

Data Model Creation with MetaConfigurator


Felix Neubauer ¹, Jürgen Pleiss ², and Benjamin Uekermann ¹


Abstract: In both research and industry, significant effort is devoted to the creation of standardized data models that ensure data adheres to a specific structure, enabling the development and use of common tools. These models (also called schemas), enable data validation and facilitate collaboration by making data interoperable across various systems. Tools can assist in the creation and maintenance of data models. One such tool is MetaConfigurator, a schema editor and form generator for JSON schema and for JSON/YAML documents. It offers a unified interface that combines a traditional text editor with a graphical user interface (GUI), supporting advanced schema features such as conditions and constraints. Still, schema editing can be complicated for novices, since MetaConfigurator shows all options of JSON schema, which is very expressive. The following improvements and functionalities have been designed and implemented to further assist the user: 1) A more user-friendly schema editor, distinguishing between an easy and an advanced mode based on a novel meta schema builder approach; 2) A CSV import feature for seamless data transition from Excel to JSON with schema inference; 3) Snapshot sharing for effortless collaboration; 4) Ontology integration for auto-completion of URIs; and 5) A novel graphical diagram-like schema view for visual schema manipulation. These new functionalities are then applied to a real-world use case in chemistry, demonstrating the improved usability and accessibility of MetaConfigurator.


Keywords: RDM, Research Data Management, JSON, YAML, Configuration, Schema, Editor, Tool, GUI, Ontology

1 Introduction³

In both research and industry, data has become increasingly important for driving innovation and generating new insights [OE15, Sc10]. To facilitate interoperability across systems, structured data formats, such as JSON [Cr06], YAML [BKEdN09], XML [Ye06], and CSV are used. Data in these formats differ from natural language by following a strict syntax [Ch56, Ba59], making it parsable and readable by a machine in a deterministic manner. Data formats define the syntax of data, but neither the semantics (meaning) [Kn68] of it, nor which properties are required or which constraints they need to satisfy. On top of a data format, a data model (schema) can be introduced, which specifies required properties and constraints and can imply semantics (for example through descriptions). A data model (also called schema) allows communicating the data structure with others, validating data and

¹ University of Stuttgart, Institute for Parallel and Distributed Systems, Universitätsstraße 38, 70569 Stuttgart, Germany, felix@neuby.de,  <https://orcid.org/0009-0008-5367-2034>;

Benjamin.Uekermann@ipvs.uni-stuttgart.de,  <https://orcid.org/0000-0002-1314-9969>

² University of Stuttgart, Institute of Biochemistry and Technical Biochemistry, Allmandring 31, 70569 Stuttgart, Germany, juergen.pleiss@itb.uni-stuttgart.de,  <https://orcid.org/0000-0003-1045-8202>

³ This paper is a shortened and adapted version of the master thesis of Felix Neubauer [Ne24a]. Some paragraphs (mainly in section 2) are one-to-one copies from the master thesis.

developing common tools and techniques, which manipulate or analyze the data. However, despite their clear advantages, the creation and adoption of standardized data models are often neglected. Many industries and research fields still rely on ad-hoc data structures, typically found in tools, such as Excel, which, while flexible, lack the rigor required for collaboration across different systems and domains [He11]. The lack of standardization can lead to inconsistencies, inefficiencies, and data loss when information needs to be exchanged or integrated across systems [WS96].

Although organizations increasingly recognize the importance of data models, they often develop custom solutions that are incompatible with those of other organizations. Such practices lead to fragmentation and siloed data ecosystems, hindering the reuse and sharing of valuable information [Pa19, Ko19]. Addressing this issue presents two key challenges: first, designing data models that serve as a common ground for diverse stakeholders requires substantial expertise; second, manually editing or updating these models is both time-intensive and error-prone, especially for users unfamiliar with schema languages. These challenges underscore the urgent need for tools that simplify the creation and management of data models, enabling broad adoption and reducing technical overhead.

To address this challenge, *MetaConfigurator* has been developed by us [Ne24b] as an open-source tool⁴ specifically designed to help users create and edit data models for data in JSON or YAML. *MetaConfigurator* is not an extension of any existing proprietary tool but a standalone free web application⁵ developed to address the unique needs of schema and data creation and management. It provides a unified interface that combines a traditional text editor with a GUI editor for additional assistance. The tool supports advanced schema features, such as conditions, constraints, and composition, enabling the creation of comprehensive and robust data models. It uses the schema language *JSON Schema* [Pe16].

