Semantic Web journal
XX(X):1–20

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Dataspecer: Development of Consistent Semantic Data Specification Ecosystems

Štěpán Stenchlák, Jakub Klímek, Petr Škoda and Martin Nečaský[1]

Abstract

To achieve interoperability for effective data exchange on the web, we need a contract. Depending on the field, we may refer to technical data schemas, web vocabularies, or, more generally, to data specifications (DS). The development and management of these DSes can become difficult in complex domains with multiple stakeholders and related DSes involved. In this paper, we present Dataspecer, an open-source, modular web application for the development of semantic data specifications (SDSes), DSes that target the semantic and technical layers of data exchange. Dataspecer allows users to design web vocabularies and their application profiles, maintaining relations between reused concepts and their original SDSes. Furthermore, Dataspecer assists users in the creation of technical artifacts such as schemas for JSON or XML, while maintaining consistency of the artifacts with the application profiles. We motivate the need for SDSes and derive requirements for such a tool. In case studies based on the ecosystem of DCAT-based specifications, we demonstrate that SDSes created in Dataspecer meet these requirements and are of higher quality. We show SDSes that were created directly in Dataspecer, and in the evaluation section, we argue that using our tool is more efficient than creating them manually, even for smaller domains.

Keywords

Semantic Interoperability, Technical Interoperability, Semantic Data Specification, Vocabularies, Application Profiles

Introduction

As businesses grow and the number of systems they employ increases, or when multiple independent organizations coexist within a shared environment, data interoperability becomes increasingly critical. Consistent and accurate data exchange, unified terminology used, and support for the development of scalable, maintainable systems that rely on a common understanding of the exchanged data are all effects of high data interoperability.

Achieving interoperability requires that all parties adopt a shared contract. Developing such contracts requires a methodological approach, recognized in communities that participate in model-driven development, conceptual modeling, and data governance (Atkinson and Kühne (2003); Guarino et al. (2009)). These communities converge on the idea that interoperability must be addressed both (i) **semantically**, by providing meaning - usually by vocabularies and ontologies, and (ii) **technically**, by ensuring that data is correctly exchanged in machine-readable formats - usually validated by technical schemas (Masmoudi et al. (2024); Bayerlein et al. (2024); Jordan et al. (2025); Vogt et al. (2025)). We refer to such contracts as *semantic data specifications* (SDS).

In practice, we often find that SDSes reuse other, already existing specifications. Reusing known concepts helps consumers understand the specification since they may already be familiar with the reused concepts. This approach is widely adopted, for example, in the world of web vocabularies (Noy et al. (2001); Corcho et al. (2005)). However, some specifications go even further and specify how to apply reused concepts in their target application domain by adding additional contextual information (Park and Childress (2009)). Historically, such refinements were handled ad hoc, lacking a more systematic approach. Specifications that reuse existing concepts while providing

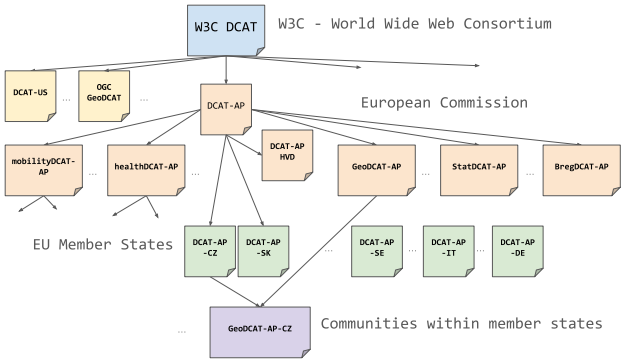

Figure 1. An example of an application profile (AP) hierarchy of the Data Catalog Vocabulary (DCAT) (Perego et al. (2024)). Lower profiles reuse concepts from the higher ones and apply them in their specific application scenario, often with restrictions and terminological refinements.

[1]Charles University, Faculty of Mathematics and Physics, Department of Software Engineering, Prague, Czechia

Corresponding author:
Štěpán Stenchlák, Charles University, Faculty of Mathematics and Physics, Department of Software Engineering, Malostranské nám. 25, 118 00 Prague, Czechia
Email: stepan.stenchlak@matfyz.cuni.cz

them with contextual meaning within a specific application domain are called *application profiles* (APs) (Heery and Patel (2000)), which can form hierarchies similar to the one in the data catalog domain (Figure 1). To support the structured description of such profiles, we have published the *Data Specification Vocabulary* (DSV) (Klímek et al. (2025b,c)) - a vocabulary for describing the reused concepts, how they should be used in the given application scenario, to which vocabularies they belong, and metadata about the individual specification's parts and reused specifications.

