

A framework for data specification modeling in complex domains

Štěpán Stenclák¹

¹*Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic*

Abstract

Correctly and completely describing the structure and semantics of data is laborious but necessary for complex domains, critical systems, or interoperability at the national and international levels where potential errors from misinterpretation of data could be costly. However, traditional data modeling processes often do not consider more complex approaches where structure and semantics may be designed separately, derived from already existing models, or composed from multiple models from different teams. We propose a robust framework inspired by a model-driven approach to support and ease the development and management of data specifications mapped to semantically or structurally richer models and keep them consistent through evolution.

Keywords

conceptual models, semantics, ontologies, data specification

1. Problem

A data specification describes the structure and semantics of data stored within systems or used for communication between various parties. The structure determines how the data are represented, whereas semantics defines meaning and how the concepts should be interpreted. The degree of explicitness of expressing structure and semantics depends on a specific use case. In the case of small domains, the data itself can implicitly describe its structure, and the structure can describe the semantics if the person knows the domain well.

However, in complex domains where potential design flaws would be costly, the precise description of both syntax and semantics may be essential [1]. As the syntax is technical, it is often sufficient to provide a technical schema (such as JSON or SQL Schema). However, for semantics, besides the technical specification (in the form of ontology, for example), documentation with diagrams and examples may be helpful. Some domains may benefit from additional documents, such as application skeletons, API specifications, tests, etc.

Manually designing such data specifications is a laborious and error-prone task. It may require monotone work with copying names and definitions, making it easy to miss concepts or misplace texts. The designer needs to know the semantics well and be able to create all necessary parts of the data specification. The latter task can be handled by numerous tools


ER2023: Companion Proceedings of the 42nd International Conference on Conceptual Modeling: ER Forum, 7th SCME, Project Exhibitions, Posters and Demos, and Doctoral Consortium, November 06-09, 2023, Lisbon, Portugal

✉ stepan.stenclak@mffyz.cuni.cz (Š. Stenclák)

🆔 0000-0003-4843-2470 (Š. Stenclák)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

automatically generating various documents. For the first task, numerous tools have emerged, typically based on the model-driven approach [2], which assists with the design by deriving the structure from the description of the semantics. However, we observe a prevalent issue with these tools leading to poor practice. They typically do not adequately separate the semantics and structure, thereby forcing users to think about semantics in the context of structure. The lack of separation often results in poor-quality models that do not reflect reality. For example, the designer may merge two concepts for optimization reasons, even if they are semantically different. The ideal way is to have the so-called semantic model [3], created by someone with the knowledge of the domain, isolated from the process of designing a structural model [1, 4, 5], where more technical knowledge is needed. By structural model, we mean a formal description of data structure; for example, a database schema (often referred to as conceptual model) or a schema of a serialization format, such as JSON or XML.

However, there is little availability of tools [6, 7] that can handle the separation, keeping the models consistent and up-to-date, or well-defined methodologies for managing the process.

The strict two-level separation of semantic and structural models is not always sufficient. In some cases, it may be beneficial to reuse existing models instead of creating them from scratch. These are often called application profiles [8] that define a model for use in specific application scenarios but are based on more generic models.

An example is DCAT-AP¹. DCAT [9] is the Data Catalog Vocabulary from W3C designed to facilitate interoperability between data catalogs published on the Web. DCAT-AP is then an extension of the vocabulary for the needs of the European Union. DCAT-AP-CZ [10] for the Czech Republic is derived from it. Each adds or changes some of its specifics. From DCAT-AP-CZ is then derived a specific data specification defining a JSON data structure [11], but with the semantics described in DCAT, DCAT-AP, and DCAT-AP-CZ. See Figure 1a.

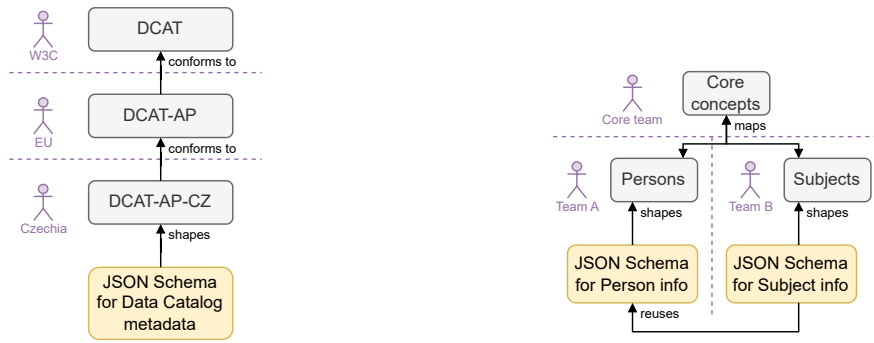
Because of the lack of available tools, the application profiles are manually created [10], which brings similar issues with consistency, synchronization, and difficulty in deriving structural models from the given semantics. For example, any change in DCAT must be propagated manually to data that conforms to the JSON Schema according to DCAT-AP-CZ.

