information

Article

A Methodology for Integrating Hierarchical VMAP-Data Structures into an Ontology Using Semantically Represented Analyses

Philipp Spelten [1,2], Morten-Christian Meyer [1], Anna Wagner [2], Klaus Wolf [1] and Dirk Reith [1,3,*]

1 Fraunhofer-Institute for Algorithms and Scientific Computing SCAI, Schloss Birlinghoven, 53757 Sankt Augustin, Germany; philipp.spelten@phd.h-brs.de (P.S.); klaus.wolf@scai.fraunhofer.de (K.W.)
2 PROSTEP AG, Dolivostr. 11, 64293 Darmstadt, Germany; anna.wagner@prostep.com
3 Institute of Technology, Resource and Energy-Efficient Engineering (TREE), Bonn-Rhein-Sieg University of Applied Sciences, Grantham-Allee 20, 53757 Sankt Augustin, Germany
* Correspondence: dirk.reith@h-brs.de

Abstract: Integrating physical simulation data into data ecosystems challenges the compatibility and interoperability of data management tools. Semantic web technologies and relational databases mostly use other data types, such as measurement or manufacturing design data. Standardizing simulation data storage and harmonizing the data structures with other domains is still a challenge, as current standards such as the ISO standard STEP (ISO 10303 "Standard for the Exchange of Product model data") fail to bridge the gap between design and simulation data. This challenge requires new methods, such as ontologies, to rethink simulation results integration. This research describes a new software architecture and application methodology based on the industrial standard "Virtual Material Modelling in Manufacturing" (VMAP). The architecture integrates large quantities of structured simulation data and their analyses into a semantic data structure. It is capable of providing data permeability from the global digital twin level to the detailed numerical values of data entries and even new key indicators in a three-step approach: It represents a file as an instance in a knowledge graph, queries the file's metadata, and finds a semantically represented process that enables new metadata to be created and instantiated.

Keywords: ontology; simulation process; data management; CAE metadata structures; semantic technologies

check for updates

Citation: Spelten, P.; Meyer, M.-C.; Wagner, A.; Wolf, K.; Reith, D. A Methodology for Integrating Hierarchical VMAP-Data Structures into an Ontology Using Semantically Represented Analyses. *Information* **2024**, *15*, 21. https://doi.org/10.3390/info15010021

Academic Editor: Katsuhide Fujita

Received: 13 November 2023
Revised: 21 December 2023
Accepted: 27 December 2023
Published: 29 December 2023

1. Introduction

Simulations are used in modern engineering to model physical effects and improve a product or machine. Simulation Process and Data Management (SPDM) requires (a) archiving the accruing data, (b) linking various simulation types implemented by different software vendors, and (c) transferring data between the simulation and design domains.

Standardized data formats, such as the ISO 10303 "Standard for the Exchange of Product model data" (STEP), play a major role in archiving [1]. To transfer data between different tools, most software is compatible with vendor-specific formats. However, this is not the case for all vendors and is tedious to implement for specialized software developed by Small and Medium Enterprises (SMEs) themselves. Therefore, the industrial standard "Virtual Material Modelling in Manufacturing" (VMAP) offers a vendor-neutral format and an I/O library to link simulation chains, including modeling information [2,3].

On the other hand, linking data between disciplines, particularly linking simulation data to design or measurement data, remains a challenge for SPDM due to the lack of data formats and integration methods. A proxy for this challenge is the simulation chain modeling a blow molding process. The process is shown in Figure 1 [2,4]. Here, VMAP data are successfully used for data transfer and archiving. However, data management capabilities and links to other disciplines are still lacking in this as well as other standardization approaches [2,5].

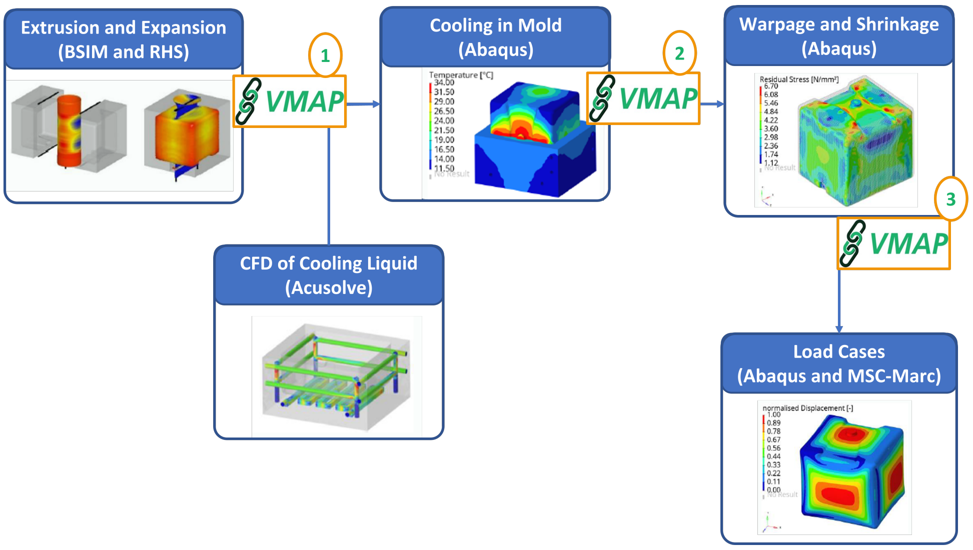

Extrusion and Expansion (BSIM and RHS)
Cooling in Mold (Abaqus)
Warpage and Shrinkage (Abaqus)
CFD of Cooling Liquid (Acusolve)
Load Cases (Abaqus and MSC-Marc)
VMAP
1
2
3

Figure 1. The simulation chain of the VMAP blow molding use case for a cubic tank. Between simulation tools, data often have to be converted to use the outputs of one tool as inputs for another. Data are transferred in VMAP files and translated to vendor-specific formats using the I/O library [6,7].

The solution we propose is realizing interoperability by harmonizing each data source's metadata schema and data access methods. Here, metadata refer to data that are relevant outside of the simulation domain, such as core results, numerical parameters, and boundary conditions, or the engineer responsible. This is where an open framework, especially a semantic information model, comes into play.

This article introduces a software design for SPDM implementers based on a semantic information model. VMAP data are used as a proxy for semantic data management and automated knowledge graph population. During this research, the following were developed:

- a set of requirements for Ontology-Based Data Access (OBDA) of bulk data;
- mechanisms that upload metadata and automatically query additional information from the bulk data;
- queries for additional metadata using semantically representing routines that process the bulk data and produce metadata;
- semantic routine management that can be expanded from data processing to manufacturing and management processes.

To this end, current research approaches and their relation to this work are introduced (Section 2), the requirements imposed by handling simulation data in digital twins are clarified (Section 3), and a strategy for semantic definitions is developed (Section 4.1). The results include a method for information retrieval from the semantic framework (Section 4.2) and a prototype of the method (Section 4.3). Finally, the capabilities and limitations of the presented method and implications for future research are discussed (Section 5).

2. Standardization and Semantic Technologies for Simulation Data Management

This section introduces major efforts made in the simulation data standardization and explores its current shortcomings. The standardized data models contain a closed-world assumption (CWA). Their files are therefore implicit, i.e., require knowledge of the domain specifications, and have thus no reasoning capabilities. To alleviate these shortcomings, semantic technologies for SPDM are used in this work and are described in this section. Semantic technologies keep an open-world assumption (OWA) and their files do contain explicit knowledge, making them expandable and reasonable. Table 1 compares additional aspects of the different methods introduced in this section.

Table 1. The different methods considered for SPDM.

	Domain Knowledge	**Data Access**	**Storage Logic**
STEP	Application Protocol (AP) [8]	Part21 [9] or binary file [10]	ASCII file
VMAP	General Specifications [2]	Standard Specifications [3]	HDF5 file [11]
Data pipelines to knowledge graphs	Ontology	Mapping-based OBDA	Relational database or supported proprietary formats
This work	Ontology	Procedural knowledge in semantically represented access methods	Semantic knowledge graph and HDF5 data

2.1. Standardization of SPDM

2.1.1. The ISO Standard STEP

ISO 10303 "Standard for the Exchange of Product model data" (STEP) [12] was developed for the long-term archiving of product development data in safety-critical engineering disciplines such as aviation, the automotive industry, and defense. It offers a standardized software-independent description of product data throughout the product life cycle, providing data modeling and storage definitions for large portions of Product Lifecycle Management (PLM). In ISO 10303-209 [1,13], simulation data were included to allow SPDM to be integrated into PLM. However, it falls short in important aspects of data management:

- The data model requires tedious and lengthy standardization processes for new modules due to the CWA: only defined classes are accepted.
- The data definition varies fundamentally due to the different granularity of design and simulation data.
- Simulation data can either be referenced as a whole file or fully stored in a STEP file. Referencing, however, lacks the possibility of linking to key results within the model, while STEP file formats require large amounts of storage space as described in Section 3.