A variety of methodologies and frameworks have emerged to address this challenge of reuse in the form of application profiles. In particular, the European Interoperability Framework (EIF)[1] and the SEMIC Style Guide[2] provide guidance and methods to operationalize the principles of interoperability, with an emphasis on the reuse of existing semantics. However, they lack proper tooling and technical standards to support the authoring of such SDS.

Although Semantic Web technologies are, in theory, sufficient to ensure technical interoperability, many systems still rely on conventional data formats such as JSON or XML. These formats require a systematically defined schema that represents the structure of the data.

The manual design of SDSes is a demanding and monotonous process. It requires knowledge of multiple technologies and a good grasp of the domain being described. In complex domains, having high-quality SDSes is essential to prevent errors during integration.

Therefore, in this paper, we distinguish three forms of SDSes (formally defined in Motivation and Requirements): (i) *vocabularies*, which define terms and their relationships, (ii) *application profiles* (APs), which select and refine terms from existing SDSes for a specific domain, and (iii) *technical specifications*, which define the data structure in a given format.

To address the challenge of creating and maintaining high-quality coherent semantic data specifications, we introduce Dataspecer, a web application for authoring and managing high-quality semantic data specifications, specifically:

- web vocabularies in the context of other existing vocabularies and ontologies,
- application profiles of other vocabularies and application profiles by selecting concepts and contextualizing their definitions for a given application domain,
- interlinked technical schemas for formats such as XML, JSON, and CSV while maintaining strict separation between them and the semantics they were derived from.

Building on our previous work on the generation of technical schemas from ontologies (Stenchlák et al. (2022)) and the creation of vocabularies and application profiles (Klímek et al. (2024)), in this article we provide a comprehensive presentation of the tool as a whole. We show how these functionalities are integrated, revisit their core principles, and demonstrate the tool's complete functionality on real-world specifications, highlighting its maturity and applicability. We also evaluate the suitability of the tool for practical use.

In the following chapter, Motivation and Requirements, we introduce the key concepts and motivate the requirements for such a tool. In the Dataspecer chapter, we present the Dataspecer tool and its key architectural decisions. In the subsequent chapter Related Work, we compare the requirements with existing tools and show their shortcomings in our use case. Chapters Evaluation: Improvement of DCAT Application Profiles and Evaluation: Other Specifications Created in Dataspecer present real SDSs created in Dataspecer, either by us or by our users, including already existing specifications, thereby demonstrating the usability of the tool. In Evaluation: Productivity and Usability, we show systematically that even for small cases, the use of Dataspecer is advantageous, and we conclude the article with Conclusions and Future Work.

Motivation and Requirements

In this section, we first explain why semantic data specifications (SDSes) are important and why they should be of high quality. Next, we introduce other core terms that we will use. Finally, we present the requirements for a tool that supports the development and maintenance of SDSes.

Semantic Data Specification (SDS)

A semantic data specification (SDS) is a contract for data exchange that defines the (i) **intended meaning** of the data elements in terms of a shared conceptualization of the domain it describes, (ii) the **constraints** on how the data can be related, and the (iii) **structure**, i.e. the format of the representation of the data. We will refer to the last point as the technical level as it ensures technical interoperability by specifying how to structure and transfer the data. The first two points ensure semantic interoperability by providing *semantics*, and are crucial for understanding the given domain and its concepts, so that the exchanged data can be interpreted correctly. While the technical level can be omitted if the contract's purpose is only to define the semantics for further use, the semantic level must always be included to some extent; otherwise, it describes something that has no defined meaning.

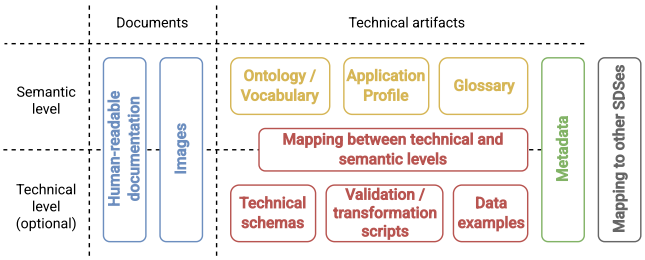

Figure 2. Example of complex SDS manifestation targeting both semantic and technical levels. SDSes typically consist of human-readable documents, machine-readable technical artifacts, and metadata for machine processability with mapping to other SDSes. The list of SDS parts is non-exhaustive.

