lean as possible but executable. It serves as the basis for the development of the generator templates, which perform the transformation into source code. Once the reference implementation has been created, it is analyzed and the code is broken down according to the following criteria (Stahl and Völter, 2006):

1. generic code, that is the same for all possible applications
2. schematic, repetative code, which is individual for each application but has the same schematic structure
3. application-specific code (individual code)

This approach is illustrated in Figure 3. The generic, repetetive code (1) can simply be used, while the schematic code (2) is the starting point for creating the transformation rules in the generator templates. For this purpose, the code fragments are generalized into transformation rules and merged with the model information, as shown in Figure 2.

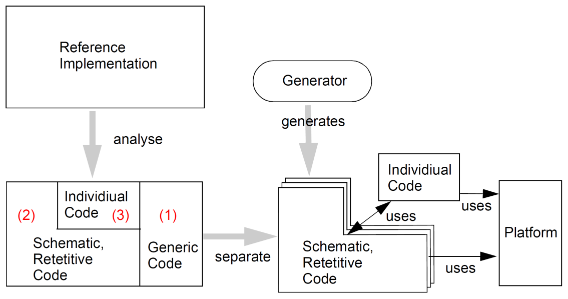


Figure 3: Principle of MDSD (adapted from (Stahl and Völter, 2006)).

3 FAIRlead APPROACH

In a preliminary step, we choose the target platform. This is the platform on which the generated application must be able to run. This can be a programming language such as PHP or a platform such as .net. A framework such as django (Vincent, 2019) or symfony (Zaninotto and Potencier, 2008) can also serve as a target platform. The use of a framework has the advantage that some of the tasks that would otherwise have to be covered by the generator are already covered by the framework (e.g. creation of CRUD interfaces for the concepts occurring in the ontology, creation of database schemas, object-relational mapping).

In line with the MDSD approach presented in the previous section, we will develop a simple reference implementation for our PV domain in a first step. The functionality of the application corresponds to that described in Section 1.3. We then identify both the generic part and the repetitive schematic part. The reference implementation can be very simple, as it only serves to verify the functionality of the generator. In the case that one or more components support introspection, generic approaches can also be pursued, as the meta information (e.g. properties of a class and their types) can be read and analyzed at runtime. This is especially important in order to implement communication interfaces for domain components, and also for the generic creation and extension of database schemas.

The next step is to analyze whether the input ontology provides enough information for generation of the schematic code part, or if additional information must be supplied to the generator. An example are SHACL definitions that extend OWL ontologies with the definitions of necessary properties, cardinality constraints or value ranges. Furthermore, platform-specific information, such as the graphical layout of the GUI, etc., will certainly be added. However, it remains to be seen to what extent this will be necessary.

A further task is the selection of a suitable generator.

The spectrum here ranges from in-house development with a scripting language such as PHP or Python together with a templating module available for the programming language, such as *twig* (Twig, 2024) or *smarty* (Smarty, 2024) for PHP, *mako* (Mako, 2024) or *jinia2* (Jin, 2024) (for Python), to the use of a tool such as the modeling workflow engine (MWE, 2024) in the Eclipse Modeling Project (EMP, 2024). The decisive factor is which information the abstract metamodel of the generator already contains and how flexibly the metamodel can be extended to meet the application's requirements. If a suitable generator is not available, an abstract metamodel with all the necessary properties has to be developed programmatically (i.e. as a set of Java classes) and then combined with an existing template framework in the selected language to be used as a software generator. Figure 4 shows the dependencies when selecting a suitable generator. In order to map the concepts described in the ontology, the generator's internal meta model must support them, or one must be able to extend the existing meta model to do so. Only under this condition is it possible to address the aspects described in the ontology within the templates and thus implement them in the generated application.

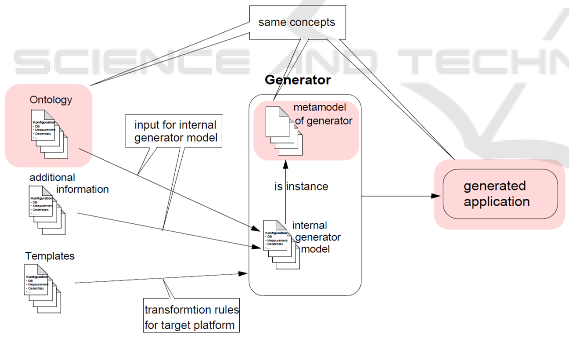


Figure 4: Dependencies between ontologies, generator and generated application. The concepts from the ontology can only be implemented in the application to be generated if the generator's internal meta-model supports them.

