RESEARCH ARTICLE

# Agile Research Data Management with Open Source: CaosDB

Daniel Hornung [1]
Florian Spreckelsen [1]
Thomas Weiß [1]

1. IndiScale GmbH, Göttingen.

**Date Received:**
2021-08-01



**Keywords:**
Data Management, Research Data Management, Agile Data Management, Software Tools, FAIR Data, Good Scientific Practice

**Data availability:**

**Software availability:**
Software can be found here:
https://gitlab.com/caosdb

**Abstract.**
Research data management (RDM) in academic scientific environments increasingly enters the focus as an important part of good scientific practice and as a topic with big potentials for saving time and money. Nevertheless, there is a shortage of appropriate tools, which fulfill the specific requirements in scientific research. We identified where the requirements in science deviate from other fields and proposed a list of features which RDM software should fulfill to become a viable option.
Finally we analyzed the open-source RDMS CaosDB for compatibility with the proposed features and found that it fulfills the requirements.

## 1 Introduction

Research units, from small research groups at universities to large research and development departments are increasingly confronted with the challenge to manage large amounts of data, data of high complexity[1], [2] and changing data structures[3], [4]. The necessary tasks for research data management include storage, findability and long-term accessibility for new generations of researchers and for new research questions[4]–[6].

In spite of the advantages of implementing data management solutions[7], there is a lack of standard methods or even standard software so far for research data management, especially in the context of quickly evolving methods and research targets. In this article, we define the specific challenges for research data management (RDM) and propose features which suitable RDM software should have to (a) fulfill the practical needs and (b) be accepted by the potential users. We also demonstrate how the CaosDB[1][8] toolkit is a viable approach to satisfy all the proposed requirements.

1. Website: https://caosdb.org, source code: https://gitlab.com/caosdb/

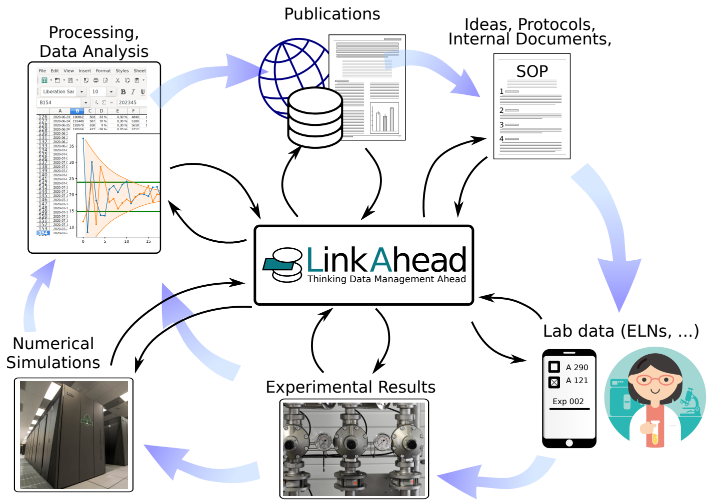


**Figure 1:** Schematic illustration of the scientific data lifecycle. Data can be obtained from every step, and in most cases the relationship between data entities is just as relevant as the raw data. While this example focuses on experimental and laboratory centered disciplines, comparative lifecycles also exist for theoretical sciences and most fields in the humanities.

## 2 Challenges for research data management

### 2.1 The scientific data lifecycle

Data which accrues in scientific research is more than just experimental readings, field notes or interview recordings. In order to fully represent the research journey and eventually enable reproducible science, the data from every research step may become relevant and thus should be consistently managed.

Figure 1 shows a schematic of different research steps during the research lifecycle, during which important data is generated. For full reproducibility, it is not sufficient however to simply store the data acquired at each point, but also to represent the semantic connections and make them searchable.

In more detail, the most relevant data source in scientific research are:

**Prior publications** An important part of good scientific practice (GSP) is to credit the influence of prior work, by the scientists themselves or third parties. Linking one's work to previous publications and making these connections public also helps to assess reproducibility and may lead to fruitful data-reuse in unforeseen contexts.