Heery and Patel [8] do not restrict application profiles to only one source model. During the development of systems based on the microservice architecture [12], multiple independent models may arise. If there is a need to connect data across multiple services, the individual models must be aligned and merged.

We did not specify the type of models the application profiles work with, as the definition may be interpreted freely. For example, application profiles may extend the semantics or the structure. This leads us to another real-world use of extending a concrete structure (for example, JSON Schema) for more specific applications. See Figure 1b, where the structural model for *Subjects* reuses *Persons*.

This motivates us to design a metamodel that can describe various representations of semantic and structural models and mappings between them as they can be derived from each other in the form of application profiles and ensure consistency and change propagation if the model changes. Moreover, the mapping must be robust enough to give designers of the application

¹<https://joinup.ec.europa.eu/collection/semic-support-centre/solution/dcat-application-profile-data-portals-europe/release/300>



(a) Data Catalog Vocabulary and its Application Profiles for use in Czechia’s National Open Data Catalog.

(b) Subset of the study information system for persons and school subjects management modules with schemas for provided APIs.

Figure 1: Real-life examples of using semantic models (in grey) managed by different teams together with structural models (in yellow).

profiles enough freedom they have when they follow the now common path of designing a simple data schema in which they can do whatever they want and is not constrained by external semantic models. We also plan to design a methodology on how these application profiles should be properly designed as there are currently no well defined approaches [4].

To support our research, we will also develop an ecosystem of tools that will support designers in creating application profiles. As a complete example, consider the following:

1. A domain expert decides to create a semantic model for their domain and derive a few JSON Schemas from it describing data exchanged between the system. They may create a semantic model from scratch, but they decide to use Wikidata [13] and Schema.org as it already describes most of their domain. This means they may choose concepts that will be included, may change them, or add new ones as only sometimes the existing models describe the reality as wanted. They may want to create a documentation website from the finished semantic model with diagrams and detailed explanations of individual concepts.
2. If the domain is large, the model may be further extended or modified by other teams that need more detail or have different expectations.
3. Data designers may then create structural models that shape the domain’s semantic model. Each structural model is then used to generate schemas for given formats, documentation, examples, and other artifacts.
4. Others may extend the structural models by creating their own or use them as is in other schemas.

2. Related work

This work is inspired by the author’s supervisor and his team’s previous research [14, 15, 16, 17] from 2012 to 2015. The team built on the well-defined concepts of model-driven, model weaving,

and evolution [2]. They considered two models, semantic as platform-independent and structural platform-specific, to design XML [18] Schemas from a shared semantic model. However, some model-driven concepts may no longer apply as we consider more robust solutions where application profiles only patch the source model, and the models may be linked under different contexts.

Some European countries are pursuing a similar goal, although more focused on developing tools, for publishing data specifications to describe open data according to the New European Interoperability Framework guide [4]. In some parts, it identifies the issues described in section 1, thus confirming the correct direction of our research and the possible application in practice. However, the quality of these tools is often low regarding the general and easy use outside their domain. An example is the Finnish tool Tietomallit² or a Belgian toolchain for technical standardization on a national level OSLO³. Both tools only focus on a specific use-case as their main goal is to support publishing open data in closed ecosystem.

Many approaches solve the partial problem of this work. Research in model-driven engineering [2] can be considered regarding models' transformation when creating structural models from semantics and evolution and change propagation between various application profiles. There is research in ontology engineering [19] that can cover the semantics. There are also practical tools, such as LinkML [20], that handle the transformation of platform-independent structural models to various formats.

3. Proposed solution

We propose a solution inspired by a model-driven approach. Any semantic or structural model managed atomically by a single entity is a model from a model-driven standpoint. In Figure 1, all rectangles are individual models, and we also consider external models, such as one constructed in Enterprise Architect and stored in an SQL database.