For small domains, the manifestation of an SDS can be just a verbal agreement or may not even exist, if the SDS is simple. However, in complex domains, especially when multiple parties are involved, SDSes must be more descriptive and formal. The formal description helps

stakeholders avoid misunderstandings that could cause costly errors and complicate integrations (Tan et al. (2024)).

For the purpose of this paper, we consider SDSes as a collection of interlinked (i) documents with attachments, and (ii) technical artifacts, including metadata, as can be seen in Figure 2. The documents target human readers; the technical artifacts are designed for machine processing.

We will target three types of SDSes that correspond to three use cases that will be described in more detail further in the paper: (i) vocabularies, which focus mainly on the definition of terms and their relationships, (ii) application profiles, which select concepts from other vocabularies and application profiles, add constraints and terminological specifications, to be used in a specific application scenario, and (iii) technical SDSes, describing the structure of data.

In Linked Data ecosystem, **(web) vocabularies**[3] are sets of classes and properties that provide shared meanings for describing data in RDF. Vocabularies are described using RDFS (RDF Schema; Guha and Brickley (2014)) sometimes extended with OWL (Wood et al. (2014)) statements that can be used for more descriptive vocabularies, often referred to as ontologies.

Reuse and Application Profiles (AP)

One of the key principles for achieving interoperability is the *reuse* of existing, usually more established, SDSes. Those SDSes are often already proven in practice, so reusing them reduces the risk of errors and increases confidence. Reusing semantic concepts increases the chance that the other party may already understand them, thus simplifying the whole integration process on the semantic level. Reusing technical artifacts or data schemas, if they are suitable in the given context, also eases the process.

Focusing only on semantics, two main approaches to reuse are typically found in practice: (i) **reuse** of individual terms **as is** and (ii) **reuse with modifications** for the purpose of applying the terms to a specific domain.

Reuse as-is The nature of RDF, which is used in Linked Data ecosystem, allows for this as-is reuse as vocabularies and ontologies can link to others and use their concepts as they were defined. This effectively splits the semantics horizontally between multiple SDSes. For example, if we want to describe people in our SDS, we can use the FOAF[4] ontology, which defines the concept *Person*, including its own properties. We can use directly the *Person* class as-is by creating a relation that has a *Person* as its end. Or we can create a new subclass from it, such as *Student*. This is supported by numerous methodologies for ontology creation (Noy et al. (2001); Corcho et al. (2005)) that recommend reusing terms that are already used and proven in practice.

Reuse with modifications Reuse with modifications aims to take an existing SDS, or part of it, and apply it to a specific domain or context in such a way that certain modifications may occur. For example, in the DCAT vocabulary, a property for a title of a *Distribution* is directly `dct:title` from the DCMI Metadata Terms vocabulary[5]. DCAT reuses this property, in their context, as a *Distribution title*, while the original DCMI Metadata Terms define it only as a *Title*. We can also find cases where cardinality or datatype is narrowed for the given domain.

Indirectly, this reuse with modifications is recommended by the Best Practices for Publishing Linked Data (Hyland et al. (2014)) and preferred by technically oriented experts in the community of contributors to DCAT-AP and its profiles, where properties from existing vocabularies are reused by classes from other vocabularies. Following the previous example, `dct:title` does not have a domain defined, which means that it can be used anywhere, without any side effects on inference, etc. DCAT states that this title is used with the *Dataset* class. However, this information about reuse with modifications cannot be correctly expressed in RDFS + OWL since one would need to modify the domain of the original property, which belongs to a different SDS. Technically, changing this information is possible simply by adding RDF statements about the given concept, but this may have consequences in loss of provenance information or incorrect reasoning - assuming something that is not true. Therefore, in practice, this reuse information is available only in human-readable specification documents or manifests itself in derived artifacts such as SHACL validation rules[6].