2.1.2. The VMAP CAE Data Interface Standard

Contrary to the strict ISO standardization process, a standardization community from different industry branches recently developed the industrial standard "Virtual Material Modelling in Manufacturing" (VMAP) to provide data transfer between simulation tools. VMAP offers a vendor-neutral data format and a library of I/O routines to link chains of various simulation data formats [2].

The VMAP standard is based on Hierarchical Data Format (HDF5), a widely accepted implementation platform for many I/O-related applications [11]. Data were aligned into groups defined in the VMAP standard specifications [3] and commonalities were identified among many software packages to create the standard. Keeping this in mind, the VMAP storage structure defines the four main groups that form the essence of any simulation. Computer Aided Engineering (CAE) data are sorted into datasets and attributes within four main groups [3]:

1. GEOMETRY contains spatial information on nodes and elements.
2. VARIABLES stores physical quantities referenced to the nodes and elements.
3. In SYSTEM, the simulation parameters such as coordinate system and integration method are defined.

4. MATERIAL contains material information stored in different tables, which can be imported or shared across files.

Generally, data formats for SPDM must be compatible with varying material assumptions, boundary conditions, and numerical parameters simulation tools [14]. As VMAP is capable of this, it has been well-accepted within and outside of the standardization community. However, alignment methods to other product data should be included for full SPDM capabilities, such as design data, measurement data, and subjective intent capturing [15,16].

As a consequence, semantics should be added to the VMAP standard. Currently, no similar research is known to the authors.

2.2. Semantic Technologies for SPDM

Ontologies structure concepts in terms of meaningful relations (semantics). Knowledge graphs with node–edge–node structures (triples) incorporate semantic definitions in data schemas called ontologies. These graphs enable additional information (inferences) to be derived (reasoned) from the explicitly stated (asserted) relations. Semantic technologies promise to provide the foundation for a flexible and expandable data management system [17,18].

A noteworthy alternative is the addition of links to the VMAP standard. Generally, the HDF5 format supports hard links (to a physical address within a file) and symbolic links (a string for links within and outside of the file) [11]. This would partially enable SPDM capabilities for simulation files. It would, however, remain restricted to HDF5 files and require a translation or reference method for the integration into PLM systems. Conversely, ontologies are already used for PLM system development [19,20].

Two main issues need to be addressed in order to make good use of ontologies for data management. First, the mapping and interrelation between all domain-specific schemas need to be defined in a common ontological core framework. Second, the semantic resolution and a corresponding method for using the ontology within and beyond this resolution are necessary to achieve data permeability in practice. In the following, we relate our work to other research results.

2.2.1. Ontological Core Framework

The first point of a common ontological core framework is provided by MpCCI Ontologies for Digital Twins (smartMpCCI), developed by two of this paper's authors [21,22]. This ontology combines process-centered descriptions with data resources from various disciplines to align digital twin assets with simulation models and data analytics.

Other simulation ontologies are more restricted to a system-level perspective [23], which does not allow for the required integration of resolved physics. On the other hand, application-focused ontologies for digital twin systems often capture only the system or event-level [24]. Moreover, both generally struggle on a semantic level to reconcile simulation with the various Industry 4.0 standards (for plant planning and operation) [25].

Recently, Singh et al. [26] developed a central knowledge base in which databases have one-to-one mappings between the ontological relations and the data tables. However, this is only demonstrated for factory data acquisition, not simulation data, underlining that digital twin data management is an ongoing challenge in which simulation is seldom addressed.

The Physics-based Simulation Ontology (PSO) by Cheong and Butscher [27] extends the Basic Formal Ontology (BFO) [28] to simulation data. It explicitly models the physics and qualities of spatiotemporal assets in both the physical domain and the simulation domain. Underlying physical equations, discretization, and boundary conditions can be related to their physical equivalents. However, the PSO falls short of addressing how to store and handle the corresponding simulation data resources. The schema of smartMpCCI, on the other hand, is comparatively limited, focusing on the digital twin context. Therefore, harmonizing PSO and smartMpCCI seems recommendable for future studies.

2.2.2. Ontological Data Integration

This study is an original contribution to semantic resolution and corresponding integration methods. It presents a method for Ontology-Based Data Access (OBDA), which is novel to the authors' best knowledge. Four key aspects of Ontology-Based Data Access (OBDA) are relevant to evaluate the state of the art:

1. Bulk data must be stored in an efficient format. This may be propriety or semantic.
2. To store data efficiently, it may be integrated to different extents.
3. Data access must be defined if data are not fully imported into the database.
4. An ontology, i.e., a domain model, is required for full reasoning capabilities and must be maintained.

OntoDB is a leading data integration system based on ontologies [29], in which data storage and schema are inseparable in the same database. The data are migrated into a performance-optimized "state table per class" structure, making the ontology the database schema. This does, however, not leave the resolved data in its optimal database (binary and distributed), which is a vital requirement of most digital twins.

The separation of the ontology and the database was later shown to work well on the NoSQL database MongoDB [30] and OntoP [31]. Here, an ontology describes the knowledge domain, while an access interface stores the semantic concepts in a form that can be translated into the database queries. The database and its contents themselves are not semantically described, but rather the queries that can be addressed to the database. This approach is similar to the non-ontological Representational State Transfer (REST) Application Programming Interfaces (APIs) [32]. For hierarchical data, this raises the question of how to separate database-structure-specific semantics from the knowledge domain. Additionally, the data interface must be easily adapted to knowledge domain updates. This paper proposes such a separation using a semantic representation of extraction routines.

The SPARQL Protocol And RDF Query Language [33] (SPARQL)-Anything project [34] separates the storage-specific semantics from the knowledge domain by parsing data formats to Resource Description Framework [35] (RDF) and accesses the actual data contents with property functions that are an extension to a SPARQL engine. While this approach does not contain domain-specific mappings, Steindl et al. [36] implemented an OBDA method for the machine operation data standard Open Platform Communications (OPC) Unified Architecture [37,38] (OPC-UA). Both methods vary from our study due to the complete translation of the data model to RDF-triples without semantic representation of the property functions themselves, thus keeping the data structure non-ontological and removing the OWA. However, reconciling the approaches could be a fruitful combination of two important types of assets for digital twins.

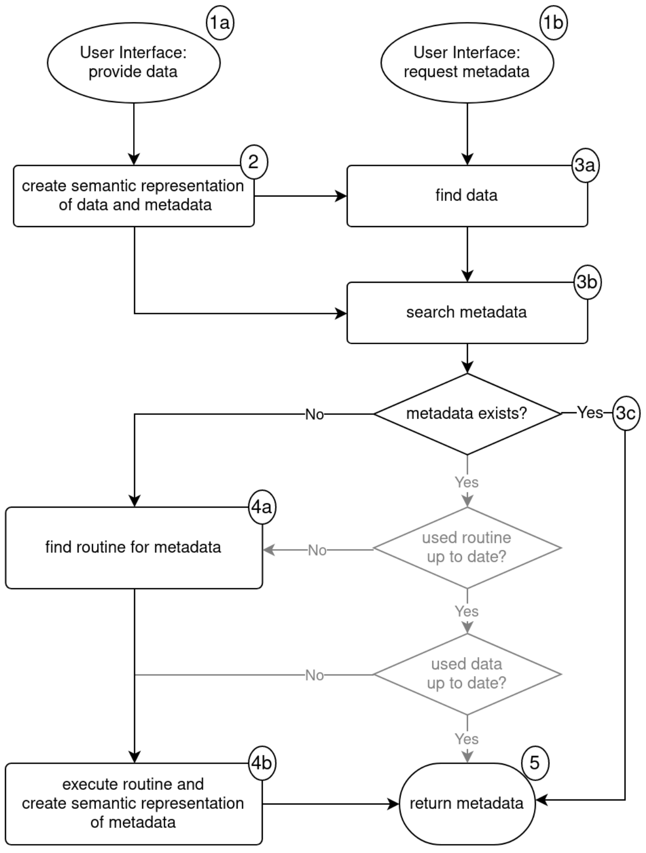
Each of these data access methods uses ontologies for mappings of some sort. Numerous possible mechanisms exist to create and update the joint knowledge base. With the separation between the ontology-structured metadata and the databases themselves, a knowledge graph needs to be maintained. RDF mapping languages such as the RDB to RDF Mapping Language (R2RML) [39] express custom mappings between relational databases. Furthermore, Leshcheva and Begler [40] developed an automated ontology population. As such, these methods are restricted to semi-structured data with additional semantics in data source ontologies, contrary to the hierarchical VMAP format. However, the HDF5 files may be translated to a relational format and thus imported into the knowledge graph. While this may be possible, it requires multiple mappings, creates large amounts of in the process, and still leaves the question of domain knowledge definitions in an ontology. Therefore, this approach was not applied in this work but should be investigated in future studies.