**Ideas and SOPs** The data here consists mostly of text documents which describe thoughts, hypotheses and planned standard operating procedures (SOPs). These documents fill the gap between previous work and the next round of data acquisitions, they often also work as a blueprint for the data acquisition phase.

**Lab data** Environmental data, device settings, used SOPs and ingredients and other incidental data typically accrues during the course of experiments and was traditionally stored in paper laboratory notebooks. Currently, a lot of laboratories switch to electronic lab notebooks (ELNs) for the same purpose. While this data is often seen as second-class "metadata", we hold that since often conclusions can be drawn from it, it deserves the same handling as final instrument readings.

**Experimental results** These are what is often considered the *main* data. For analysis, it mostly still needs parameters from at least the previous step, to filter for special conditions, to compare settings or to verify that values are compatible with standard literature.

**Numerical simulations** Similarly to experimental results, data obtained from numerical methods can not be interpreted without knowledge about used software and parameters, possibly hardware conditions and input from laboratories or third-party data sources. Since bit-exact reproducibility is possible in theory, all relevant settings should be stored unchanged.

**Data analysis** When analyzing data from previous steps, storing not only the used programs, scripts and their parameters, but also the semantic connections enables later researchers to reconstruct which method was used, which assumptions were made and under which conditions the input data was gathered.

**Next publication** Formally the end of the lifecycle, but of course also the beginning of many new ones, a publication contains a number of statements which are supported by data from previous steps. A comprehensive research RDM system (RDMS) could quickly summarize, for each plot in a publication, how the data was analyzed and acquired and under which assumptions the experimental setting was planned.

This list focuses on experimental and laboratory centered disciplines, but of course in the humanities and theoretical sciences, there are equivalent steps which are equally important to preserve and link to each other.

### 2.2 Specifics of scientific research data management

Existing data management systems (DMS) often still follow the paradigms of relational databases[9], [10]: There is a number of types for the data, with each type having a number of possible properties. Each stored entity belongs to one type and has the properties defined by the type, occasionally it is also possible to leave a property empty or *undefined*. Searching or filtering in the stored data entities is possible by criteria which operate on the content of the properties and on the data types. These DMS have been widely successful in many areas such as finance, administration and high-tech industries[11], [12], but remain scarce in both academic and private sector research[12], [13].

We hold that the main reasons are:

**Interoperability** Scientists tend to work with their own custom-written software, which often requires files with data to be directly accessible to the OS via a – remote or local – file system. Many DMS store data files internally and make them available only via download. Also programmatic access (query, retrieve, update) to data DMS solutions via network APIs was uncommon until a few years ago, so that acquired data could not be instantly integrated in the DMS.

**Agility** Traditional DMS require users to define a data model and stick to it. All data to be entered has to conform to the data model as it was defined. Research however is defined by having undefined outcomes, the research questions, experimental setup or analysis methods change more often than not over the course of one investigation. A DMS based on rigid SQL databases thus may soon become a liability instead of an asset.

**Learning curve** Research is impossible without the contribution of many participants, with different qualifications. With many DMS, it takes a lot of active learning before users can benefit from the provided data.

**Practical use** Systems which only store data, but do not provide short-term advantages, have high entrance barriers everywhere. In academic research however, junior scientists with short-term contracts have little incentive to invest time and money in systems which only may pay out on longer timescales.

## 3 Proposed features

Under the assumptions from the previous sections, we propose the following core properties which an RDMS should have:

**Semantic linkage** To fully map the real-world environment of data entities to the RDMS, it must be possible to link data sets with each other in a meaningful way. The default linking possibilities and properties of the data types in the RDMS form the *data model*.

**Deep search** The RDMS should have easily accessible search options not only for property values of stored entities, but also for links to other entities and properties (and link) thereof.

**Flexible data model** Researchers require an RDMS where the data model can be changed on the fly, without the need to migrate or discard existing data. When the data model is changed, for example due to new machines, protocols or evolving research questions, the existing data must remain valid and usable.