Then, the problems addressed in the previous chapter can be divided into three groups. (i) Change propagation, model derivation from more general models, and model management can be addressed by a generic model-driven framework. (ii) We then build tools for deriving semantic and (iii) structural models on top of this general framework. A high-level representation of our solution is shown in Figure 2.

To generalize the idea, by **model**, we mean a representation of either structure, semantics, or any other representation of arbitrary data. This allows us to apply the solution to other domains that may benefit from change propagation and model derivation and are not directly related to semantic or structural models. For example, consider a model that represents OpenAPI specifications⁴. Although most of the specification consists of data schemas, some operations may change in time and may be necessary to propagate changes between various specifications and other technical artifacts properly.

The aim of the generic framework is to define the metamodel of models, evolution and derivation of models.

²<https://tietomallit.suomi.fi/>

³<https://joinup.ec.europa.eu/collection/oslo-open-standards-linked-organisations-0/about>

⁴<https://swagger.io/specification/>

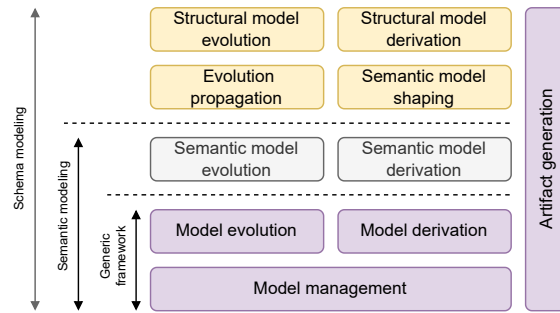


Figure 2: Proposed decomposition of the project into logical units with a generic framework on which semantic modeling and schema modeling tools will be built.

To achieve that models can be extended and modified, we introduce a **model derivation** as seen in Figure 1. Individual models will refer to each other, creating a rooted **model graph** with the model of a question as the root. According to the type of reference and based on the context, the information from the referenced models will be interpreted differently, resulting in a single aggregated model with the desired properties. This does not mean that the base model contains all the content but rather consists of another model. Depending on the situation, we may treat the base model as an aggregation, hence the full model. This would mean, we can generate, for example, a documentation of all concepts with diagrams, or just as a patch that fixes the included model and generate only the patch documentation. The base model may overwrite, modify or delete entities from the included model in a form of patches or include only explicitly listed entities. This may be beneficial, for example, with the Wikidata model, as it is too large, and only part of it may be relevant in a given situation.

Individual types of entities that will be stored in the model depend on the model type and use, and their exact list and function are out of the scope of this paper. To briefly mention them, the semantic model will contain classes and associations; the structural model will contain objects and their properties, choices and references. If the model modifies an existing concept, it will be a patch. Other useful entities include various types of mappings, such as sameAs, or more complex ones between multiple entities.

Models can be arranged in the model graph in various ways depending of the intent of the modeler. Let us imagine a situation where we will make an application profile to a semantic model. In this case, we have two models - the original model, which may be external, and the model that patches it. The newly created concepts then need to be stored in the new model. However, it may make sense to put any bug fixes into a new model that can be later applied to the original model.

Alternatively, we may have multiple external models as a source. Then it may be helpful to create a model for potential bug fixes for each external model and then one model that aggregates all of them and provides additional entities that do not belong to any of the source models.

If there is a designated model for bug fixes, it should be possible to generate a patch to the original model or directly commit the changes if the model would support this.

Two sub-activities will be addressed regarding the evolution: model versioning and change

propagation. Versioning is only possible for models created within the tools and not externally. The purpose is to keep the whole model history and allow the change propagation to be executed later, not immediately, when change occurs. Change propagation will then be handled individually within the semantic and structural models. Each change to the model will have to be made using a predefined operation. With the knowledge of the operations, it will be possible to apply this list of operations to other models or translate them to a different set of operations for models of other types.

For external models that are not versioned, we would have to obtain the list of operations manually. We can address this by using another model as a mirror. By comparing these models, we can infer changes that can then be propagated. In some cases, this may not be necessary. Then, if the inconsistency is found, users would have to fix it by themselves.

The individual models may serve as the output if they are in the desired format. Although we plan to introduce a native format for storing the models, users may use adapters for other formats if the given format disposes of the required functionalities. Then we call the model to be **external**. External models can be a common use case for representing semantic models in various formats, such as Enterprise Architect or Draw.io.