3. Software Requirements

Considering the described state of the art, we require the following key features of modern ontology-based data management:

- Semantic access to simulation data is necessary due to the described digital twin context. Simulation data, including all data values, must be found from a semantic access point.
- Simulation data are too large for a one-to-one mapping to a semantic framework, as is the current practice in OBDA [26,29]. Storage space is already a significant issue for SPDM, especially in dynamic simulations. Storing it in triples can be expected to take up a multiple amount of storage space, while the knowledge gained may be minimal. This is due to the clear-text nature of semantic formats and the inefficiency of RDF in storing ordered lists or even arrays, as structural relations would be repeatedly stated [18].
  The extra space is easily demonstrated when comparing the size of a use case example for STEP Application Protocol 209 “Multidisciplinary analysis and design” in ISO 10303-209:2014 (AP209) [41]. The recreated test case file in VMAP takes up only 176 kB of disk space, much less than the AP209 test file with 4278 kB. An Ontology Web Language [42] (OWL) instances file (translated using ExpressToOwl [43]) requires 43,652 kB. Therefore, the respective data must remain in its optimal storage format to minimize storage space while maximizing interoperability.
- Access methods must be easily available to create new metadata without manual labor. Currently, the post-processing of VMAP files requires mapping back and forth between vendor-specific formats. This is computationally intensive and unnecessary for well-defined post-processing steps and easily automatable tasks that can be bundled in a batch process. The requirement we impose is automating data processing to obtain the desired information. For this, very performative routines are available in open-source software packages for the binary HDF5 structure of VMAP.

To fulfill the requirements, the following actions must be executed by the software solution (see also the UML® [44] activity diagram in Figure 2):



1a
User Interface: provide data
1b
User Interface: request metadata
2
create semantic representation of data and metadata
3a
find data
3b
search metadata
metadata exists?
No
Yes
3c
Yes
used routine up to date?
No
4a
find routine for metadata
Yes
used data up to date?
No
Yes
4b
execute routine and create semantic representation of metadata
5
return metadata

Figure 2. The proposed UML® activity diagram to be implemented in an SPDM including binary data. Grey: The possible version control loop.

1. providing a file and request metadata via a user interface;
2. uploading vital information from the file to the ontology down to a certain depth;
3. searching the requested metadata and returning it if it is available;
4. finding the routine required to create the metadata if it is unavailable, executing the routine, and adding the metadata to the knowledge graph;
5. returning the metadata to the user.

Additionally, a version control loop may be added to find out whether the metadata and its routine are up to date.

4. Software Architecture and Prototype

The software requirements are put into practice in a prototype software architecture with three significant characteristics:

1. All data can be accessed via the knowledge graph (see Section 4.3). A GUI is implemented to show the current capabilities.
2. The knowledge graph is populated only with such information required for new metadata to be queried and found or created with efficient methods (Section 4.1), thus significantly reducing the required storage space.
3. Access methods are semantically available in the knowledge graph and can be easily managed (see the process ontology in Section 4.1 and the access layer in Section 4.3).

This section presents the characteristics of the developed prototype. First, the ontologies are introduced; second, the information access from the knowledge graphs is discussed; third, the method implementation is presented; finally, the method's capabilities are shown. Figure 3 shows an abstract representation of the architecture.

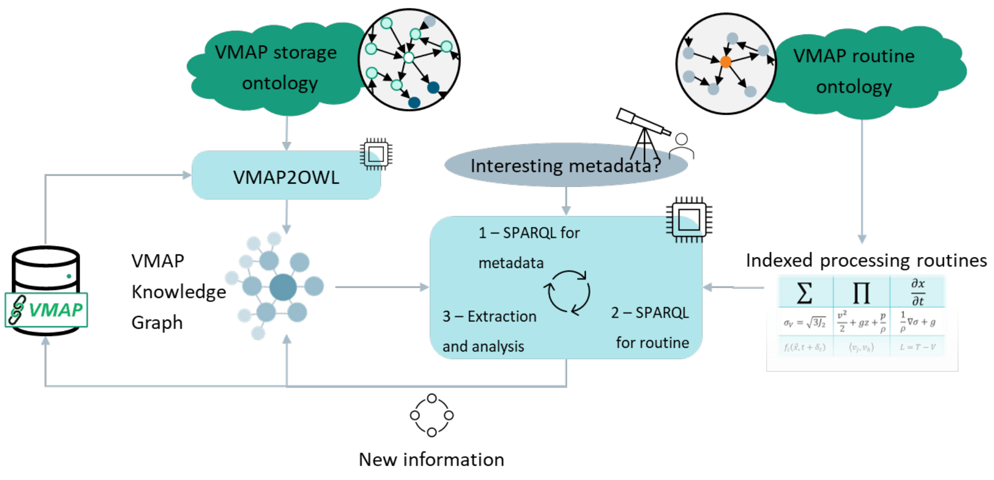


Figure 3. Abstract representation of the developed architecture. Two ontologies form the basis for two knowledge graphs. Processing routines are manually indexed. The VMAP graph is automatically created from a VMAP file using the VMAP2OWL process. When metadata are requested, information from both knowledge graphs is used to extract new information and write it to the VMAP file and graph.

4.1. Semantic Definitions

Two ontologies are used: one for knowledge graph population (storage ontology) and one for querying the database from within the ontology (process ontology). These are separated here, as both ontologies may be used independently for their respective purposes. While the storage ontology is intended to be format-specific, the process ontology uses generic classes and format-specific classes to organize data access across formats. Both ontologies are publicly available at https://gitlab.scai.fraunhofer.de/vmap-onto/vio-method/-/tree/main/vmapontologies (accessed on 26 December 2023). The RDF notations of both ontologies' top-level classes are provided in Appendix B.

4.1.1. Storage Ontology

An ontology of the storage structure is required to find and reference the desired information. Recently, an ontology to represent the mereology (part-of structure or storage structure) of the VMAP standard specifications [3] was developed. It is aligned with the smartMpCCI ontology currently being developed at Fraunhofer-Institute for Algorithms and Scientific Computing (SCAI) [21,22]. It relates storage levels (groups, attributes, datasets, and dataset columns) to their parent directory and contents via an `isStoredIn` and an inverse `stores` object property restriction. The semantic integration is capped at the dataset column level, for which only value type and column number or column name are specified via data property restrictions. The top-level classes of the VMAP storage ontology are shown in Figure 4.

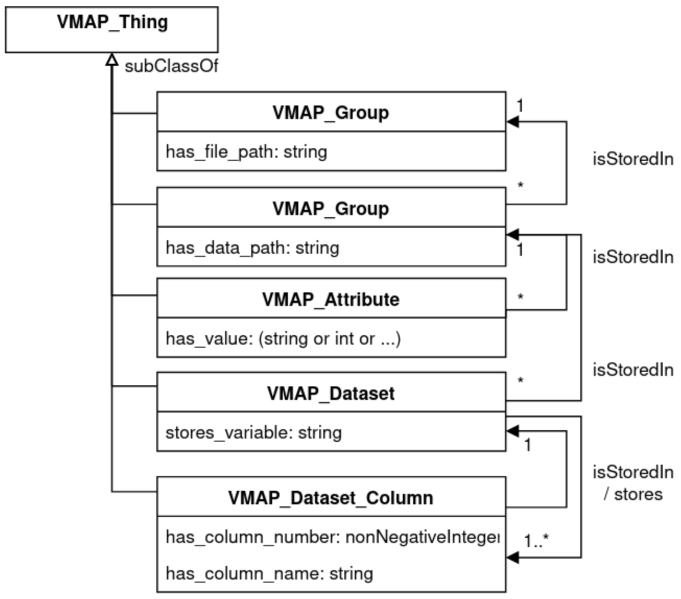


Figure 4. Top-level classes of the VMAP storage ontology. The hierarchical structure is implemented using `isStoredIn` and the inverse `stores` relations. '*' signifies an object property restriction with arbitrary cardinality, i.e., `someValuesFrom`.