**Versioning** Mistakes during data acquisition happen, and it must be possible to correct existing data sets. At the same time, this editing must be made transparent and the history of each data set must be kept for future inspection.

**File system integration** For interaction with third-party programs, raw data files must be available on standard file systems. Ideally the scientists' workflows should remain unchanged by the RDMS.

**Open APIs** For machine interaction with third-party programs, the RDM must have a well-documented API. In research contexts, these programs are often custom-written by scientists without explicit computer science background, so extensive documentation is very desirable.

### 3.1 User acceptance

Additionally, to increase the acceptance by potential users, we focus on two additional areas, automated data integration and low-threshold search options.

### 3.1.1 Automation

Automation of repetitive data integration reduces error rates and frees users to concentrate on more challenging tasks. It is therefore desirable for an RDMS to have:

**Synchronization** The RDMS should make it easy for its administrators to integrate existing data sources (for example databases or file systems with structured folder hierarchies) into the RDMS: The RDMS should be synchronized automatically with data from these sources, which makes these data available in a unified manner via the RDMS interface. Note that the RDMS can not solve the conceptual problem of a single source of truth when synchronizing data from different sources, but it can at least highlight potential conflicts and where they first occurred to administrators.

**ELN integration** Research work in the lab is increasingly documented with electronic lab notebooks (ELNs)[14], [15], which allow to conveniently enter device and experimental settings in a semi-structured way. This data is usually critical in the analysis of acquired raw data from instruments, e.g. for searching specific data sets or filtering by parameters. There should be a possibility that the RDMS integrates the ELN data and presents it like data from other sources.

**Workflow representation** While following one SOP, the laboratory workflow is often highly standardized, which makes it suitable for representation within the RDMS. The RDMS should support workflows with different states, which can only be switched in an admin-defined pattern. This simplifies the work for users, because they may e.g. only see the interfaces which are relevant for the current sample processing step.

### 3.1.2 Simplified search

To overcome initial reluctance by users, it is important to flatten the learning curve[16]. Besides obvious requirements such as user-friendly documentation, we hold that it is especially important to provide simplified search options. The simplified search in an RDMS should give some early sense of achievement, so users can understand that an RDMS will lead to a simplification of their work.

## 4 CaosDB

We hold that CaosDB[8], a flexible data management framework, fulfills the above-mentioned requirements. CaosDB was initially developed by one of our colleagues, Timm Fitschen, during his time at the Max Planck Institute for Dynamics and Self-Organization. In 2018, CaosDB was released under the AGPLv3 license on gitlab.com[2]. Since 2020, CaosDB has found increased adoption in multiple research facilities.

### 4.1 Data Model

CaosDB's *meta* data model is shown schematically in Figure 2. The base type for everything is ENTITY, with the inheriting types PROPERTY (attributes of ENTITIES, may be list values and

2. https://gitlab.com/caosdb

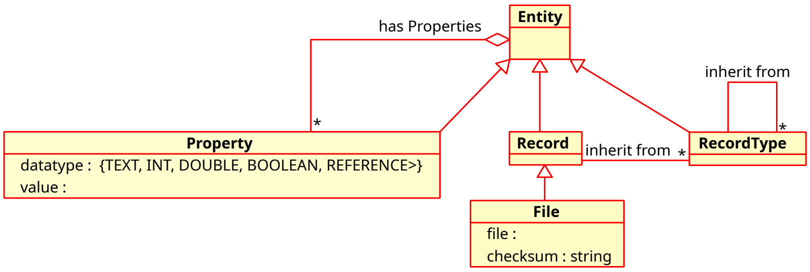


**Figure 2:** The meta data model of CaosDB.

references to other Entities), RecordType (templates for actual data sets) and Record. Actual data is typically stored in Records, which *inherit* from one or more RecordTypes and thus have all the Properties defined therein. The RecordTypes may form a complex inheritance hierarchy themselves. File entities are similar to Records, but additionally are connected to files which may reside on conventional file systems or potentially in abstracted cloud storages. This approach to use files at their current locations instead of duplicating file content not only increases CaosDB's scalability, but also lower the entrance barrier, since scientists can access the managed file in their traditional ways.