Reuse with modifications preserves IRIs of classes that are used, in contrast with subtyping, where new classes are created with new IRIs. Reuse as is with subtyping is not an issue if the other party correctly interprets the result, which means that it knows all relevant SDSes and performs the reasoning in order to derive the proper types. However, this adds complexity to the data exchange, which is why authors of SDSes prefer reuse with modifications.

The idea of not creating subclasses but rather reusing concepts with modifications is conceptually addressed by Application Profiles (AP) (Heery and Patel (2000); Baker and Coyle (2009)). An AP is an SDS that (i) selects concepts from other SDSes that will be reused, and (ii) alters or concretizes their established semantics in order to apply them in the given application domain.

APs form a hierarchy where reused SDSes are more general, targeting broader applicability (see Figure 1). The most general SDSes define only the core concepts, minimizing potential integration issues, while the more specialized extend them to a specific domain or a subdomain or *application*. The derived APs then add the necessary context for the given domain.

This idea of separating semantics to vocabularies that only define terms and APs that reuse them in a specific context aligns with real-world practice in ontologies (Corcho Associate professor et al. (2015)), where we often see *core ontologies* such as the DCMI Metadata Terms vocabulary that introduces only basic terminology without enforcing any strict rules.

As the original definition of AP is more of a guideline, we developed the Data Specification Vocabulary (DSV) (Klímek et al. (2025b,a)), which supports the representation of the necessary information regarding reuse in the context of AP, that is, in which contexts are which terms from which vocabularies reused in a data specification. Maintaining this information in machine-readable form is especially important in complex data specification hierarchies such as DCAT, DCAT-AP, DCAT-AP HVD, GeoDCAT-AP, StatDCAT-AP, DCAT-AP-CZ, etc. (see Figure 1), where

the same classes and properties are reused and refined in different ways in different contexts.

A typical setting for many SDSes is a combination of a vocabulary and an application profile, which is the effect of applying the reuse with modifications principle. The vocabulary defines new domain-specific terms, while the profile reuses existing terms. We call this combination a Default Application Profile (DAP), a term first used in a blog post[7] about application profiles in the context of the SEMIC Style Guide, and later show that some vocabularies are, in fact, DAPs.

For SDSs to be of high quality, they must comprise many different documents (see Figure 2) that are consistent with each other. Creating them by hand is a demanding and error-prone task. The designer then has to employ several technologies at once and have a thorough understanding of the domain being described. Therefore, there is a clear need for machine assistance in the SDS creation.

Requirements

We proceed by presenting three real-world use cases, on the basis of which we derive a set of requirements for the tool. Later in the evaluation, we will show that our implementation meets these requirements. These requirements are (i) a summary from us, the authors of this article, based on the needs we identified during our work in the areas of data modeling in public administration, the development of DCAT application profiles and software development, and (ii) feedback from other Dataspecer users.

Use case: Vocabularies The first use case concerns the creation of RDFS vocabularies on the Web. Examples of such vocabularies include the previously mentioned DCMI Metadata Terms or FOAF. A non-exhaustive list of them can be found, for instance, in the Linked Open Vocabularies (LOV) application[8]. Such vocabularies consisting of class and property definitions then use definitions from other vocabularies, which correspond to reuse as-is according to our definition. The list of these reuses can be found in the LOV application by opening the given vocabulary.

Req. 1. *The tool shall support the creation and management of web vocabularies in RDFS+OWL.*

Req. 2. *The tool shall support importing existing web vocabularies and guide the user when reusing them.*

Because this use case covers a wide range of applications, such a tool must be highly versatile in order to be used in different scenarios. The bare minimum includes the following:

Req. 3. *The tool shall support customization of the generated documents and technical artifacts.*

Req. 4. *It shall be possible to extend the tool to create other technical artifacts and documents.*

Use case: Application profiles As stated in Reuse and Application Profiles (AP), an Application Profile (AP) is a specialized SDS that reuses terms from other SDSes, that is, vocabularies or APs, with some modifications and/or extensions to fit a specific domain.

Since APs are based on a vocabulary or another AP that they extend, it is necessary to maintain consistency between individual APs.