However, in most scenarios, users may want to **generate artifacts** from models using various generators. For example, a JSON Schema from a structural model, RDFS [21] from a semantic model, documentation, etc. From an architectural perspective, three types of generators can be distinguished.

- *single model* - generates artifacts from one model in a specific version. Either directly from the model or by aggregating it with all models from the model graph,
- *evolution* - generates from a sequence of changes that are propagated from the model, for example, SQL UPDATE statements or an HTML document with a list of changes in a human-readable form,
- *diff* - the difference between two models, such as transformation scripts that produce data that conforms to one structural model from data that conforms to the other. This, however, is the most challenging part of the project and, in many use cases, unnecessary.

4. Current status

The project started with the author's master thesis, which aimed to develop a proof-of-concept tool called Dataspecer [22] to facilitate and improve the work of creating so-called Formal Open Standards (FOSes) [23] in the Czech Republic. FOSes are data specifications for open data publishers and public institutions. Currently, the tool can effectively replace most of the original manual work and is actively used for that. It generates HTML documentation of the concepts with diagrams, JSON Schema [11], JSON-LD [24], XSD [25], CSVW [26], SPARQL queries [27], and XSLT [28] transformation scripts between XML and RDF [29] representation.

A key consideration during the tool's development was user-friendliness. It reads an external semantic model and suggests concepts that can be used to construct the schema in a graphical bullet list-like format. See Figure 3. It lets users rename keys and move properties in the tree to create schemas that fit their needs. The format is not tied to a specific technology but rather

generic. Therefore it is possible to generate JSON, XML, and CSV schemas from the same structural model. Possible format-specific constructs can be added, but our proposed model is sufficient for all the FOSes we are working on.

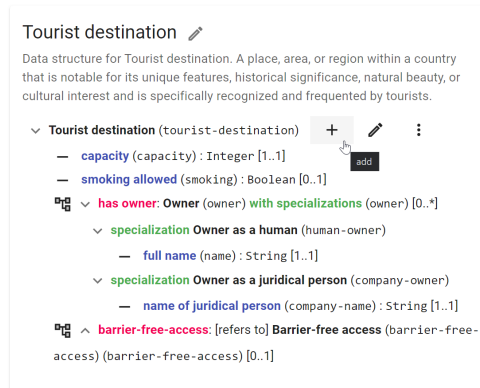


Figure 3: Screenshot of Dataspecer tool with a structural model for tourist destinations. This is a human-friendly view, where the owner can be of two types. From this, for example, a JSON Schema with a oneOf construct can be generated together with HTML documentation describing all used concepts.

Regarding the graph model described in section 3, the three-layer architecture depicted in Figure 4 is currently embedded in Dataspecer. The user chooses the source of the external semantic model. A new, empty model is created underneath, which is used to patch changes and simultaneously, copies all concepts used in case the external semantic model is unavailable. Underneath are the individual structural models from which the tool generates the artifacts, as mentioned earlier.

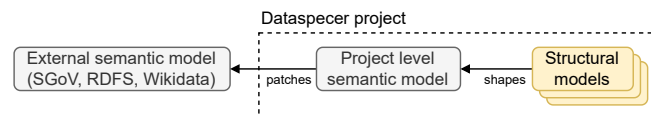


Figure 4: The current fixed graph model of Dataspecer.

Although this architecture is sufficient for many cases, we would like to generalize it for other scenarios, such as instances where the user works with multiple semantic models or wants to create their own model during schema modeling.

We use the Czech Semantic Government Vocabulary (SGoV) [30] as the external semantic model. It contains concepts from Czech laws and important domains, their definitions, and relations. For example, it contains concepts about Tourist destinations that are used in FOS to prescribe a JSON structure for publishing open data about them. A typical publisher is a local city administration that needs to attract visitors by providing standardized data that various online mapping and planning services can consume. It also contains more than 40 different properties characterizing the barrier-free accessibility of a public place that requires a deeper understanding of the needs of people with disabilities. Hence it would be too demanding for

one person to design a structure without semantics defined in advance.

We are also actively working with several students on other generators such as SHACL, sample application generator, or example data generator. We are also working on a Wikidata adapter as an external semantic model.

In addition to a schema generation tool, we want to focus on a tool that would work primarily with semantic models - load external ones, create application profiles, and publish those in different formats and technologies according to section 3.