Discussions with researchers and practitioners from various fields highlighted various limitations and difficulties in the work with *MetaConfigurator*, which this paper addresses with targeted solutions. The process of schema editing in *MetaConfigurator* can be difficult for beginners, as the full expressiveness of *JSON Schema* is directly exposed to the user without abstraction. To lower the entry barrier and hide complexity from users, a distinction is introduced between an easy and an advanced mode. Also, a novel graphical schema view is introduced, which visualizes schemas in a way similar to a Unified Modeling Language (UML) class diagram and allows for graphical editing of the schema. To ensure a smooth transition from Excel to JSON, a CSV import functionality has been developed, allowing researchers to convert their data into JSON with a few clicks and automatically inferring the schema. Additionally, because there already exist different ontologies in many fields and researchers are interested in linking their data models with it, a novel ontology integration is added to *MetaConfigurator*, providing auto-completion for existing ontology URI. Finally, to foster collaboration and make it easy for users to share their data or schema, a snapshot sharing functionality is developed.

⁴ <https://github.com/MetaConfigurator/meta-configurator>, accessed 2025/01/11.

⁵ <https://www.metaconfigurator.org>, accessed 2025/01/13

The paper is structured as follows: Section 2 presents the background and related work. Section 3 details the design and implementation of the new functionalities. Section 4 demonstrates the application of the CSV import and diagram-like schema editor for a biochemistry use case. Finally, section 5 summarizes the contributions and discusses potential future work.

2 Background and Related Work

This section introduces work related to schema editors, the original version of MetaConfigurator, and schema visualization techniques.

2.1 JSON Schema

JSON schema has evolved to being the de-facto standard schema language for JSON documents [Ba21]. JSON schema and other schema languages for JSON can also be applied to YAML, as JSON and YAML documents have a similar structure (JSON is a subset of YAML). Some syntactical details of YAML can, however, not be expressed with JSON schema. Theoretical research about JSON schema is concerned with schema inference [ČS20, Ba19], validation [At24], witness generation [At22] and schema matching [Wa20], among other techniques.

2.2 Schema Editors

There exist several so-called schema editors, which are tools for creating and editing schemas that are text-based or graphical (or both). JSON Editor Online⁶ is a web-based editor for JSON schemas and JSON documents. It divides the editor into two parts, where one part can be used to edit the schema and the other part can be used to edit a JSON document, which is validated against the schema. The editor provides various features, such as syntax highlighting and highlighting of validation errors. However, the features of the editor are limited. For example, it does not provide any assistance for the user, such as tooltips or auto-completion. For new documents, it does not show any properties of the schema, so the user has to know the schema beforehand.

There also exists a variety of schema editors that are paid software, such as Altova XMLSpy⁷, Liquid Studio⁸, XML ValidatorBuddy⁹, JSONBuddy¹⁰, XMLBlueprint¹¹, and Oxygen

⁶ <https://jsoneditoronline.org>, accessed 2024/09/24.

⁷ <https://www.altova.com/xmlspy-xml-editor>, accessed 2024/09/24.

⁸ <https://www.liquid-technologies.com/json-schema-editor>, accessed 2024/09/24.

⁹ <https://www.xml-buddy.com/>, accessed 2024/09/24.

¹⁰ <https://www.json-buddy.com/>, accessed 2024/09/24.

¹¹ <https://www.xmlblueprint.com>, accessed 2024/09/24.

XML Editor¹². Those are editors for XML or JSON schema, mostly with a combination of text-based and graphical views. These tools are not web-based and not open-source. Furthermore, they do not focus on editing a JSON document based on a schema, but rather only on editing the schema itself.

*Adamant*¹³ is a JSON Schema-based form generator and schema editor specifically designed for scientific data [SSB22]. It generates a GUI from a JSON schema, allows editing and creating JSON schema documents, and differentiates between a schema edit mode and a data edit mode. A noteworthy feature is that it supports the extraction of units from the description of a field, which is helpful for scientific data. Besides the frontend, Adamant also provides a backend that allows the integration into other tools as well as storing and reusing schemas. Adamant's UI-based schema editor is intuitive and user-friendly, even for users who are not familiar with JSON schema. However, Adamant does not provide a text-based editor as an alternative to the GUI and does not support various JSON schema keywords. For example, it is not possible to restrict strings to a certain regular expression and it does not support the keyword `oneOf`, which many schemas use [Ba21].