Req. 5. *The tool shall support the authoring of APs that may extend other APs or vocabularies, and shall guide users when creating them and maintain consistency between them.*

Use case: Technical interoperability The core of data interoperability is the actual data exchange; hence the goal is to enable users to easily create an SDS that describes the data structure in the form of typical documents such as a JSON Schema or XSD. In theory, sharing data in RDF that conforms to the RDFS vocabulary, possibly extended with a rich ontological description when necessary, should be sufficient. However, most software developers and data engineers are not yet comfortable with these languages and formats and require the well-known XML, JSON, or CSV (Espinoza-Arias et al. (2021); Köcher et al. (2022); Klíma et al. (2023)).

These formats have a defined structure, described by a schema (typically XSD (Thompson et al. (2012)) for XML or JSON Schema (Wright et al. (2022)) for JSON). The schema is usually sufficient to provide technical interoperability; therefore, the main challenge is to create the schema in a way *consistent with the defined semantics*, ideally deriving it from it as doing it manually is laborious and error-prone task (Nuyts et al. (2023); Westermann et al. (2025)).

Vocabularies and their application profiles define the semantics. For the purposes of this paper, we can view the semantics as a UML class diagram, with concepts as classes with their properties and relations, having cardinalities and data types. By *deriving* we mean taking this graph representation of the semantics, and transforming it into the graph or tree representation of a schema for a given data format.

Req. 6. *The tool shall support authoring technical schemas derived from semantics.*

However, in real-world applications, it is common to find multiple slightly different schemas that all describe the same underlying concept, usually differing by the context in which the data are queried. For example, an API specification for an e-commerce system may have two endpoints: `/items` and `/items/{id}`. The first one may return an array of items with only the basic properties, such as *name* and *price*. The second then would return a single item, but with all other properties as *description* or *specification*.

As another example, in some cases, it may be necessary that the structure of the schema does not directly reflect the graph representation of the semantics. For example, consider an *Address* as a concept with one of its properties *city* and a *Person* with the *Address* as a *primary address*. In JSON, the designer may decide to skip this correct, but complex relationship to *Address* and model `primary-city` as a direct property of an object representing the *Person* concept.

These two examples imply a not-so-strict relation between the structure and semantics. Designers should have the opportunity to *shape* the structure to their needs.

If this flexibility is not maintained, it may force designers to adjust the semantics to conform to the data structures. This adjustment may involve selecting concept names

and organizing associations within concepts based on the designed data structures rather than on the desired and correct semantics in the domain (such as breaking the concept *Address* from the example above).

Req. 7. *The designer shall have the freedom to customize the structure implied by the semantics without breaking the original semantics.*

In practice, we may encounter two opposing approaches regarding how a data structure corresponds to its original semantics. Considering the implicit structure derived from semantics and the corresponding data, we can:

1. Create a data structure that relaxes requirements such as cardinality and thus entirely omits certain properties. Such a data structure can represent a view that contains only the basic properties of the dataset, effectively a subset of the information.

2. Create a data structure that tightens requirements, such as restricting cardinality or adding properties. This use case is precisely addressed by application profiles. However, there may also be a need to add technical attributes that do not necessarily belong to the semantics. These might include, for example, an internal ID, or attributes for limit and offset on collections.

Req. 8. *The designer shall have the freedom to model only parts of the domain and add concepts that are not part of the semantics.*

Schemas often reference each other, for example, using `$ref` or `<import>`, enabling the incremental construction of larger, more complex schemas from smaller ones already defined.

Req. 9. *The tool shall support referencing schemas defined in other specifications.*

Thanks to well-defined semantics, it becomes possible to understand what the data actually describes (Doan et al. (2012)) and to avoid errors such as misinterpreting the meaning of a data field or drawing assumptions that, in fact, do not hold. For that, we need the mapping between the semantics and the structure. Depending on the complexity of the domain, mappings can be implicit, for example, by matching names (e.g., the *dataset-distribution* element corresponds to the *Dataset Distribution* concept), but this approach is neither formal, unambiguous, nor machine-processable. A formal mapping (in JSON-LD (Champin et al. (2020)), SAWSDL for XML (Lausen and Farrell (2007)), CSVW (Kellogg and Tennison (2015)), etc.) is needed to avoid ambiguity and ensure machines can interpret the relationship between data structure and semantics.

Req. 10. *The tool shall generate a mapping from the technical schema to the semantics.*

To reduce the technical complexity of schema creation, which is still necessary due to Requirement 7 and 8, we can employ a simpler language that is then translated into the final schema.