4.1.2. Process Ontology

A `Routine` is a subclass of a `BFO:Process` [28]. It can be any data processing routine, from a simple line of code to a batch process. An ontology of the metadata routines provides information on the inputs and results of the routines, such that a routine can be found based on the requested metadata. The routines can be generic for HDF5 files or datasets, specific for VMAP files or datasets, or tailored to any other specified file type. Hence, a `VMAP_Routine` class should be defined with a `hasOutput` object property restriction pointing to a `VMAP_Metadata` class and `hasInput` pointing to a `VMAP_File` or `VMAP_Dataset` class. The top-level definitions of the VMAP routines ontology are given in Figure 5.

This architecture forms the basis of querying semantically represented data. Additional semantics can be provided by general ontologies such as PSO, for example, understanding the stored quantity and thus including more profound knowledge of the metadata. Additionally, the schema can be integrated into a broader digital twin context by aligning it with the BFO or smartMpCCI by defining `Routine subClassOf BFO:Process` [28] or `Routine subClassOf smartMpCCI:Simulation_Aspect` [21]. Such linkages should be further explored in future research to fully exploit the advantages of ontological data management.

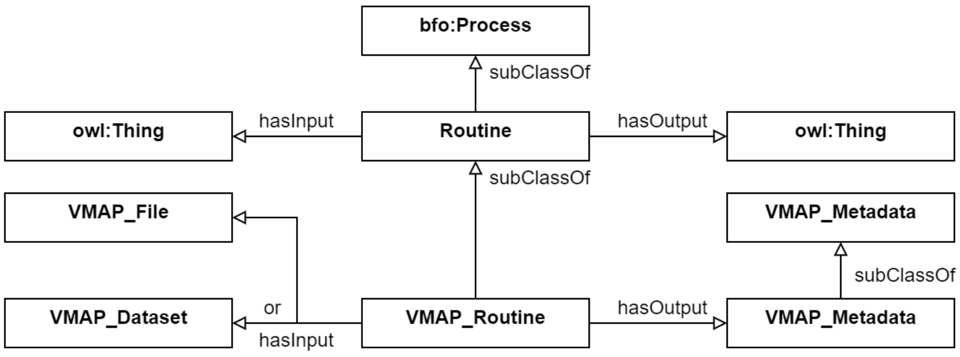


Figure 5. The top-level definitions of the VMAP routines ontology.

4.2. Querying Information in Three Steps

The core concept of information access is demonstrated here by querying basic storage relations. The knowledge graph is assumed to be instantiated, containing the explicit metadata of a given VMAP file, and the required processing routines have been indexed for automated execution. Three queries are executed to search the metadata and find the required routine. The code listings in RDF Schema [45] (RDFS), OWL, and SPARQL notations are given in Appendix C.

1. Find the File

Instead of the user providing the file, some information about the file can be given to run a SPARQL query. This can, for example, be the data property `stores_variable` “`temperature`” of a column within a dataset. Figure 6 demonstrates the queried graph.

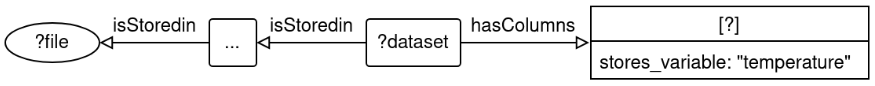


Figure 6. Schematic of the query to find a file containing a dataset for a specific variable. Oval: The queried instance. Rectangle: Instances and classes. `?` represents a variable instance or class. `...` represents a chain of `isStoredIn` edges, marked by `:isStoredIn+` in the listing.

2. Find the Metadata

Some metadata can be available within the file or dataset. The desired metadatum can easily be found via a data property, as shown above, or via its type (Figure 7 demonstrates the queried graph).

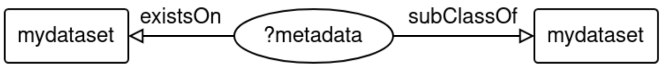


Figure 7. Schematic of the query to find a metadata instance of a specific class type and related to a specific dataset. Oval: The queried instance. Rectangle: Instances and classes. `?` represents a variable instance or class.

3. Find the Metadata Routine

Individuals of `Routine` can have relations to `Metadata`, and `File` individuals and can be queried directly. Figure 8 demonstrates the knowledge graph and related SPARQL query.

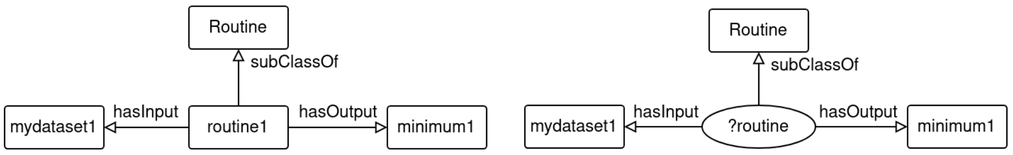


Figure 8. Schematic of the knowledge graph (**left**) and query (**right**) using `Routine` instances for metadata routine search. Oval: The queried instance. Rectangle: Instances and classes. `?` represents a variable instance or class.

However, instantiating a specific routine for each individual input and output may not be desirable, as it leads to new individuals for each metadata query. Instead, `routine1` should be an instance of a `Routine` subclass, as shown in Figure 9. As class restrictions are implemented by making the class a subclass of the restriction, the routine must be queried via the `inputrestriction` and `outputrestriction`. As the `hasInput` and `hasOutput` object properties may be refined, any of their subproperties suffice. Therefore, the queried subgraph is more complex, as can be seen in Figure 10.

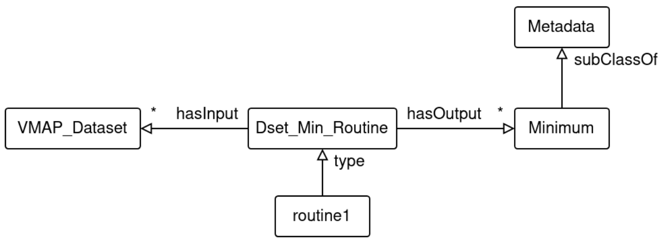


Figure 9. Schematic of the knowledge graph using `Routine` subclasses for metadata routine search. Rectangle: Instances and classes. * represents an `owl:someValuesFrom` object property restriction.

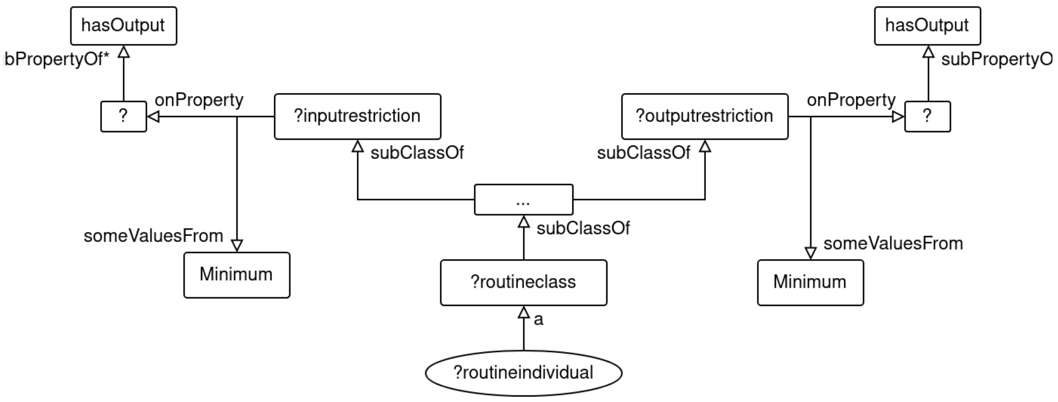


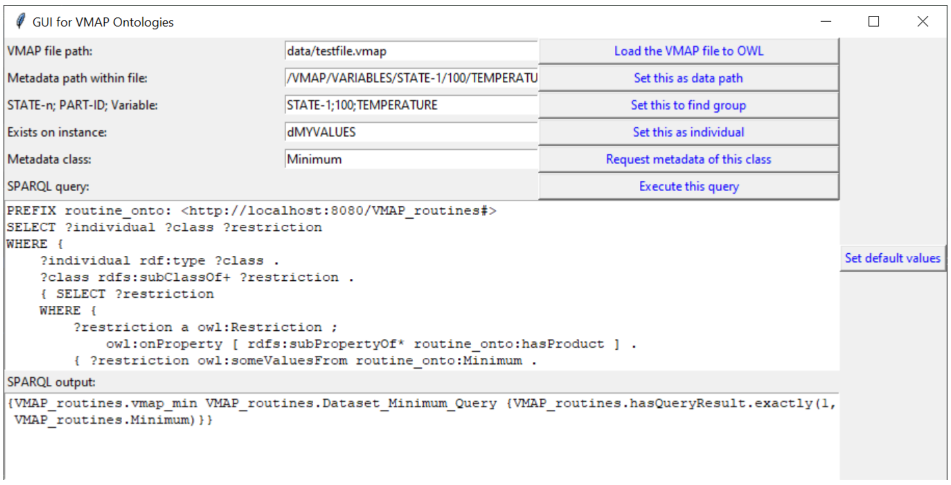
Figure 10. Schematic of the query using `Routine` subclasses for metadata routine search. Oval: The queried instance. Rectangle: Instances and classes. `subPropertyOf*` represents a chain of one or more edges of `subPropertyOf`. `?` represents a variable instance or class.