2.3 MetaConfigurator

MetaConfigurator [Ne24b] is a general-purpose form generator and schema editor that is not bound to a specific domain. It uses a modular and extensible architecture that supports different data formats (e.g., JSON and YAML) and different ways to present and edit the data (e.g., a text editor and a GUI editor). The schema editor and the UI for editing data are the same, as the schema itself is treated as a structured data file. MetaConfigurator is a client-side web application, which means that it runs in the browser of the user and does not require a server. It has three distinct views:

1. Data editor: In this view, the user can modify their structured data file, based on a schema.
2. Schema editor (Figure 1): In this view, the user can modify their schema.
3. Settings: In this view, the user can adjust the parameters of the tool.

The UI of each view is divided into two main panels: the text editor (on the left) and the GUI editor (on the right). In the text editor, the user can modify their data by hand, the same way as in a regular text editor. Features, such as syntax highlighting and schema validation, assist the user. In the GUI panel, the user can modify their data with the help of a GUI. The GUI is based on the JSON schema file that the user provides. This design combines the benefits of both a text editor with the benefits of a GUI. A text editor is efficient for many tasks and more suited for users with a technical understanding of the data structure, while a GUI enables users without deep technical understanding to work with the data.

¹² <https://www.oxygenxml.com>, accessed 2024/09/24.

¹³ Current version of Adamant as of writing this paper: Adamant v1.2.0

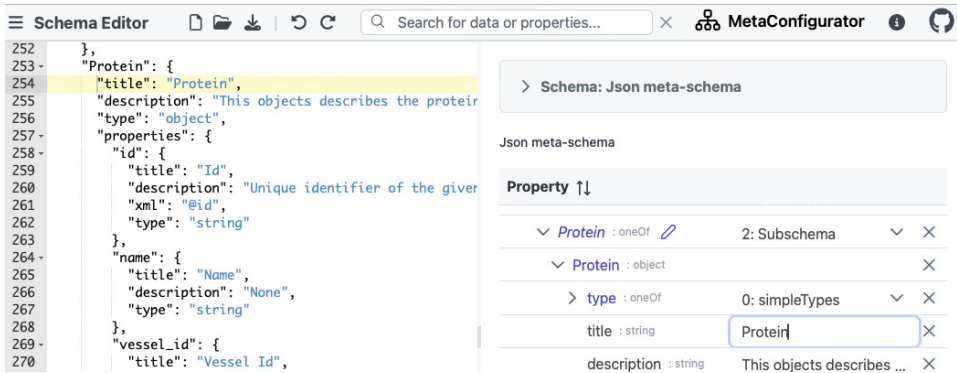


Fig. 1: The MetaConfigurator schema editor view. The schema editor is based on the same UI as the data editor.

Tab. 1: File data and schema for the different views.

Mode	Data	Schema
Data editor	User data	User schema
Schema editor	User schema	JSON Schema meta-schema
Settings	Settings data	Settings schema

Nevertheless, a GUI also simplifies the editing process for expert users. As a schema is a structured data file itself, it is treated as such, and the tool offers assistance accordingly. The user can edit the schema in the same way as they can edit their data. An adapted version of the JSON schema meta-schema is used to generate the UI for the schema editor. The schema editor itself is generated automatically and supports all JSON schema keywords, due to being generated from the meta-schema. Whenever the user edits a structured data file using the tool, they do so using some underlying schema. Even the settings of the tool are treated as a structured data file, for which there is an underlying settings schema. Table 1 shows which data and schema the tool uses in the different views/modes.

2.4 Schema Visualization

This section describes existing work related to visualizing data models and schemas.

2.4.1 Entity Relationship Diagrams

The entity relationship model is a conceptual data model and was proposed by Chen [Ch76, Ch77] in 1976, focusing on relational database design. The three core components of the

model are *entity* types, *relationship* types and *attribute* types. In the example diagram (Figure 2) the Person entity type is depicted with a rectangle, attribute types with a circle and relationship types with a diamond shape. The `firstName` and `lastName` attribute types make up the *keys* for the entity in the database, which is why they are underlined. Besides the diagrammatic notation described by Chen, others have been proposed, such as the *Bachmann* notation and notations that allow *n-ary* relationships [TYF86]. Song et al. [SEP95] compare different notations.