Once the generator is chosen, templates are created to regenerate the code of the reference implementation.

In a further step, the reference implementation is extended. This includes domain-specific tasks like the connection of external components such as proprietary systems or real time measurement software (i.e. Beckhoff automatisation software (Beck, 2024)) as well as the integration of already established repositories and registries following with the FAIR principles. The aim of this step is to establish meaningful extension interfaces that behave independently of the domain (i.e. the time series exporter tool `Zeitgeist` (Schmidt et al., 2023)). The newly established extension interface can then also become the starting point for a plug-in architecture of the generated code.

Our plan is to develop a generator framework which, in a first stable version, provides a web-based interface with CRUD functionality. This framework will allow the integration of meta information and also enables the referencing of primary data in various data sources (e.g. relational databases, time series databases, ...). At this time, the framework is also planned to be released as a GitHub project to gain input and enhancements from the community.

Figure 5 gives an overview of the architecture. Input for the *FAIRlead* generator are the ontologies and the generator templates, which define the transformations on the target platform. Further input can come from SHACL files, which further specify the input ontologies. In addition to the source code for GUI and CRUD functionality, the database schema is also generated. On the right-hand side you will find manually created application logic. The arrows that originate from the application logic represent the connection of the manually created code to the extension interfaces of the generated code. Patterns on how this can be done can be found in (Stahl and Völter, 2006). The database with the meta information, which is linked to the actual data (bottom), is located near the center of the figure. Since we want to support the widest possible range of data sources, we need to find an API that is as generic as possible. However, there are already approaches here, such as (DTP, 2024; ODC, 2024; Beam, 2024), which we will examine for their suitability.

Further research steps include:

- finding interfaces between the individual parts of the application so that the individual components are as interchangeable as possible (i.e. exchange a simple web-basd formular with a graphical editor for specifying the instances of domain concepts).
- Mechanisms for the connection between metadata and data.
- Detection of inconsistencies between data and metadata (consistency checks).

Figure 6 shows a flowchart of the individual activities we have defined so far, along with their dependencies.