4.3. Software Design

The database queries and data processing routines are executed by a software framework described in this section. The working prototype is publicly available at https://gitlab.scai.fraunhofer.de/vmap-onto/vio-method/-/tree/main/vmapintegration (accessed on 26 December 2023), which provides the neighboring `vmapontologies` directory. Next to `data` (VMAP files for testing) and `tests` (scripts implementing the tests described in Section 4.4), `src` houses the implemented classes. They are divided into three layers: a presentation layer with the user interface, an application layer for the application-specific classes and methods, and a (data) access layer to load and access the files [46,47]. The UML® diagram in Figure A1 in the Appendix A demonstrates the class relations.

4.3.1. Presentation Layer

The presentation layer handles user inputs and passes them to available methods. It is separated into a `UserInterface` class and a `Control` class, taking over the functions of View and Control in a Model–View–Control (MVC) architecture. The `UserInterface` class is loaded directly from the access point (`main`). Figure 11 shows the GUI implemented in this layer.



VMAP file path:	data/testfile.vmap
Metadata path within file:	/VMAP/VARIABLES/STATE-1/100/TEMPERATU
STATE-n; PART-ID; Variable:	STATE-1;100;TEMPERATURE
Exists on instance:	dMYVALUES
Metadata class:	Minimum

SPARQL query:

```
PREFIX routine_onto: <http://localhost:8080/VMAP_routines#>
SELECT ?individual ?class ?restriction
WHERE {
    ?individual rdf:type ?class .
    ?class rdfs:subClassOf+ ?restriction .
    { SELECT ?restriction
    WHERE {
        ?restriction a owl:Restriction ;
            owl:onProperty [ rdfs:subPropertyOf* routine_onto:hasProduct ] .
        { ?restriction owl:someValuesFrom routine_onto:Minimum .
```

SPARQL output:

```
{VMAP_routines.vmap_min VMAP_routines.Dataset_Minimum_Query {VMAP_routines.hasQueryResult.exactly(1,
 VMAP_routines.Minimum)}}
```

Figure 11. GUI of the prototype. Only one of either data path, group, or individual is needed to find the minimum of the desired dataset. The derived SPARQL query and its results are displayed below.

4.3.2. Application Layer

In the application layer, two classes are implemented.

The class `VmapOwlConstructor` populates the knowledge graph in the method `vmap2owl`. The method uses the sub-methods `create_all_groups` and `create_all_attributes`. The received and generated information is stored in fields such as `vmap_file`, `vmap_ontology`, `vmap_instances`, `ID`, and others.

The class `MetaData` calls the method `metadata_search` upon initialization, using a SPARQL query to check whether the requested metadata already exists in the knowledge graph. Here, the version control mentioned above can be added. If `metadata_search` is unsuccessful, it calls the `routine_search` method to find the relevant metadata routine and returns the information vital to importing and executing the routine. These two methods use the SPARQL queries suggested in Section 4.2.

`MetaData` has an `exists_on_instance` field pointing to the instance to which the metadatum applies. The field `metadata_class` is the class of which a metadatum is requested. During the `metadata_search` method, a `routine_instance` field is added. After executing the routine, the metadata instance and its value are added to the knowledge graph as a triple. Accordingly, the `metadata_instance` and the corresponding `value` fields are added to the class.

4.3.3. Access Layer

The routines found by `routine_search` are created as separate classes in the access layer. These can be housed by classes or created as independent methods. They are not imported unless required for the `metadata_search` method: for example, a method `VmapDatasetMinimum` inherits the minimum calculation method from its ancestor `ArrayMinimum`. This can be a simple NumPy function or a whole batch process.

4.3.4. Naming Conventions and Used Packages

The architecture requires naming conventions to be introduced throughout different development frameworks, as considerable amounts of information are stored as strings in OWL. In particular, the names of files or dataset paths, functions, and classes should be automatically linked or tested regularly.

Custom data types may be developed to achieve an integrated environment. For example, this would enable a data property of a `Metadata` or `VMAP_Dataset_Column` instance to be an HDF5 dataset, which otherwise cannot be stored efficiently in OWL. This can only be achieved using an integrated approach such as owlready2 in Python [48].

To realize the architecture in Python [49], four external packages are required for the user interface, ontologies, file access, and numeric calculations:

1. The classes of the prototype's presentation layer use the tkinter package, which provides functions for both the user interface and the control classes [50].
2. To handle ontologies, owlready2 comes with SPARQL functionalities and the Pellet and HermiT reasoners [33,48].
3. The h5py package can import HDF5 files, i.e., VMAP data [51].
4. NumPy uses HDF5 for storage and computing and provides efficient numerical methods [52].

4.4. Test Functions

Several test functions for the basic required functionalities have been created. They are available at https://gitlab.scai.fraunhofer.de/vmap-onto/vio-method/-/tree/main/vmapintegration/tests (accessed on 26 December 2023) and briefly introduced here.

- Requesting metadata should require a minimal amount of information about its object. Finding the referred dataset instance has been implemented for an owlready2 entity, the datapath of a dataset, the required state variable, and the HDF5 dataset itself. This has been tested in `test00existsoninstance.py`, `test01existsoninstancename.py`, `test02datapath.py`, `test03variable.py`, and `test04dataset.py`, respectively.
- When instantiating a file to the knowledge graph, duplicates must be identified. This is tested in `test05duplicatefile.py` based on ID parameters.
- If a metadata instance is already available, it must be found in the knowledge graph and directly returned, rather than repeatedly searching and executing the data processing routines. In `test06duplicatemetadata.py`, metadata instances are found with the same `existsOn` and `subClassOf` specifications as requested.
- More complex metadata, i.e., the difference between two datasets, and translations of Cauchy stresses into a Von Mises criterion, are created in `test08difference.py` and `test09cauchytomises.py`, respectively.
- A more complex blow molding simulation result from an industrial use case for VMAP [2] was instantiated and processed in `test07bsimresults.py`. The data for this test case is currently not publicly available.

The prototype has successfully completed all test functions.

5. Discussion and Outlook

This paper presents an ontology-based data management method for physics-based simulation chains built on the VMAP standard. General strategic decisions are based on literature and concrete requirements for VMAP use cases. The proposed architecture can automatically populate a knowledge graph with storage and metadata information of a VMAP file, query some simple metadata semantically, generate new metadata directly from the VMAP file using efficient routines, and add the acquired data to the knowledge graph.

The results of this research indicate a significant improvement of two key issues of an SPDM for hierarchical simulation data by the architecture suggested in this paper:

1. Without manual mapping, data can be directly analyzed on the binary VMAP file. Loading only vital information into the ontology minimizes the storage space footprint of the stored triples, and the bulk data remains in its optimal form.
2. New numerical methods and data analyses can be imported into the process ontology for knowledge management of the available routines and automatic use in the semantic framework.

This is achieved by separating the knowledge base from the data sources, using the data storage definitions for automatic knowledge graph creation, and semantically representing data retrieval and processing. The architecture has been realized as a prototype in Python and is provided in the supplementary information of this article.

Some limitations remain and should be considered and investigated in future research:

1. As outlined in Section 2, a widely accepted common ontological framework for OBDA of simulation data is yet to be developed. Harmonizing smartMpCCI and PSO appears to be a promising starting point.
2. The ontology models have, thus far, been created manually. This should be automated for the storage ontology to allow scalability to more formats and version updates. Some progress has been made regarding ontology generation from source code [53] or raw text [54] and should be considered.
3. The method is currently only implemented as a prototype and is not yet being used in an industrial context, which is a vital prerequisite for the method's success [5,14]. Currently, PLM methods and particularly SPDM methods lack standardized benchmarks, restricting the evaluation to expert assessments for individual industry scenarios [15].