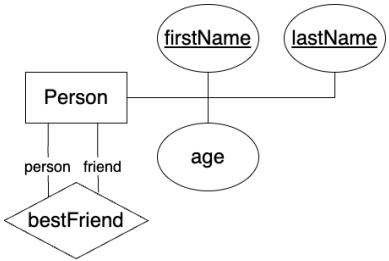


Fig. 2: Example entity relationship diagram in Chen notation.

2.4.2 UML Diagrams

UML [RJB04, Er03, PP05] is a modeling language to standardize the visualization of the design of a system. UML specifies different types of view (*use-case*, *logical*, *implementation*, *process* and *deployment* views) and different types of diagrams (*class diagrams*, *object diagrams*, *state machines*, *activity diagrams*, *interaction diagrams* and others). Figure 3¹⁴ shows a UML diagram that models the Person class from the entity relationship diagram in Figure 2. It does not distinguish between primary and secondary keys and expresses the constraint that age must not be negative, which the entity relationship diagram in Figure 2 does not model.

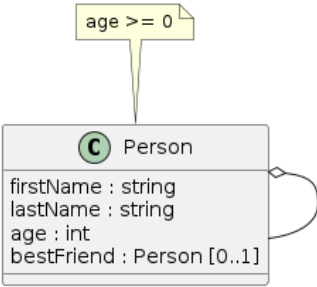


Fig. 3: Example UML class diagram.

¹⁴ created with <http://www.plantuml.com/plantuml/uml/>, accessed 2024/09/24.

3 Design and Implementation

This section describes the novel MetaConfigurator functionality introduced by this paper.

3.1 Simplified and Customizable Schema Editor

The complexity of the schema editor arises from the expressiveness of JSON schema and *MetaConfigurator's* support for most of its features, resulting in a wide range of options for users to navigate. To simplify the editing process while retaining expressiveness, we introduce a novel "meta schema builder", which dynamically constructs a custom JSON meta schema based on user-defined settings. MetaConfigurator dynamically creates the schema editor GUI from the provided meta schema. Giving it a simpler schema with less options results in a simpler schema editor, while a meta schema with all features of JSON schema results in a more advanced and complex schema editor. Additional features, such as a functional preview of the resulting data editor GUI further enhance usability, allowing users to visualize their schema in real-time.

3.2 CSV Import

We introduce a CSV import feature for MetaConfigurator to facilitate the transition from Excel or CSV data into JSON. Users can choose between importing a table as an independent JSON array or expanding an existing table by linking data through a primary-foreign key relationship, similar to SQL JOIN operations. Additionally, we provide a mapping interface to customize the transformation of CSV columns into JSON paths, supporting complex nesting and hierarchical structures. To further ease the process, schema inference functionality has been added, which generates an initial JSON schema from the uploaded data using the *jsonhero/schema-infer*¹⁵ library, reducing manual effort. However, the current implementation does not yet support appending multiple CSV imports into the same JSON path, a limitation that may be addressed in future updates.

3.3 Sharing Snapshots

Two key features are introduced in MetaConfigurator to enhance collaboration: query string support and backend storage for snapshots. Query strings allow users to share pre-loaded configurations (data, schema, settings) via customized URLs, including compatibility with GitHub URLs, ensuring a seamless sharing experience without backend reliance. A backend powered by Python Flask¹⁶ and MongoDB¹⁷ enables users to save, retrieve, and share session

¹⁵ <https://github.com/triggerdotdev/schema-infer>, accessed 2025/01/17.

¹⁶ <https://flask.palletsprojects.com/en/stable/>, accessed 2024/12/17.

¹⁷ <https://www.mongodb.com>, accessed 2024/12/17.

snapshots, supporting collaborative workflows through project creation and secure updates via project IDs and passwords. Rate limits and periodic cleanup ensure backend stability and scalability, while Docker Compose¹⁸ simplifies deployment across frontend, backend, and database services. Future plans include stabilizing backend hosting and integrating snapshot sharing into more prominent parts of the MetaConfigurator UI for broader usability.