In the following year (the 2nd year of the author's Ph.D. studies), the author plans to complete all parts of the generic framework, redesign Dataspecer to support a full model graph, and coordinate the development of the semantic modeling tool. In the next year, we plan to start working on evolution and methodology based on the feedback we get. We will also optimize the developed tools for use in practice. In the last year, we want to analyze and implement mappings between different concepts and further work on integrating the tools into practice.

From a publishing point of view, the author wants to describe the structure of the models and how they can be applied in different domains, the principle of evolution, and mapping.

5. Plan for evaluation and validation

The primary part of the evaluation and validation will be the use of the tools, that will use the proposed metamodel, in practice. Specifically, several data experts will actively use the tool for schema generation in the context of FOSes development and management for Czechia. Evolution will also be used in long-term management, as the source semantic models continuously change. Occasionally, the semantic model creation tool will also be used to create application profiles.

The author will be actively using the semantic model management tool to manage concepts from the business analysis as he is involved in the development of a new student information system. Due to the size of the project, multiple semantic models will be created and need to be synchronized. This corresponds to Figure 1b with several semantic models and one main model containing the definition of the core concepts. Subsequently, Dataspecer will be used to create API specifications for the interfaces between the different modules of the system. As the system is incrementally rewritten, the semantic model will evolve. The changes will need to be documented, and API updates will need to be issued accordingly. We expect the tools to do most of the work automatically.

We also actively seek involvement in other projects, such as European Union's SEMIC, where data specifications are being developed.

We then plan to conduct a methodological survey among users to determine whether these tools make their work more effortless than a manual approach, whether all use cases can be covered, and whether manual intervention in the model design process is still needed.

Acknowledgments

I am grateful to my advisor, doc. Mgr. Martin Nečaský, Ph.D., for his supervision of my research and to other members of our research group for valuable discussions. This research was supported by the Charles University project GAUK no. 262823 and SVV project 260 698.

References

- [1] A. Doan, A. Y. Halevy, Z. G. Ives, Principles of Data Integration, Morgan Kaufmann, 2012. URL: <http://research.cs.wisc.edu/dibook/>.
- [2] D. C. Schmidt, Guest Editor's Introduction: Model-Driven Engineering, Computer 39 (2006) 25–31. doi:10.1109/MC.2006.58.
- [3] J. Peckham, F. Maryanski, Semantic data models, ACM Computing Surveys (CSUR) 20 (1988) 153–189.
- [4] E. Commission, D.-G. for Informatics, New European interoperability framework : promoting seamless services and data flows for European public administrations, Publications Office, 2017. doi:doi/10.2799/78681.
- [5] View, Towards a model-driven organization (part 1) – anchor modeling, 2022. URL: <https://www.anchor modeling.com/towards-a-model-driven-organization-part-1/>.
- [6] F. Rademacher, J. Sorgalla, S. Sachweh, Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective, IEEE Softw. 35 (2018) 36–43. doi:10.1109/MS.2018.2141028.
- [7] P. Espinoza-Arias, D. Garijo, Ó. Corcho, Crossing the chasm between ontology engineering and application development: A survey, J. Web Semant. 70 (2021) 100655. doi:10.1016/j.websem.2021.100655.
- [8] rachel heery, manjula patel, Application profiles: Mixing and matching metadata schemas, Ariadne (2000). URL: <http://www.ariadne.ac.uk/issue/25/app-profiles/>.
- [9] D. Browning, S. Cox, R. Albertoni, A. Perego, A. G. Beltran, P. Winstanley, Data Catalog Vocabulary (DCAT) - Version 2, W3C Recommendation, W3C, 2020. URL: <https://www.w3.org/TR/2020/REC-vocab-dcat-2-20200204/>.
- [10] J. Klímek, Dcat-ap representation of czech national open data catalog and its impact, Journal of Web Semantics 55 (2019) 69–85.
- [11] A. Wright, H. Andrews, B. Hutton, G. Dennis, JSON Schema: A Media Type for Describing JSON Documents, Internet-Draft draft-bhutton-json-schema-01, Internet Engineering Task Force, 2022. URL: <https://datatracker.ietf.org/doc/draft-bhutton-json-schema/01/>, work in Progress.
- [12] S. Newman, Building Microservices, O'Reilly Media, Inc., 2021. URL: <https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/>.
- [13] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Communications of the ACM 57 (2014) 78–85.
- [14] M. Nečaský, I. Mlýnková, J. Klímek, J. Malý, When conceptual model meets grammar: A dual approach to XML data modeling, Data & Knowledge Engineering 72 (2012) 1–30. doi:10.1016/j.datak.2011.09.002.
- [15] M. Nečaský, J. Klímek, J. Malý, I. Mlýnková, Evolution and change management of XML-based systems, Journal of Systems and Software 85 (2012) 683–707.
- [16] J. Klímek, L. Kopenec, P. Loupal, J. Malý, XCase - A Tool for Conceptual XML Data Modeling, in: Advances in Databases and Information Systems, Associated Workshops and Doctoral Consortium of the 13th East European Conference, ADBIS 2009, Riga, Latvia, September 7-10, 2009. Revised Selected Papers, volume 5968 of LNCS, Springer, 2009, pp. 96–103. doi:10.1007/978-3-642-12082-4_13.
- [17] J. Klímek, J. Malý, M. Nečaský, I. Holubová, eXolutio: Methodology for Design and