Details of this meta data model in CaosDB are elaborated on in [8], but it should be clear now already that CaosDB provides the *Semantic linkage* feature.

In CaosDB, the *data model* of the stored data refers to the RecordTypes and their Properties, which together describe the pattern to which newly created data sets should conform. The data model in CaosDB can be modified at any time, but the changes only take effect for data to be inserted *after* this modification. Existing data is not affected and remains unchanged. This property fulfills the proposed *Flexible data model* feature.

This possibility to completely change the data model, while not giving up on a general structure, places CaosDB between traditional SQL based relational databases and NoSQL[3] approaches (c.f. Figure 3). While we described above why rigid SQL databases are not suited for use in dynamic research environments, giving no structure (the NoSQL paradigm) tends to lead to incoherent data which is hard to search[4]. A third approach, using graph databases to represent semantic information, has not found its way into general adoption to our knowledge, presumably because the query languages tend to become very unwieldy, compare the appendix 7 for an example.

### 4.2 Architecture and Libraries

CaosDB uses a client/server based architecture, as depicted in Figure 4a. CaosDB has is a REST API for simple access by traditional clients and a web interface for browsers, as well as a gRPC API which allows for more complex operations, such as atomic content manipulations. The existing client libraries[5] and the open APIs provide the proposed *Interoperability* requirement.

One particularly useful client library component is the *CaosDB Crawler* framework. This extensible framework simplifies the work to synchronize external data sources with CaosDB

3. Popular examples for NoSQL database systems are CouchDB or MongoDB.
4. Missing structure in Data Lakes has lead to the tongue-in-cheek colloquialism "Data Swamp".
5. A list of the available libraries with the respective source code repositories are given in the Appendix section 6.

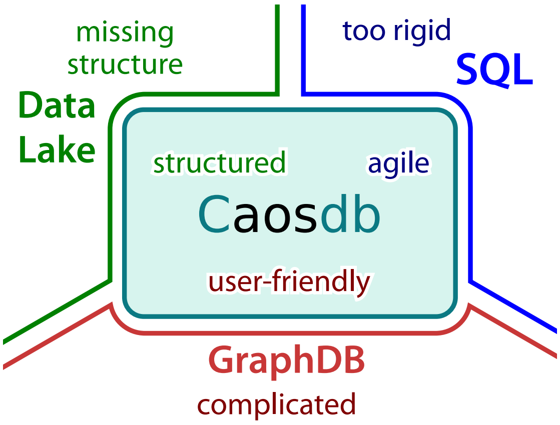


**Figure 3:** CaosDB compared to other database approaches.

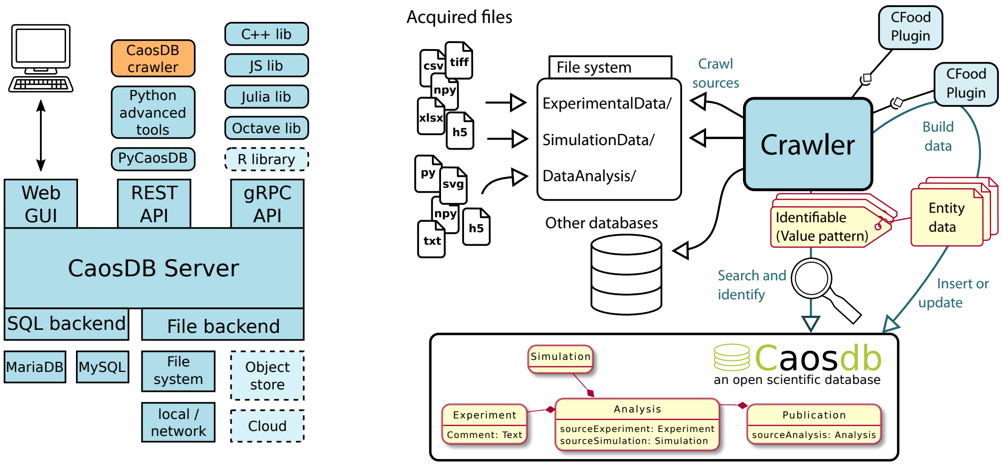


**Figure 4:** **(a)** CaosDB's server-client architecture with client libraries and backend components. Dotted elements are under development. **(b)** The crawler framework facilitates fast development of custom data integration from a diversity of sources.

through a plugin system. The crawler workflow can be characterized as follows:

1. The crawler periodically checks its data sources for new or changed data stores, such as file systems or the content of other databases.