For the method presented in this paper, at least three points should be implemented in an integrated framework and benchmarked against other methods:

1. The capabilities of OBDA can be fully exploited when aligning ontologies of different domains. This is an open field of research where progress has been made in recent years [55,56].
2. The computational overhead of the method depends on the search along the knowledge graph. This can be minimized using efficient semantic search engines [57]. The method may be limited to the simple routines tested in this paper. However, APIs can be used to automate integrated workflows efficiently.
3. The organizational overhead of SPDM methods typically remains dependent on the implementation of new routines and may limit the applicability [5,14].

Unfortunately, this extensive in-industry testing lies outside of the scope of this research paper.

In future work, we propose firstly refining and extending the semantics, secondly developing industry scenarios and benchmark cases for both SPDM and OBDA, and thirdly testing and validating them against traditional SPDM and OBDA methods. With further development, the architecture could form the basis of OBDA that can be extended to various data formats, processing routines, and application contexts when embedded in or mapped to a common semantic framework. It unites the advantages of binary data storage, knowledge management, and semantic inferences. SPDM development remains a significant challenge within digital twin technologies, for which this research offers a promising approach.

Supplementary Materials: The supplementary data are publicly available at https://gitlab.scai.fraunhofer.de/vmap-onto/vio-method (accessed on 26 December 2023).

Author Contributions: Conceptualization, P.S., M.-C.M. and D.R.; data curation, P.S.; formal analysis, P.S.; funding acquisition, K.W. and D.R.; investigation, P.S.; methodology, M.-C.M.; project administration, K.W.; resources, P.S.; software, P.S.; validation, P.S.; visualization, P.S.; writing—original draft, P.S.; writing—review & editing, M.-C.M., A.W., K.W. and D.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the supplementary material.

Acknowledgments: We thank Olaf Bruch (Reinold Hagen Stiftung) and Martin Striezel (PROSTEP AG) for fruitful discussions and advice about industry use cases. We are also grateful to Andre Oeckerath (Fraunhofer-Institute for Algorithms and Scientific Computing SCAI) for recreating the AP209 test file.

Conflicts of Interest: Author Dr. Anna Wagner was employed by the company PROSTEP AG. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

STEP	ISO 10303 “Standard for the Exchange of Product model data”
IoT	Internet of Things
SDM	Simulation Data Management
SPARQL	SPARQL Protocol And RDF Query Language [33]
SCAI	Fraunhofer-Institute for Algorithms and Scientific Computing
SQL-DB	Structured Query Language database
SQL	Structured Query Language
DB	database
MpCCI	Mesh-based parallel Code Coupling Interface
HDF5	Hierarchical Data Format
PSO	Physics-based Simulation Ontology
BFO	Basic Formal Ontology
OBDA	Ontology-Based Data Access
smartMpCCI	MpCCI Ontologies for Digital Twins
OWL	Ontology Web Language [42]
OPC-UA	Open Platform Communications (OPC) Unified Architecture [37,38]
RDF	Resource Description Framework [35]
AP209	STEP Application Protocol 209 “Multidisciplinary analysis and design” in ISO 10303-209:2014
SPDM	Simulation Process and Data Management
RDFS	RDF Schema [45]
CAE	Computer Aided Engineering
PLM	Product Lifecycle Management
SME	Small and Medium Enterprise
API	Application Programming Interface
OWA	open-world assumption
CWA	closed-world assumption