3.4 JSON Schema Diagram

To further simplify editing JSON schemas and for better visualization, a novel graphical representation technique is developed and integrated into MetaConfigurator. This approach generates an interactive and editable schema diagram, overcoming limitations of existing visualization methods for JSON data and UML-based representations. A custom algorithm recursively maps JSON schema elements to nodes and edges, supporting features like properties, enumerations, compositions, and conditionals. The visualization, implemented using Vue Flow¹⁹ and Dagre²⁰, adapts to horizontal and vertical layouts to accommodate diverse schema structures and panel dimensions. Users can interact with the graph to navigate, zoom, and make schema edits such as renaming nodes or adding properties or classes, with real-time synchronization across MetaConfigurator panels. Advanced schema features, such as composition or conditionals, are visualized but not editable. Future possible enhancements include UML-like edge styling and support for keyboard shortcuts, making this solution a powerful tool for both schema exploration and modification.

4 Use Case in Chemistry

All sources, a demonstration video and a more detailed guide for this section are available on GitHub²¹.

Many researchers store research data in Excel files with researcher-specific structures, hindering automation and standardization. To address this, they aim to migrate their data to JSON, adopting common data standards for applying automated scripts, machine learning, and integrating data into a knowledge graph. Researchers, who lack expertise in JSON and JSON Schema, require user-friendly tools to create schemas without needing in-depth technical knowledge. MetaConfigurator was introduced to meet these needs, and its functionality was extended based on researcher feedback, prioritizing CSV import and graphical schema editing.

¹⁸ <https://docs.docker.com/compose/>, accessed 2024/12/17.

¹⁹ <https://vueflow.dev>, accessed 2024/12/17.

²⁰ <https://github.com/dagrejs/dagre>, accessed 2024/12/17.

²¹ https://github.com/MetaConfigurator/meta-configurator/tree/main/documentation_user/examples/mof_synthesis, accessed 2024/12/17.

A collaborative use case was developed where synthesis experiment data in Excel was migrated to JSON. The process began with exporting the Excel data as a CSV file and importing it into MetaConfigurator, which inferred the schema and generated a JSON document. Users could visually adjust the schema to include additional chemical data, such as as identifiers and properties retrieved from the PubChem [Ki16] database. Adjustments were made using MetaConfigurator's graphical schema editor, which simplifies the process of editing data models. Figure 4 shows the graphical schema editor.

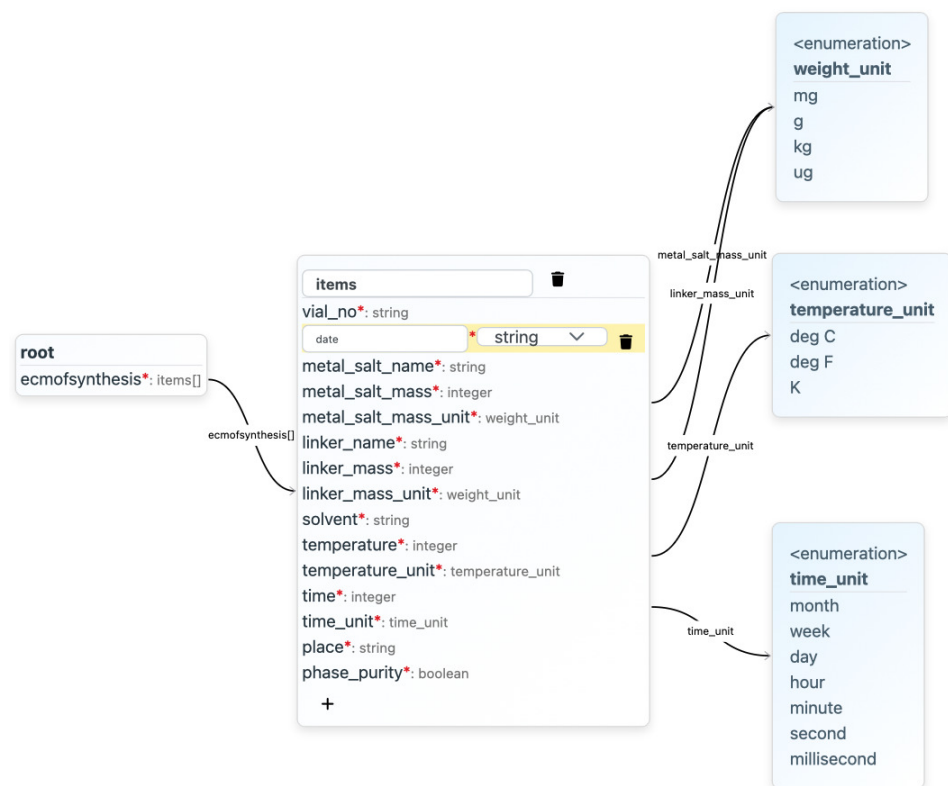


Fig. 4: The new graphical schema editor in MetaConfigurator.