2. Each new data source is fed to a so-called *CFood plugin* for consumption. There is a choice of existing plugins, or administrators can write their own. The CFood plugin's job is to build CaosDB entities from the consumed data and to specify *Identifiables*, which work as search patterns.

3. The crawler checks for each *Identifiable* if a corresponding entity exists already in CaosDB. If there is no corresponding entity, the entity as returned by the CFood plugin is inserted into CaosDB. If there is already an existing entity, the Crawler will attempt to merge the existing with the new entity and notify the data curators in case of merge conflicts.

This tool set provides the *Synchronization* requirement, and if ELNs are used as external data source, the *ELN integration*. Practical use of CaosDB crawler framework has previously been demonstrated in [17] and ELN integration was implemented in [18].

### 4.3 Additional features

**Deep search** CaosDB offers a simple semantic query language, which borrows some semantics from SQL, but has a focus on usability for non-technical users. The CaosDB query language makes deep search easy with expressions like the following:

```
FIND Analysis WITH quality_factor > 0.5
            AND WITH Sample WITH weight < 80g
```

This convenient nesting of query expressions circumvents the `JOIN` operations from traditional SQL languages. A full documentation of CaosDB's query language is available online[6] or in CaosDB's sources.

**Search templates** CaosDB's web interface provides customizable search templates which allow more advanced users to create their own templated queries, which can then be shared with novice users for *simplified searches*. In templated queries, users can insert custom strings into pre-defined locations of a search query, see Figure 5.

**Versioning** When entities are modified in CaosDB, time and user of the change are recorded and CaosDB puts the previous version onto a history stack and amends the current version with link to the previous version. Over time, each entity may thus grow to a tree of linked versions, which can be retrieved via the web UI or programmatically through the APIs. This feature of CaosDB enables scientific research data management users to adhere to the principles of good scientific practice.

**State management** In CaosDB, users may declare a state machine of states and allowed transitions. Users may then affix states to entities, and these states can then only be changed according to the rules of the state machine. In this way, users can implement a *workflow representation* which ensure that for example laboratory samples run through a specified list of preparation steps in order.

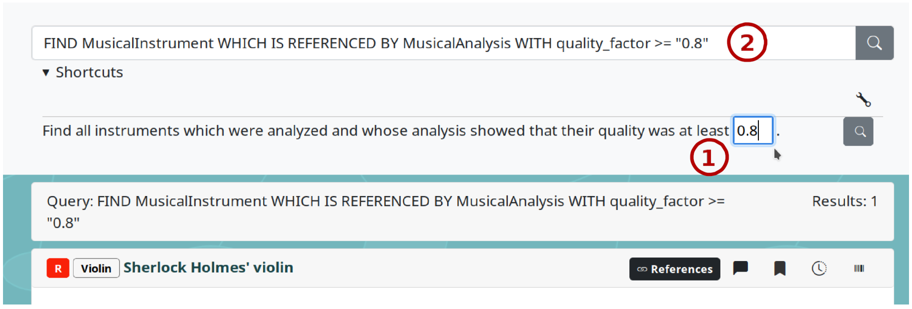


**Figure 5:** A query template in CaosDB's web UI. The user can enter a custom value into an input field ① and the template is then executed as a plain CaosDB query ②. Screenshot from https://demo.indiscale.com.