Appendix A. Class Diagram

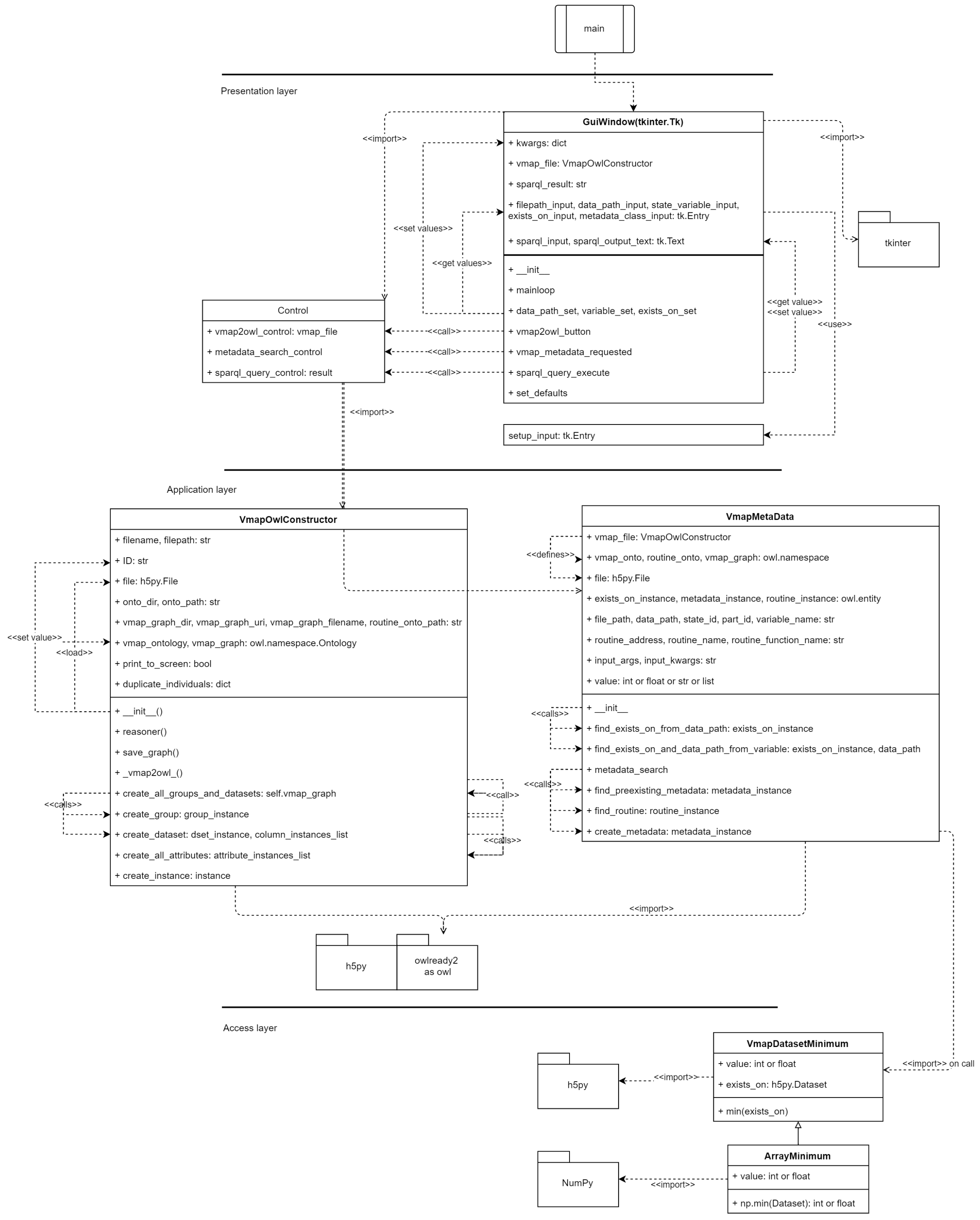

main
Presentation layer
GuiWindow(tkinter.Tk)
+ kwargs: dict
+ vmap_file: VmapOwlConstructor
+ sparql_result: str
+ filepath_input, data_path_input, state_variable_input, exists_on_input, metadata_class_input: tk.Entry
+ sparql_input, sparql_output_text: tk.Text
+ __init__
+ mainloop
+ data_path_set, variable_set, exists_on_set
+ vmap2owl_button
+ vmap_metadata_requested
+ sparql_query_execute
+ set_defaults
<<import>>
<<set values>>
<<get values>>
<<get value>>
<<set value>>
<<use>>
tkinter
Control
+ vmap2owl_control: vmap_file
+ metadata_search_control
+ sparql_query_control: result
<<call>>
<<call>>
<<call>>
setup_input: tk.Entry
<<import>>
Application layer
VmapOwlConstructor
+ filename, filepath: str
+ ID: str
+ file: h5py.File
+ onto_dir, onto_path: str
+ vmap_graph_dir, vmap_graph_uri, vmap_graph_filename, routine_onto_path: str
+ vmap_ontology, vmap_graph: owl.namespace.Ontology
+ print_to_screen: bool
+ duplicate_individuals: dict
+ __init__()
+ reasoner()
+ save_graph()
+ _vmap2owl_()
+ create_all_groups_and_datasets: self.vmap_graph
+ create_group: group_instance
+ create_dataset: dset_instance, column_instances_list
+ create_all_attributes: attribute_instances_list
+ create_instance: instance
<<set value>>
<<load>>
<<calls>>
<<call>>
<<calls>>
VmapMetaData
+ vmap_file: VmapOwlConstructor
+ vmap_onto, routine_onto, vmap_graph: owl.namespace
+ file: h5py.File
+ exists_on_instance, metadata_instance, routine_instance: owl.entity
+ file_path, data_path, state_id, part_id, variable_name: str
+ routine_address, routine_name, routine_function_name: str
+ input_args, input_kwargs: str
+ value: int or float or str or list
+ __init__
+ find_exists_on_from_data_path: exists_on_instance
+ find_exists_on_and_data_path_from_variable: exists_on_instance, data_path
+ metadata_search
+ find_preexisting_metadata: metadata_instance
+ find_routine: routine_instance
+ create_metadata: metadata_instance
<<defines>>
<<calls>>
<<calls>>
<<import>>
h5py
owlready2 as owl
Access layer
h5py
<<import>>
VmapDatasetMinimum
+ value: int or float
+ exists_on: h5py.Dataset
+ min(exists_on)
<<import>> on call
NumPy
<<import>>
ArrayMinimum
+ value: int or float
+ np.min(Dataset): int or float

Figure A1. UML® class diagram of the proposed architecture.

Appendix B. RDF Notations of Semantic Definitions

Appendix B.1. Top-Level Classes of the Storage Ontology

See Figure 4 for a graphical representation.

```
VMAP_File          subClassOf     VMAP_Thing
VMAP_Group         subClassOf     VMAP_Thing
VMAP_Group         isStoredIn     VMAP_file
VMAP_Group         has_data_path  xsd:string
VMAP_Attribute     subClassOf     VMAP_Thing
VMAP_Attribute     isStoredIn     VMAP_Group
VMAP_Attribute     has_value     (xsd:string or xsd:int or xsd:
    float or ...)
VMAP_Dataset       subClassOf     VMAP_Thing
VMAP_Dataset       isStoredIn     VMAP_Group
VMAP_Dataset       hasColumns     VMAP_Dataset_Column
VMAP_Dataset       stores_variable xsd:string
VMAP_Dataset_Column subClassOf     VMAP_Thing
VMAP_Dataset_Column isStoredIn     VMAP_Dataset
VMAP_Dataset_Column has_column_number xsd:nonNegativeInteger
VMAP_Dataset_Column has_column_name   xsd:string
```

Appendix B.2. Top-Level Classes of the Process Ontology

See Figure 5 for a graphical representation.

```
Routine        subClassOf bfo:Process
Routine        hasInput   owl:Thing
Routine        hasOutput  Metadata
VMAP_Routine   subClassOf Routine
VMAP_Metadata  subClassOf Metadata
VMAP_Routine   hasInput   VMAP_File or VMAP_Dataset
VMAP_Routine   hasOutput  VMAP_Metadata
```

Appendix C. SPARQL Notations of Semantic Queries

Appendix C.1. Finding the File

```
SELECT ?file WHERE {
  ?dataset :isStoredIn+ ?file
  ?dataset hasColumns [ :stores_variable "temperature" ] }
```

Appendix C.2. Finding the Metdata

```
SELECT ?metadata WHERE {
  ?metadata :existsOn  mydataset
  ?metadata rdf:type   Minimum }
```

Appendix C.3. Finding the Metadata Routine Using `Routine` Instances

Knowledge graph:

```
routine1 rdf:type  Routine
routine1 hasInput  mydataset1
routine1 hasOutput minimum1
```

SPARQL query:

```
SELECT ?routine WHERE {
  ?routine :hasInput  mydataset
  ?routine :hasOutput minimum }
```

Appendix C.4. Finding the Metadata Routine Using `Routine` Subclasses

Knowledge graph:

```
Minimum          subClassOf      Metadata
Dset_Min_Routine hasOutput  some Minimum
Dset_Min_Routine hasInput   some VMAP_Dataset
routine1         rdf:type        Dset_Min_Routine
```

SPARQL query:

```
SELECT ?routineindividual WHERE {
?routineindividual a                ?routineclass .
?routineclass      rdfs:subClassOf+ ?outputrestriction .
?routineclass      rdfs:subClassOf+ ?inputrestriction .
{ SELECT ?outputrestriction WHERE {
?outputrestriction a   owl:Restriction ;
  owl:onProperty   [ rdfs:subPropertyOf* :hasOutput ] ;
  owl:someValuesFrom :Minimum . } }
{ SELECT ?inputrestriction WHERE {
?inputrestriction a    owl:Restriction ;
  owl:onProperty   [ rdfs:subPropertyOf* :hasInput ] ;
  owl:someValuesFrom :Minimum . } } }
```

References

1. *ISO 10303-209:2014*; Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 209: Application Protocol: Multidisciplinary Analysis and Design. Standard, International Organization for Standardization: Geneva, Switzerland, 2014.
2. VMAP Project Consortium. VMAP. A New Interface Standard for Integrated Virtual Material Modelling in Manufacturing Industry. General Information. 2022. Available online: https://vmap.vorschau.ws.fraunhofer.de/content/dam/scai/vmap/VMAP_v100-General_Information.pdf (accessed on 14 September 2022).
3. VMAP Project Consortium. VMAP. A New Interface Standard for Integrated Virtual Material Modelling in Manufacturing Industry. Standard Specifications. 2022. Available online: https://vmap.vorschau.ws.fraunhofer.de/content/dam/scai/vmap/VMAP_v100-Standard_specifications.pdf (accessed on 21 November 2022).
4. Kärger, L.; Bernath, A.; Fritz, F.; Galkin, S.; Magagnato, D.; Oeckerath, A.; Schön, A.; Henning, F. Development and Validation of a CAE Chain for Unidirectional Fibre Reinforced Composite Components. *Compos. Struct.* **2015**, *132*, 350–358. [CrossRef]
5. Spelten, P. Bridging the Gap between Product and Simulation Data Management. An Analysis of the Needs and Possibilities in Industrial Engineering. Bachelor's Thesis, Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V., Munich, Germany, 2021. [CrossRef]
6. Spelten, P. Simulation Data Goes Ontology. Master's Thesis, Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V., Munich, Germany, 2023. [CrossRef]
7. Michels, P.; Bruch, O.; Gulati, P. ITEA VMAP - How the simulation workflow of blow moulded plastic parts benefits from the VMAP Interface Standard. In Proceedings of the NAFEMS World Congress VMAP Conference 2019, Quebec, QC, Canada, 17–20 June 2019.
8. *ISO 10303-14:2005*; Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 14: Description Methods: The Express-X Language Reference Manual. Standard, International Organization for Standardization: Geneva, Switzerland, 2005.
9. *ISO 10303-21:2015*; Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 21: Implementation Methods: Clear Text Encoding of the Exchange Structure. Standard, International Organization for Standardization: Geneva, Switzerland, 2015.
10. *ISO 10303-26:2011*; Industrial Automation Systems—Product Data Representation and Exchange—Part 26: Implementation Methods: Binary Representation of Express-Driven Data. Standard, International Organization for Standardization: Geneva, Switzerland, 2011.
11. The HDF Group. Hierarchical Data Format, Version 5. 1997. Available online: https://www.hdfgroup.org/HDF5/ (accessed on 5 December 2022).
12. *ISO 10303-1:1994*; Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 1: Overview and Fundamental Principles. Standard, International Organization for Standardization: Geneva, Switzerland, 1994.
13. AFNeT. AP209 Website. Available online: www.ap209.org/ (accessed on 16 January 2021).
14. Norris, M. *How to—Get Started with Simulation Data Management*; NAFEMS: Chicago, IL, USA, 2020; ISBN 978-1-83979-027-0.