To enrich the data, a Python script was developed, automatically appending metadata to the JSON document. The updated schema ensured that the enriched data adhered to the desired structure, including new object nodes for compounds such as metal salt and linker. This script demonstrated how external tools could seamlessly integrate with JSON data produced by MetaConfigurator.

The project highlighted that schema creation and data enrichment could be accomplished without prior JSON or JSON Schema expertise, thanks to MetaConfigurator's intuitive interface. The schema enables communication of data models, facilitates schema validation,

and supports sharing across research groups. Furthermore, migrating data to JSON unlocks advanced applications, such as automated data enrichment and integration into other systems. Complete instructions, example files, and the Python script are available on GitHub, ensuring reproducibility for future projects. This use case demonstrates the potential of MetaConfigurator to bridge the gap between research workflows and modern data practices.

5 Conclusion and Outlook

Data models ensure structured, consistent, and interoperable data, facilitating collaboration, integration, and validation across domains. Tools such as MetaConfigurator simplify the creation and maintenance of these models. Based on the real-world biochemistry use case and discussions with users, several limitations of the tool were addressed. Key improvements include a more user-friendly schema editor, CSV import with schema inference, snapshot sharing, ontology integration, and a graphical schema editor for visual schema manipulation. The application on the biochemistry case demonstrates how, with MetaConfigurator, users with limited technical expertise can transition from Excel to JSON and create their own data models successfully.

Future work could refine ontology integration, enhance the graphical schema editor, and add features, such as schema satisfiability checks. Introducing AI capabilities, such as using natural language prompts for schema transformation or generating source code, is a promising direction. Conducting larger user studies could provide further insights about usability and adoption. By evolving with user needs, MetaConfigurator can remain a key tool for simplifying data modeling and supporting interdisciplinary collaboration.

Acknowledgements

We thank the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) for supporting this work by funding – EXC2075 – 390740016 under Germany’s Excellence Strategy. Furthermore, we acknowledge the support of the Stuttgart Center for Simulation Science (SimTech).

We thank Esengül Ciftci (Max Planck Institute for Solid State Research, Stuttgart, Germany) for providing a dataset and for evaluating MetaConfigurator.

Finally, we thank Torsten Gieß (University of Stuttgart Institute of Biochemistry and Technical Biochemistry, Germany) for providing Python code to query chemical properties from the PubChem database.

Bibliography

- [At22] Attouche, Lyes; Baazizi, Mohamed-Amine; Colazzo, Dario; Ghelli, Giorgio; Sartiani, Carlo; Scherzinger, Stefanie: Witness generation for JSON Schema. arXiv preprint arXiv:2202.12849, 2022.
- [At24] Attouche, Lyes; Baazizi, Mohamed-Amine; Colazzo, Dario; Ghelli, Giorgio; Sartiani, Carlo; Scherzinger, Stefanie: Validation of modern JSON schema: Formalization and complexity. *Proceedings of the ACM on Programming Languages*, 8(POPL):1451–1481, 2024.
- [Ba59] Backus, John W: The syntax and the semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference. In: *ICIP Proceedings*. pp. 125–132, 1959.
- [Ba19] Baazizi, Mohamed-Amine; Colazzo, Dario; Ghelli, Giorgio; Sartiani, Carlo: Parametric schema inference for massive JSON datasets. *The VLDB Journal*, 28:497–521, 2019.
- [Ba21] Baazizi, Mohamed-Amine; Colazzo, Dario; Ghelli, Giorgio; Sartiani, Carlo; Scherzinger, Stefanie: An Empirical Study on the “Usage of Not” in Real-World JSON Schema Documents (Long Version), 2021.
- [BKEdN09] Ben-Kiki, Oren; Evans, Clark; döt Net, Ingy: YAML ain’t markup language version 1.2. Available on: <http://yaml.org/spec/1.2/spec.html>, 2009.
- [Ch56] Chomsky, Noam: Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.
- [Ch76] Chen, Peter Pin-Shan: The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)*, 1(1):9–36, 1976.
- [Ch77] Chen, Peter Pin-Shan: The entity-relationship model: a basis for the enterprise view of data. In: *Proceedings of the June 13-16, 1977, National Computer Conference. AFIPS ’77*, Association for Computing Machinery, New York, NY, USA, p. 77–84, 1977.
- [Cr06] Crockford, D: The application/json Media Type for JavaScript Object Notation (JSON). Internet Engineering Task Force IETF Request for Comments, 2006.
- [ČS20] Čontoš, Pavel; Svoboda, Martin: JSON schema inference approaches. In: *International Conference on Conceptual Modeling*. Springer, pp. 173–183, 2020.
- [Er03] Eriksson, Hans-Erik; Penker, Magnus; Lyons, Brian; Fado, David: *UML 2 toolkit*. John Wiley & Sons, 2003.
- [He11] Heath, Tom: *Linked data: Evolving the Web into a global data space*. Morgan & Claypool, 2011.
- [Ki16] Kim, Sunghwan; Thiessen, Paul A; Bolton, Evan E; Chen, Jie; Fu, Gang; Gindulyte, Asta; Han, Lianyi; He, Jane; He, Siqian; Shoemaker, Benjamin A et al.: PubChem substance and compound databases. *Nucleic acids research*, 44(D1):D1202–D1213, 2016.
- [Kn68] Knuth, Donald E: Semantics of context-free languages. *Mathematical systems theory*, 2(2):127–145, 1968.