6. https://docs.indiscale.com/caosdb-server/CaosDB-Query-Language.html

### 4.4 Availability and documentation

CaosDB is available on the public Git repository gitlab.com at `https://gitlab.com/caosdb`, a detailed list of CaosDB's sub projects is given in the annex.

For the interested public, there is a live demo server at `https://demo.indiscale.com`, hosted by IndiScale GmbH. This demo server is actually running LinkAhead, a commercially supported distribution of CaosDB.

There are also Debian/Ubuntu packages to run precompiled LinkAhead/CaosDB for download at `https://indiscale.com/download`.

CaosDB's sub projects each have their own documentation in their source directories. The documentation is also available online at `https://docs.indiscale.com`.

## 5 Conclusion

We found that scientific research has specific requirements to data management: Interoperability, agility, adequate learning curves and early practical use. Most data management system do not satisfy these requirements, so we set up a specific list of features which a scientific RDMS should have. The open source research data management framework CaosDB has the specified features and thus fulfills the requirements for a scientific RDMS.

Special attention has to be given to the aspect of synchronizing data from external sources (e.g., crawled files, ELNs) and records in the RDMS. Different sources can (usually by some error) have conflicting data, or entries in the RDMS can be changed manually by users after their insertion. In our experience, this problem can not be solved in a general and purely technical way. Instead, best practices have to be implemented as to where possible errors should be corrected and whether some sources have precedence above each other. A RDMS like CaosDB, together with the crawler framework, can help administrators identify inconsistencies in the case of two or more data sources. Through versioning, it is visible who and when maybe changed data manually. How to optimize the help in recognizing potential conflicts, and in the end curate data both in the RDMS and in the external sources, is subject of the authors' ongoing research.

We hope that the open source license of CaosDB will inspire scientists to contribute to CaosDB.

## 6 Appendix: List of CaosDB libraries

The following libraries for programming client applications are publicly available:

**Python** `https://gitlab.com/caosdb/caosdb-pylib` The Python client library can be used for third-party applications and is the foundation for several other libraries:

- **Advanced Python tools** `https://gitlab.com/caosdb/caosdb-advanced-user-tools` Additional high-level tools, including a legacy implementation of the CaosDB crawler.

- **Crawler** `https://gitlab.com/caosdb/caosdb-crawler` A new implementation of the CaosDB crawler.

**JavaScript** https://gitlab.com/caosdb/caosdb-webui The JavaScript library is part of the web user interface component.

**Protobuf API** https://gitlab.com/caosdb/caosdb-proto The gRPC API is defined via these protobuf files.

**C++** https://gitlab.com/caosdb/caosdb-cpplib The C++ library uses the gRPC API of CaosDB.

**Octave** https://gitlab.com/caosdb/caosdb-octavelib The Octave/Matlab library is based upon the C++ library.

**Julia** https://gitlab.com/caosdb/caosdb-julialib The Julia library also is based upon the C++ library.

## 7 Appendix: Query language comparison

As an example for nested queries in different query languages, we consider the search for female UK-based writers in a certain time period, whose given or family name starts with the letter `M`. We used the RDF query language SPARQL with Wikidata[7] identifiers and CaosDB's query language with fictional but plausible identifier names.

The SPARQL query is as follows:

```
SELECT DISTINCT ?item ?itemLabel ?givenName ?familyName WHERE {
    ?item wdt:P31 wd:Q5;  # Any instance of a human.
          wdt:P27 wd:Q145; # United Kingdom
          wdt:P21 wd:Q6581072; # female
          wdt:P106 wd:Q36180; # writer
          wdt:P569 ?birthday;
          wdt:P570 ?diedon;
          wdt:P734 [rdfs:label ?familyName];
          wdt:P735 [rdfs:label ?givenName].
  FILTER(?birthday > "1870-01-01"^^xsd:dateTime
      && ?diedon < "1950-01-01"^^xsd:dateTime)
  FILTER(regex(?givenName, "M.*") || regex(?familyName, "M.*"))
    SERVICE wikibase:label { bd:serviceParam wikibase:language "en" }
  }
```

In contrast, the CaosDB query looks like this:

```
SELECT given_name, family_name FROM Writer
  WITH gender=f AND country=UK AND birthday > 1870 AND death < 1950
       AND (given_name LIKE "M*" OR family_name LIKE "M*")
```

7. https://www.wikidata.org

## 8 Acknowledgements

We acknowledge the previous work on the CaosDB software by its the main authors and independent contributors, especially Timm Fitschen.