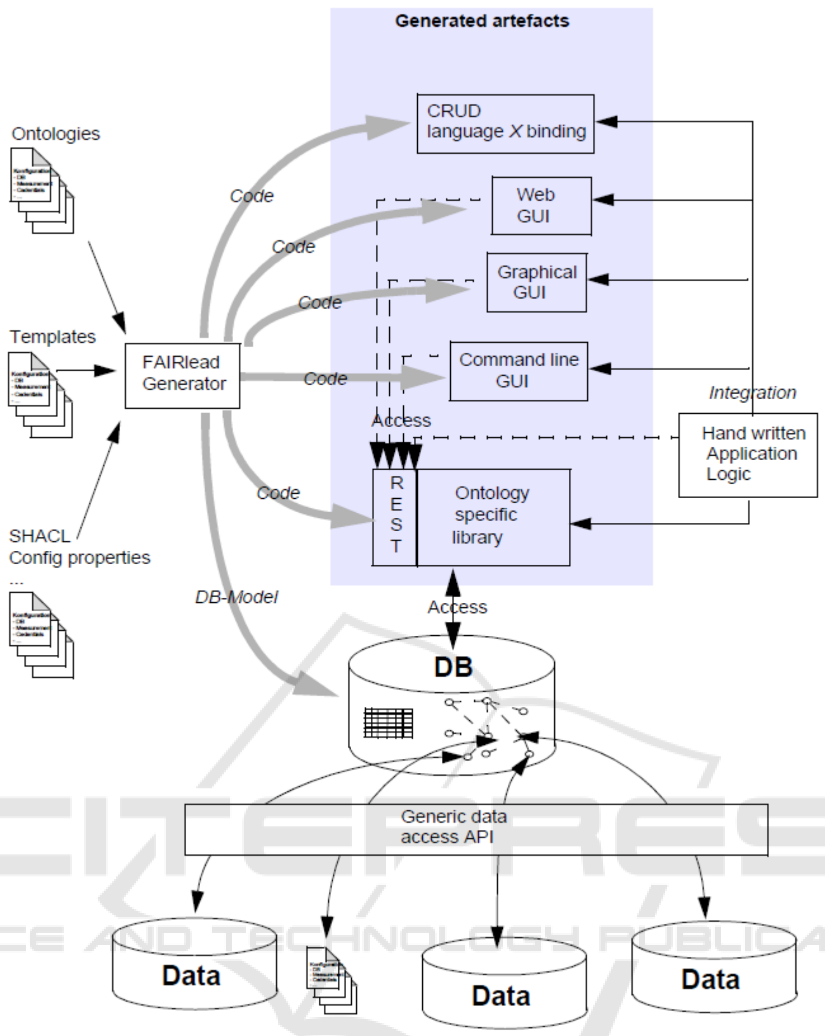


Figure 5: FAIRLead Generator architecture.

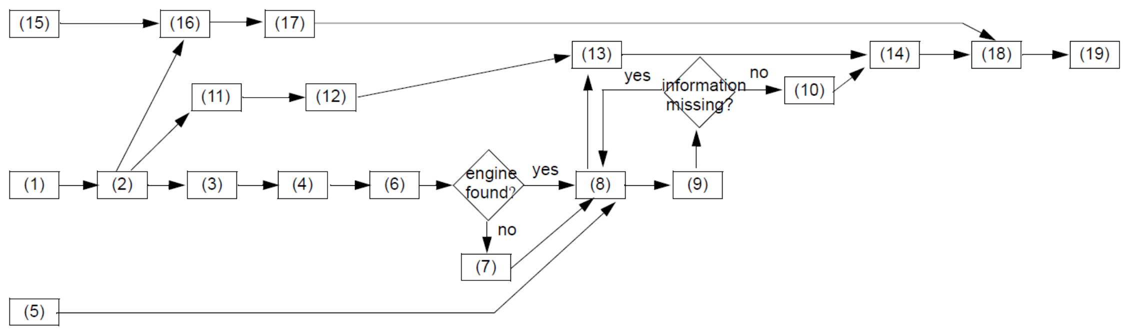
4 ADVANTAGES OF THE FAIRlead APPROACH

In the FAIR community, there are currently a number of alternative concepts for implementing the principles. An Example is given by FAIR Digital Objects (FDO) which, e.g., have been compared to Linked Data approaches in (Soiland-Reyes et al., 2024). Web Frameworks are available as well as systems based on Web Components (Schmidt and Tobias, 2024). As with respect to storage, Triple-Store-based solutions exist beside alternative persistence concepts. With our approach, it is easy to compare and evaluate all those different implementation variants in an overall system, since only the corresponding additional templates have to be developed for the different implementations.

The generic approach shifts a significant proportion of the application implementation to the conceptual, technology-independent part - defined by ontologies - and therefore takes place at a higher level of abstraction. As a result, design decisions like those mentioned in the preceding paragraph can be made later or realized in parallel and comparatively with minimum effort. The only prerequisite for this is the development of one or more corresponding technology-specific templates (see Figure 3).

In addition, once the software generator has been fully developed, it is easier to involve technical experts in the creation of new application systems, as discussions can take place at a purely conceptual level and the target architecture is generated by the generator and the templates already developed at this stage.

We plan to make the generator we have developed



1. Choose target platform for PV-application
2. Build reference implementation (RI) for PV-ontology (full stack)
3. Anaylze code of RI according to repetative patterns (RP) (see Section 2)
4. Analyze ontology information concerning information needed to generate RP
5. Search for libraries that extract information from ontologies
6. Look for an appropriate generator engine
7. Implement own generator engine (choose language, template engine, define internal metamodel
8. Build/extend ontolgy import modul for generator
9. Create templates
10. Generate PV-application using created templates
11. Extend reference implementation with application logic
12. Design application logic REST-API interface
13. Create templates for application logic interface
14. Extend generated PV-application with application logic using generated interface
15. Linking data and metadata. Investigation of existing libraries for referencing different data types/sources.
16. Integration of the referencing functionality in the RI
17. Creation of templates for referening datasets
18. Generation of the application with complete functionality
19. Applying the concept to other areas (ontologies)

Figure 6: Flowchart of the activities we have defined and are implementing.

available to the FAIR community as open source. This will enable researchers around the world to use our tool by using their ontologies as input for the generator and generating the domain-specific application for linking data and metadata.

This application can then be integrated into the respective laboratory infrastructure and used, for example, as an electronic laboratory notebook and, on the other hand, make the step towards publishing their research results much easier.