This approach gives us the benefit of having one language that can then be translated to different formats, such as XML and JSON, because both are hierarchical and have some common properties. As we show in the evaluation, even basic feature support for given formats is often sufficient. For constructs specific to a particular format, such as groups in XML, we can simply extend the language with those constructs in an appropriate way so that they can be ignored by other formats, such as JSON.

Req. 11. *The tool shall provide a simpler language that can express the desired data structure independently of the target format.*

General requirements

FAIR principles One of the critical issues is that current semantic data specifications are not fully machine-processable and, in general, FAIR (Wilkinson et al. (2016)). FAIR specifications (Garijo and Poveda-Villalón (2020); Poveda-Villalón et al. (2020)) are easily findable through a unique and persistent identifier, with which they are accessible, and are described using rich metadata. They should use open standards, have a license and contain clear provenance. In practice, not all specifications follow these principles. For example, DCAT is represented by the namespace URL `http://www.w3.org/ns/dcat#`, and the human-readable documentation URL `https://www.w3.org/TR/vocab-dcat-3/`. The namespace URL yields just the DCAT vocabulary. The documentation URL contains a bibliography, but no links to the other artifacts, not even the namespace URL. In DCAT-AP (Cock et al. (2025)), the European profile of DCAT, the situation is similar. The DCAT-AP namespace URL[9] contains just the supporting vocabulary file. There are no links to other artifacts.

A fully machine-readable SDS based on open standards will facilitate easy integration with other tools, adoption, validation, and reuse for the creation of additional SDSes built on it. The entire workflow for creating and maintaining an SDS should then not depend on proprietary tools and should be easily extensible to ensure maximum adoption.

Req. 12. *The specifications authored in the tool shall include machine-readable metadata that identifies and provides access to all artifacts of the specification and helps the publisher publish the specification under the FAIR principles.*

Dataspecer

Based on the motivation and requirements described in Motivation and Requirements, we introduce Dataspecer[10]. It is an open-source[11] web application for the complete creation and management of SDSes at the semantic (Klímek et al. (2024)) and technical (Stenchlák et al. (2022)) levels. Dataspecer supports the entire SDSes development lifecycle, starting with the import of existing vocabularies or specifications for reuse, continuing through conceptual modeling, including the definition of application profiles, followed by structural definitions and generation of the final specification.

The goal of Dataspecer is to provide a user interface for the easy authoring of SDSes, which can then be generated in the form shown in Figure 2. Dataspecer represents work-in-progress SDSes as *projects*. The projects consist of various

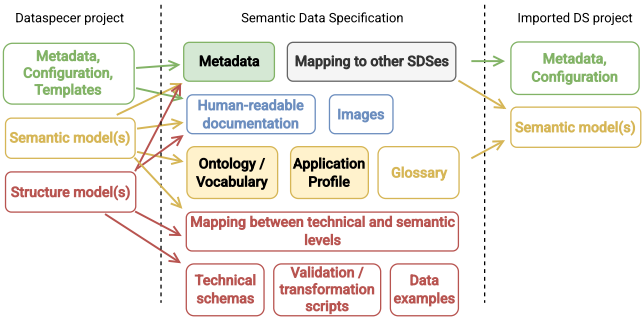

Figure 3. The relation between a Dataspecer project, a fully exported SDS and a project that imports an SDS. The arrows show import/export dependencies. The highlighted parts of the SDS are currently used for import via open standards. The rest, such as importing the structure model, is future work.

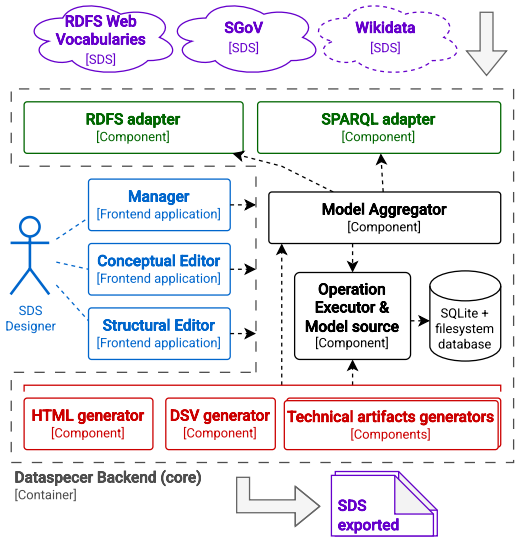

Figure 4. Architecture of the Dataspecer tool containing key parts with indication of import and export flow of SDS. Black arrows denote the relationship that the given component uses the target component.