## 9 Conflicts of interest

The authors work for IndiScale GmbH, which provides commercial support and other services for CaosDB and the derived free and open-source LinkAhead distribution. DH and FS contributed to the development of CaosDB.

## 10 Roles and contributions

**Daniel Hornung:** Conceptualization, Visualization, Writing – original draft

**Florian Spreckelsen:** Conceptualization, Writing – review & editing

**Thomas Weiß:** Conceptualization, Visualization

## References

[1] C. R. Bauer, N. Umbach, B. Baum, K. Buckow, T. Franke, Gr\&\#252, R. Tz, L. Gusky, S. Y. Nussbeck, M. Quade, S. Rey, T. Rottmann, O. Rienhoff, and U. Sax, "Architecture of a biomedical informatics research data management pipeline," *Exploring Complexity in Health: An Interdisciplinary Systems Approach*, pp. 262–266, 2016, Publisher: IOS Press. doi: 10.3233/978-1-61499-678-1-262. [Online]. Available: https://ebooks.iospress.nl/doi/10.3233/978-1-61499-678-1-262 (visited on 01/18/2023).

[2] C. L. Borgman, "The conundrum of sharing research data," *Journal of the American Society for Information Science and Technology*, vol. 63, no. 6, pp. 1059–1078, 2012, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/asi.22634, issn: 1532-2890. doi: 10.1002/asi.22634. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.22634 (visited on 01/18/2023).

[3] A. M. Khattak, K. Latif, and S. Lee, "Change management in evolving web ontologies," *Knowledge-Based Systems*, vol. 37, pp. 1–18, Jan. 1, 2013, issn: 0950-7051. doi: 10.1016/j.knosys.2012.05.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705112001323 (visited on 01/18/2023).

[4] T. Schneider and M. Šimkus, "Ontologies and data management: A brief survey," *KI - Künstliche Intelligenz*, vol. 34, Aug. 13, 2020. doi: 10.1007/s13218-020-00686-3.

[5] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. G. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. C. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag,

T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, “The FAIR guiding principles for scientific data management and stewardship,” *Scientific Data*, vol. 3, no. 1, p. 160 018, Mar. 15, 2016, issn: 2052-4463. doi: 10.1038/sdata.2016.18. [Online]. Available: https://www.nature.com/articles/sdata201618 (visited on 11/22/2021).

[6] R. Higman, D. Bangert, and S. Jones, “Three camps, one destination: The intersections of research data management, FAIR and open,” *Insights*, vol. 32, no. 1, p. 18, May 22, 2019, Number: 1 Publisher: UKSG in association with Ubiquity Press, issn: 2048-7754. doi: 10.1629/uksg.468. [Online]. Available: http://insights.uksg.org/articles/10.1629/uksg.468/ (visited on 01/18/2023).

[7] G. W. Hruby, J. McKiernan, S. Bakken, and C. Weng, “A centralized research data repository enhances retrospective outcomes research capacity: A case report,” *Journal of the American Medical Informatics Association*, vol. 20, no. 3, pp. 563–567, May 1, 2013, issn: 1067-5027. doi: 10.1136/amiajnl-2012-001302. [Online]. Available: https://doi.org/10.1136/amiajnl-2012-001302 (visited on 01/19/2023).

[8] T. Fitschen, A. Schlemmer, D. Hornung, H. tom Wörden, U. Parlitz, and S. Luther, “CaosDB—research data management for complex, changing, and automated research workflows,” *Data*, vol. 4, no. 2, p. 83, Jun. 2019, Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, issn: 2306-5729. doi: 10.3390/data4020083. [Online]. Available: https://www.mdpi.com/2306-5729/4/2/83 (visited on 01/18/2023).