- [Ko19] Koutkias, Vassilis: From data silos to standardized, linked, and FAIR data for pharmacovigilance: current advances and challenges with observational healthcare data. *Drug Safety*, 42(5):583–586, 2019.
- [Ne24a] Neubauer, Felix: Data model creation with MetaConfigurator. Master’s thesis, University of Stuttgart, 2024.
- [Ne24b] Neubauer, Felix; Bredl, Paul; Xu, Minye; Patel, Keyuriben; Pleiss, Jürgen; Uekermann, Benjamin: MetaConfigurator: A User-Friendly Tool for Editing Structured Data Files. *Datenbank-Spektrum*, pp. 1–9, 2024.
- [OE15] OECD: Data-Driven Innovation. 2015.
- [Pa19] Patel, Jayesh: Bridging data silos using big data integration. *International Journal of Database Management Systems*, 11(3):01–06, 2019.
- [Pe16] Pezoa, Felipe; Reutter, Juan L; Suarez, Fernando; Ugarte, Martín; Vrgoč, Domagoj: Foundations of JSON schema. In: *Proceedings of the 25th international conference on World Wide Web*. pp. 263–273, 2016.
- [PP05] Pilone, Dan; Pitman, Neil: UML 2.0 in a Nutshell. Ö’Reilly Media, Inc.", 2005.
- [RJB04] Rumbaugh, James; Jacobson, Ivar; Booch, Grady: Unified Modeling Language Reference Manual, The (2nd Edition). Pearson Higher Education, 2004.
- [Sc10] Schadt, Eric E; Linderman, Michael D; Sorenson, Jon; Lee, Lawrence; Nolan, Garry P: Computational solutions to large-scale data management and analysis. *Nature reviews genetics*, 11(9):647–657, 2010.
- [SEP95] Song, Il-Yeol; Evans, Mary; Park, Eun K: A comparative analysis of entity-relationship diagrams. *Journal of Computer and Software Engineering*, 3(4):427–459, 1995.
- [SSB22] Siffa, Ihda Chaerony; Schäfer, Jan; Becker, Markus M: Adamant: a JSON schema-based metadata editor for research data management workflows. *F1000Research*, 11, 2022.
- [TYF86] Teorey, Toby J; Yang, Dongqing; Fry, James P: A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys (CSUR)*, 18(2):197–222, 1986.
- [Wa20] Waghray, Kunal: JSON schema matching: Empirical observations. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. pp. 2887–2889, 2020.
- [WS96] Wang, Richard Y; Strong, Diane M: Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33, 1996.
- [Ye06] Yergeau, Francois; Bray, Tim; Paoli, Jean; Sperberg-McQueen, CM; Maler, Eve: Extensible markup language (xml) 1.0. In: *XML*. volume 1. Citeseer, p. W3C, 2006.