parts, which we call *models*, and the user edits these models through Dataspecer. Based on these models, an SDS with metadata is then generated. Thanks to the metadata and technical artifacts, such as DSV or RDFS vocabulary, it is possible to upload the SDS back into Dataspecer as a project and continue editing it. We therefore consider the SDS to be an "export," where a single piece of information, for example, the name of a concept, as mentioned in Requirements, appears in multiple places, whereas within the project it is contained only in the *semantic model*, from which technical artifacts, documentation, diagrams and other parts of the SDS are generated. This is shown in Figure 3.

The high-level architecture of Dataspecer, shown in Figure 4, can be divided into the **core** (indicated in the figure by a black dashed box) and the individual **applications** (shown in blue), which allow users to manipulate the different parts of a project. The Dataspecer core is responsible for loading external SDSes into the internal format, as demonstrated in Figure 3, and for exporting SDSes. The individual applications then access the internal representation of the project, which they can modify, thereby changing the form of the resulting specification.

Dataspecer Core and Backend

At runtime, the Dataspecer core acts as the backend of the framework. Its main responsibility is managing projects and providing applications with a simplified representation of the project consisting of models while ensuring their consistency and handling the import and export of specifications. Individual applications can thus be shielded from the complexities of SDSes and focus only on one specific part.

The applications can access the individual models via API or can read the fully exported specification via DSV or other technical artifacts that are being exported. The latter approach allows applications to be at least partially independent of Dataspecer. However, due to the proposed workflow, they cannot directly modify the artifacts.

To ensure consistency and to enable change tracking in the future, modifications via API are carried out through a set of operations, which are evaluated by the Operation Executor and applied to the model. Since models depend on each other and resolving dependencies could be problematic for the individual applications, the Model Aggregator can provide a simplified representation that applications can work with.

Dataspecer supports extensibility through adapters, which facilitate the import of existing conceptual and semantic models. These adapters allow Dataspecer to integrate various external data specifications or vocabularies, such as RDFS vocabularies or data accessible via SPARQL endpoints (Clark et al. (2013)), including, for example, the Czech Semantic Government Vocabulary (SGoV) (Křemen and Nečaský (2019)). We have also experimented with adapters for Enterprise Architect and Wikidata (Gora (2025)), and with integration with our neighboring research group's tool for multi-model database management (Klímek et al. (2023)), further proving the flexibility of the architecture.

The SDS can be previewed and exported as HTML documentation using Respec[12], accompanied by all related technical artifacts (schemas, mappings, examples, etc.). The Respec document serves as an entry point for human users and can also be accessed by machines as it links to all necessary resources and metadata. Support for another documentation system such as Bikeshed[13] can be added by implementing a suitable documentation generator.

To generate artifacts in SDS, individual generators can be registered in the backend as separate components. Based on given model(s) and configuration, these generators produce the files that form the SDS. We have already implemented generators for XML, JSON, and CSV schemas, XSLT transformations, JSON-LD contexts, JSON example, etc.

For example, JSON generators read structure models (described later) to generate JSON Schema and the aggregated result of all conceptual models to correctly generate JSON-LD context that can be used by linked data consumers for semantic interoperability. In addition, the generators can inject text into the documentation to describe the generated artifacts.

Manager Application

Manager is a simple front-end application that is tightly coupled to the backend and provides a UI for managing projects, importing and exporting SDSes, and serves as a

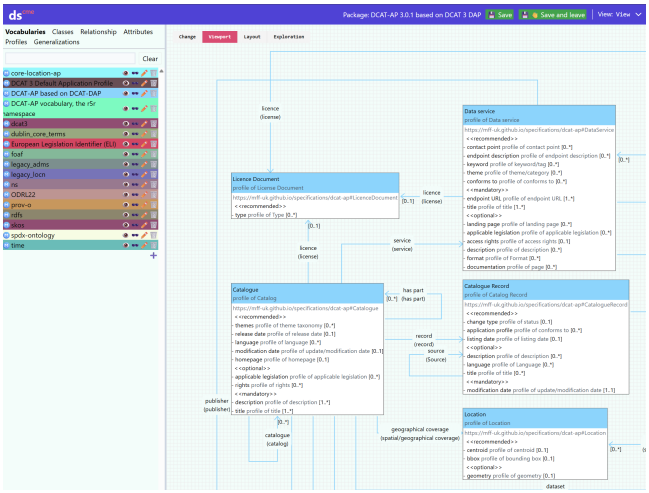

Figure 5. DCAT-AP in Conceptual Editor of Dataspecer - to the left, there are vocabularies and APs reused by DCAT-AP, to the right the current AP displayed on canvas