[9] A. Ailamaki, V. Kantere, and D. Dash, “Managing scientific data,” *Communications of the ACM*, vol. 53, no. 6, pp. 68–78, Jun. 1, 2010, issn: 0001-0782. doi: 10.1145/1743546.1743568. [Online]. Available: https://doi.org/10.1145/1743546.1743568 (visited on 01/18/2023).

[10] A. Nourani, H. Ayatollahi, and M. S. Dodaran, “Clinical trial data management software: A review of the technical features,” *Reviews on Recent Clinical Trials*, vol. 14, no. 3, pp. 160–172, Sep. 1, 2019. doi: 10.2174/1574887114666190207151500.

[11] T. P. Raptis, A. Passarella, and M. Conti, “Data management in industry 4.0: State of the art and open challenges,” *IEEE Access*, vol. 7, pp. 97 052–97 093, Jul. 16, 2019, Conference Name: IEEE Access, issn: 2169-3536. doi: 10.1109/ACCESS.2019.2929296.

[12] S. V. Tuyl and A. Whitmire, “Investigation of non-academic data management practices to inform academic research data management,” *Research Ideas and Outcomes*, vol. 4, e30829, Oct. 31, 2018, Publisher: Pensoft Publishers, issn: 2367-7163. doi: 10.3897/rio.4.e30829. [Online]. Available: https://riojournal.com/article/30829/ (visited on 01/19/2023).

[13] M. A. Kennan and L. Markauskaite, “Research data management practices: A snapshot in time,” *International Journal of Digital Curation*, vol. 10, no. 2, pp. 69–95, Jun. 30, 2015, Number: 2, issn: 1746-8256. doi: 10.2218/ijdc.v10i2.329. [Online]. Available: http://ijdc.net/index.php/ijdc/article/view/10.2.69 (visited on 01/19/2023).

[14] S. Kanza, C. Willoughby, N. Gibbins, R. Whitby, J. G. Frey, J. Erjavec, K. Zupančič, M. Hren, and K. Kovač, "Electronic lab notebooks: Can they replace paper?" *Journal of Cheminformatics*, vol. 9, no. 1, p. 31, May 24, 2017, issn: 1758-2946. doi: 10.1186/s13321-017-0221-3. [Online]. Available: https://doi.org/10.1186/s13321-017-0221-3 (visited on 01/19/2023).

[15] S. G. Higgins, A. A. Nogiwa-Valdez, and M. M. Stevens, "Considerations for implementing electronic laboratory notebooks in an academic research environment," *Nature Protocols*, vol. 17, no. 2, pp. 179–189, Feb. 2022, Number: 2 Publisher: Nature Publishing Group, issn: 1750-2799. doi: 10.1038/s41596-021-00645-8. [Online]. Available: https://www.nature.com/articles/s41596-021-00645-8 (visited on 01/20/2023).

[16] F. Abdullah, R. Ward, and E. Ahmed, "Investigating the influence of the most commonly used external variables of TAM on students' perceived ease of use (PEOU) and perceived usefulness (PU) of e-portfolios," *Computers in Human Behavior*, vol. 63, pp. 75–90, C Oct. 1, 2016, issn: 0747-5632. doi: 10.1016/j.chb.2016.05.014. [Online]. Available: https://doi.org/10.1016/j.chb.2016.05.014 (visited on 01/20/2023).

[17] F. Spreckelsen, B. Rüchardt, J. Lebert, S. Luther, U. Parlitz, and A. Schlemmer, "Guidelines for a standardized filesystem layout for scientific data," *Data*, vol. 5, no. 2, p. 43, Jun. 2020, Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, issn: 2306-5729. doi: 10.3390/data5020043. [Online]. Available: https://www.mdpi.com/2306-5729/5/2/43 (visited on 01/18/2023).

[18] H. tom Wörden, *CaosDB crawler for eLabFTW ELN data*. [Online]. Available: https://gitlab.com/caosdb/crawler-cfoods/elabftw-cfood.