- Evolution of XML Schemas Using Conceptual Modeling, *Informatica* 26 (2015) 453–472. URL: <https://content.iospress.com/articles/informatica/inf1065>.
- [18] M. Sperberg-McQueen, E. Maler, F. Yergeau, T. Bray, J. Paoli, Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation, W3C, 2008. URL: <https://www.w3.org/TR/2008/REC-xml-20081126/>.
- [19] T. Tudorache, Ontology engineering: Current state, challenges, and future directions, *Semantic Web* 11 (2020) 125–138. doi:10.3233/SW-190382.
- [20] S. A. T. Moxon, H. Solbrig, D. R. Unni, D. Jiao, R. M. Bruskiwich, J. P. Ballhoff, G. Vaidya, W. D. Duncan, H. Hegde, M. Miller, M. H. Brush, N. L. Harris, M. A. Haendel, C. J. Mungall, The Linked Data Modeling Language (LinkML): A General-Purpose Data Modeling Framework Grounded in Machine-Readable Semantics, in: J. Hastings, A. Barton (Eds.), *Proceedings of the International Conference on Biomedical Ontologies 2021 co-located with the Workshop on Ontologies for the Behavioural and Social Sciences (OntoBess 2021) as part of the Bolzano Summer of Knowledge (BOSK 2021)*, Bozen-Bolzano, Italy, 16-18 September, 2021, volume 3073 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 148–151. URL: <https://ceur-ws.org/Vol-3073/paper24.pdf>.
- [21] D. Brickley, R. Guha, RDF Schema 1.1, W3C Recommendation, W3C, 2014. URL: <https://www.w3.org/TR/2014/REC-rdf-schema-20140225/>.
- [22] Š. Stenclák, M. Nečaský, P. Škoda, J. Klímek, Dataspecer: A model-driven approach to managing data specifications, in: *European Semantic Web Conference*, Springer, 2022, pp. 52–56.
- [23] Directive (EU) 2019/1024 of the European Parliament and of the Council of 20 June 2019 on open data and the re-use of public sector information, URL: <http://data.europa.eu/eli/dir/2019/1024/oj>.
- [24] P.-A. Champin, D. Longley, G. Kellogg, JSON-LD 1.1, W3C Recommendation, W3C, 2020. URL: <https://www.w3.org/TR/2020/REC-json-ld11-20200716/>.
- [25] H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, M. Sperberg-McQueen, S. Gao, W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures, W3C Recommendation, W3C, 2012. URL: <https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/>.
- [26] G. Kellogg, J. Tennison, Model for Tabular Data and Metadata on the Web, W3C Recommendation, W3C, 2015. URL: <https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/>.
- [27] A. Seaborne, S. Harris, SPARQL 1.1 Query Language, W3C Recommendation, W3C, 2013. <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/>.
- [28] M. Kay, XSL Transformations (XSLT) Version 3.0, W3C Recommendation, W3C, 2017. <https://www.w3.org/TR/2017/REC-xslt-30-20170608/>.
- [29] Y. Raimond, G. Schreiber, RDF 1.1 Primer, W3C Note, W3C, 2014. URL: <https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/>.
- [30] P. Křemen, M. Nečaský, Improving discoverability of open government data with rich metadata descriptions using semantic government vocabulary, *J. Web Semant.* 55 (2019) 1–20. doi:10.1016/j.websem.2018.12.009.