However, the functionality of the generated application can easily extend by developing new generator templates or adapting the templates to your own requirements. This is not particularly complicated if you use the existing generator templates as a blueprint.

5 CONCLUSION AND OUTLOOK

The enrichment of data with associated metadata is seen as a necessary step to make research results reproducible, to verify scientific experiments or to build on the results of these. In this context, we presented the conceptual framework *FAIRlead*, which is designed to support scientists in enriching data with metadata. To support people from different domains, we have chosen a model-driven approach that generates software artifacts to enrich data with metadata based on an ontology description of the domain.

The functionality of the generated software includes the specification of a scientific experiment based on the concepts defined in the ontology and their relationships to each other as well as the linking to the actual data. Now that we have defined the conceptual framework for our future work, we will next carry out a reference implementation based on the PV ontology we have developed in order to derive our generator templates, which form the core of the *FAIRlead* generator. An important future step is the establishment of a clean interface architecture in the generated application so that different possible implementation variants can be easily exchanged. The same applies to the interface for connecting the business logic and external systems.

REFERENCES

Beam (2024). Apache beam i/o connectors. https://beam.apache.org/documentation/io/connectors/. (Accessed on 2024-09-24).

Beck (2024). Beckhoff new automatisation technology. https://www.beckhoff.com/en-en/products/automation/twincat/. (Accessed on 2024-09-24).

Chen, P. P. (1976). The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36.

DTP (2024). Eclipse data tools platform. https://projects.eclipse.org/projects/tools.datatools. (Accessed on 2024-09-24).

EMP (2024). Eclipse modeling project. https://projects.eclipse.org/projects/modeling. (Accessed on 2024-09-24).

Hoyer-Klick, C., Blesl, M., von Bremen, L., Frey, U., Gainnousakis, A., Hülk, L., Kronshage, S., Kuckertz, P., Lohmann, G., Muschner, C., Pehl, M., and Schroedter-Homscheidt, M. (2023). Fair data in energy systems analysis. In *Proceedings of the International Conference on Energy Meterology*.

Jin (2024). Template designer documentation. https://jinja.palletsprojects.com/en/3.0.x/templates/. (Accessed on 2024-09-24).

Mako (2024). Mako templates for python. https://www.makotemplates.org/. (Accessed on 2024-09-24).

MWE (2024). Eclipse modeling workflow engine. https://projects.eclipse.org/projects/modeling.emf.mwe. (Accessed on 2024-09-24).

ODC (2024). Open data connector. https://www.dataspaces.fraunhofer.de/en/software/connector/open_data_connector.html. (Accessed on 2024-09-24).

OMG (2011). Unified modeling language™ (uml®).

Ont (2024). Ontology (information science). https://en.wikipedia.org/wiki/Ontology_(information_science). (Accessed on 2024-09-24).

Schmidt, A., Koubaa, M. A., Schweikert, J., Stucky, K.-U., Süß, W., and Hagenmeyer, V. (2023). Zeitgeist - A Generic Tool Supporting the Dissemination of Time Series Data following FAIR Principles. In *Proceedings of the International Conference on Knowledge Management and Information Systems*. Insticc, SCITEPRESS.

Schmidt, A. and Tobias, M. (2024). Web Components for Database Developers. In *Proceedings of the Sixteenth International Conference on Advances in Databases, Knowledge, and Data Applications*. ThinkMind.

Schweikert, J., Stucky, K.-U., Süß, W., and Hagenmeyer, V. (2023). A photovoltaic system model integrating fair digital objects and ontologies. *Energies*, 16(3).

SHACL (2017). Shapes Constraint Language (SHACL). (Accessed on 2024-09-24).

Smarty (2024). Smarty template engine. https://www.smarty.net/. (Accessed on 2024-09-24).

Soiland-Reyes, S., Goble, C., and Groth, P. (2024). Evaluating fair digital object and linked data as distributed object systems. *PeerJ Computer Science*, 10:e1781.

Stahl, T. and Völter, M. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, Chichester, UK.

Twig (2024). Twig for template designers. https://twig.symfony.com/doc/3.x/templates.html. (Accessed on 2024-09-24).

Vincent, W. S. (2019). *Django for Professionals: Production websites with Python & Django*. Independently published.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., et al. (2016). The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3.

Zaninotto, F. and Potencier, F. (2008). *The Definitive Guide to symfony*. apress.