15. Eigner, M.; Stelzer, R. *Product Lifecycle Management*; Springer: Berlin/Heidelberg, Germany, 2009. [CrossRef]
16. Turnitsa, C.; Padilla, J.J.; Tolk, A. Ontology for modeling and simulation. In Proceedings of the Winter Simulation Conference, Baltimore, MD, USA, 5–8 December 2010; pp. 643–651. [CrossRef]
17. Berners-Lee, T.; Fischetti, M. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*; DIANE Publishing Company: Darby, PA, USA, 2001; ISBN 978-1439500361.
18. Wagner, A. Linked Product Data: Describing Multi-Functional Parametric Building Products Using Semantic Web Technologies. Ph.D. Thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2020. [CrossRef]
19. Bruno, G.; Antonelli, D.; Villa, A. A reference ontology to support product lifecycle management. *Procedia CIRP* **2015**, *33*, 41–46. [CrossRef]
20. El Kadiri, S.; Kiritsis, D. Ontologies in the context of product lifecycle management: State of the art literature review. *Int. J. Prod. Res.* **2015**, *53*, 5657–5668. [CrossRef]
21. Meyer, M.C.; Yu, Z.; Delforouzi, A.; Roggenbuck, J.; Wolf, K. *Ontologies for Digital Twins in Smart Manufacturing*; Whitepaper; Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.: Munich, Germany, 2020.
22. Meyer, M.C.; Delforouzi, A.; Schlimper, R.; John, M.; Link, T.; Koster, D.; Summa, J.; Krauß, C. A digital twin for lightweight thermoplastic composite part production. In Proceedings of the NAFEMS World Congress 2021, Online, 25–29 October 2021.
23. Grolinger, K.; Capretz, M.A.M.; Marti, J.R.; Srivastava, K.D. Ontology – based Representation of Simulation Models. In Proceedings of the Twenty-Fourth International Conference on Software Engineering and Knowledge Engineering (SEKE), San Francisco, CA, USA, 1–3 July 2012; pp. 432–437.
24. Bao, Q.; Zhao, G.; Yu, Y.; Dai, S.; Wang, W. The ontology-based modeling and evolution of digital twin for assembly workshop. *Int. J. Adv. Manuf. Technol.* **2021**, *117*, 395–411. [CrossRef]
25. Nagy, L.; Ruppert, T.; Abonyi, J. Ontology-Based Analysis of Manufacturing Processes: Lessons Learned from the Case Study of Wire Harness Production. *Complexity* **2021**, *2021*. [CrossRef]
26. Singh, S.; Shehab, E.; Higgins, N.; Fowler, K.; Reynolds, D.; Erkoyuncu, J.A.; Gadd, P. Data management for developing digital twin ontology model. *Proc. Inst. Mech. Eng. Part J. Eng. Manuf.* **2021**, *235*, 2323–2337. [CrossRef]
27. Cheong, H.; Butscher, A. Physics-based simulation ontology: An ontology to support modeling and reuse of data for physics-based simulation. *J. Eng. Des.* **2019**, *30*, 655–687. [CrossRef]
28. Arp, R.; Smith, B.; Spear, A.D. *Building Ontologies with Basic Formal Ontology*; MIT Press: Cambridge, MA, USA, 2015.
29. Dehainsala, H.; Pierra, G.; Bellatreche, L. OntoDB: An Ontology-Based Database for Data Intensive Applications. In *Proceedings of Database Systems for Advanced Applications*; Springer: Berlin/Heidelberg, Germany, 2007.
30. Araujo, T.H.; Agena, B.T.; Braghetto, K.R.; Wassermann, R. Ontomongo—Ontology-based data access for NoSQL. *Proc. Ceur Workshop Proc.* **2017**, *1908*, 55–66.
31. Calvanese, D.; Cogrel, B.; Komla-Ebri, S.; Kontchakov, R.; Lanti, D.; Rezk, M.; Rodriguez-Muro, M.; Xiao, G. Ontop: Answering SPARQL queries over relational databases. *Semant. Web* **2017**, *8*, 471–487. [CrossRef]
32. Fielding, R.T. *Architectural Styles and the Design of Network-Based Software Architectures*; University of California: Irvine, CA, USA, 2000.
33. Harris, S.; Seaborne, A. SPARQL 1.1 Query Language. W3C Recommendation 21 March 2013. 2013. Available online: http://www.w3.org/TR/sparql11-query/ (accessed on 14 September 2022).
34. Asprino, L.; Daga, E.; Gangemi, A.; Mulholland, P. Knowledge Graph Construction with a Façade: A Unified Method to Access Heterogeneous Data Sources on the Web. *ACM Trans. Internet Technol.* **2022**, *23*, 1–31. [CrossRef]
35. Klyne, G.; Carroll, J. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation 2004. Latest Version. Available online: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (accessed on 26 December 2023).
36. Steindl, G.; Frühwirth, T.; Kastner, W. Ontology-Based OPC UA Data Access via Custom Property Functions. In Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Zaragoza, Spain, 10–13 September 2019; Volume 2019, pp. 95–101. [CrossRef]
37. *OPC 10000-1 UA V 1.05.02*; Unified Architecture Core—UA. Standard, OPC Foundation. The Industrial Interoperability Standard: Scottsdale, AZ, USA, 2022.
38. Mahnke, W.; Leitner, S.H. OPC Unified Architecture—The future standard for communication and information modeling in automation. *ABB Rev.* **2009**, *3*, 3.
39. Cyganiak, R.; Das, S.; Sundara, S. R2RML: RDB to RDF Mapping Language. W3C Recommendation, W3C. 2012. Available online: https://www.w3.org/TR/2012/REC-r2rml-20120927/ (accessed on 26 December 2023).
40. Leshcheva, I.; Begler, A. A method of semi-automated ontology population from multiple semi-structured data sources. *J. Inf. Sci.* **2022**, *48*, 223–236. [CrossRef]
41. Boy, J.; Crepel, J.M.; Rosché, P. Test Suite for the CAE Implementor Forum Round 5S. 2019. Available online: https://www.mbx-if.org/documents_cae/test_suite%20CAE-IF%20R5S_v1.0.pdf (accessed on 9 July 2020).
42. Dean, M.; Schreiber, G. OWL Web Ontology Language Reference. W3C Recommendation 2004. Latest Version. Available online: http://www.w3.org/TR/2004/REC-owl-ref-20040210/ (accessed on 26 December 2023).
43. Pauwels, P.; Terkaj, W. EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Autom. Constr.* **2016**, *63*, 100–133. [CrossRef]

44. *OMG®Unified Modeling Language®Version 2.5.1 Specifications*; OMG UML®. Standards, Object Management Group (OMG®) 2017. Available online: https://www.omg.org/spec/UML/2.5.1/PDF (accessed on 26 December 2023).
45. Brickley, D.; Guha, R. RDF Schema 1.1. W3C Recommendation 2014. Latest Version. Available online: http://www.w3.org/TR/2014/REC-rdfschema-20140225/ (accessed on 26 December 2023).
46. Tony Liu, D.; William Xu, X. A review of web-based product data management systems. *Comput. Ind.* **2001**, *44*, 251–262. [CrossRef]
47. Abdelrahman, M.M.; Zhan, S.; Chong, A. A three-tier architecture visual-programming platform for building-lifecycle data management. In Proceedings of the SimAUD, Online, 25–27 May 2020. [CrossRef]
48. Lamy, J.B. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artif. Intell. Med.* **2017**, *80*, 11–28. [CrossRef] [PubMed]
49. Van Rossum, G.; Drake, F.L., Jr. *Python Tutorial*; Centrum voor Wiskunde en Informatica Amsterdam: Amsterdam, The Netherlands, 1995.
50. Foundation, P.S. Tkinter—Python Interface to Tcl/Tk. 2023. Available online: https://docs.python.org/3/library/tkinter.html (accessed on 5 December 2023).
51. Collette, A. *Python and HDF5*; O'Reilly: Sebastopol, CA, USA, 2013. [CrossRef]
52. Harris, C.R.; Millman, K.J.; Van Der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; et al. Array programming with NumPy. *Nature* **2020**, *585*, 357–362. [CrossRef] [PubMed]
53. Ganapathy, G.; Sagayaraj, S. To generate the ontology from java source code. *Int. J. Adv. Comput. Sci. Appl.* **2011**, *2*, 111–116. [CrossRef]
54. Bedini, I.; Nguyen, B. Automatic ontology generation: State of the art. In *PRiSM Laboratory Technical Report*; University of Versailles: Paris, France, 2007; pp. 1–15.
55. Chu, S.C.; Xue, X.; Pan, J.S.; Wu, X. Optimizing ontology alignment in vector space. *J. Internet Technol.* **2020**, *21*, 15–22.
56. He, Y.; Chen, J.; Antonyrajah, D.; Horrocks, I. BERTMap: A BERT-based ontology alignment system. In Proceedings of the AAAI Conference on Artificial Intelligence, Washington DC, USA, 7–14 February 2022; Volume 36, pp. 5684–5691.
57. Roy, S.; Modak, A.; Barik, D.; Goon, S. An overview of semantic search engines. *Int. J. Res. Rev.* **2019**, *6*, 73–85.