signpost to other applications. It also manages metadata, including a Handlebars[14] template that is used to generate an HTML document for SDSes. This gives users the flexibility to customize the SDS front page to their needs.

For easier use, it provides a wizard that helps users create their own vocabulary, application profile, or technical schema from existing specifications.

Conceptual Editor Application

The conceptual editor is an application capable of creating and editing *semantic models* - vocabularies, and application profiles. It can work with multiple models simultaneously (Requirement 2), allowing the designer to import existing specifications and build alongside them or interconnect them, forming a hierarchy like the one in Figure 12. The dependency between models is maintained by the backend in a hierarchical structure, ensuring consistency and enabling the display of complex class-profiling scenarios across multiple layers of application profiles.

With the use of the Conceptual Editor (and Backend with the Manager), a user can import existing web vocabularies, which are displayed on the canvas, and create a new vocabulary alongside them by defining new concepts and linking them to the existing ones. In this case, the result of the process is an SDS with RDFS+OWL web vocabulary accompanied by documentation that describes all the concepts and reused vocabularies.

In the case of creating an application profile, the process is similar. The user imports existing vocabularies or creates new ones and then reuses individual classes, attributes, and properties, defining their profiles. The generated documentation includes only the profiled concepts, along with information on how they are applied in a specific application domain, and a technical description of the application profile.

Currently, we only support basic modeling constructs: classes, attributes, associations, specializations, and profiles. However, we plan to extend this in the future with more complex OWL constructs such as restrictions, disjointness, etc.

The tool supports various layout options, categorizing concepts into roles, multiple graphical views of a single specification, and various visual adjustments such as hiding features, grouping concepts into a single node, or duplicating them. This allows the designer to graphically express the complexity of the specification in a simple way. These views are then included in the documentation.

Structural Editor Application

To support the requirements regarding technical interoperability, we provide the Structural Editor. The goal of this application, in combination with the Dataspecer core, is to enable users to create a data structure from their semantics and generate various technical schemas and related artifacts from it.

Our main focus is on JSON and XML formats, as these are our main use cases, which are further explained in Case Study: Czech Formal Open Standards (FOSes). Due to both formats sharing a similar hierarchical structure and to support Requirement 11, we designed a simple graphical language for the structural model that corresponds to nested lists (see Figure 6).

The structural language consists of three main constructs: objects, attributes, and associations. Objects represent classes from the conceptual model and are translated as complex types in XSD or as objects in JSON Schema. Associations represent relationships or a chain of relationships between classes and are translated as elements in XSD or as nested objects in JSON Schema. The ability to represent the chain of relationships allows us to skip complex structures at the conceptual level and make the data structure flatter, as shown in an example with an *Address* in Requirement 6. Attributes then represent primitive values in the context of the given serialization format. For XML, users can decide whether the attribute should be translated to an XML element or an XML attribute. The language also supports specifying alternatives between multiple objects, references to other structures, and format-specific extensions, such as wrapping multiple XML elements of the same name, e.g., a set of `<item>` elements representing an array, in a container element `<items>`. The detailed description of the language is out of the scope of this paper and will be described in a separate publication.

We chose this approach of a richer structural language for several reasons: (i) our main use case is support for JSON and XML, which share a similar hierarchical structure. (ii) some of our users are familiar with these formats and thus need a language that is close to them, and (iii) the structural changes between the conceptual model and the technical schema are often small, which is sufficiently supported by the proposed language.

Designers can use the existing semantics to build the structure (Requirement 6) simply by searching for the correct term or clicking on the plus button, but are not restricted by it. Elements can have their order changed by dragging, and properties can be moved under parents or children as long as the rules that ensure that the structure remains mappable to the original conceptual model are not violated. In this way, the desired data structure can be easily reached with just mouse clicks (Requirement 7, 8).

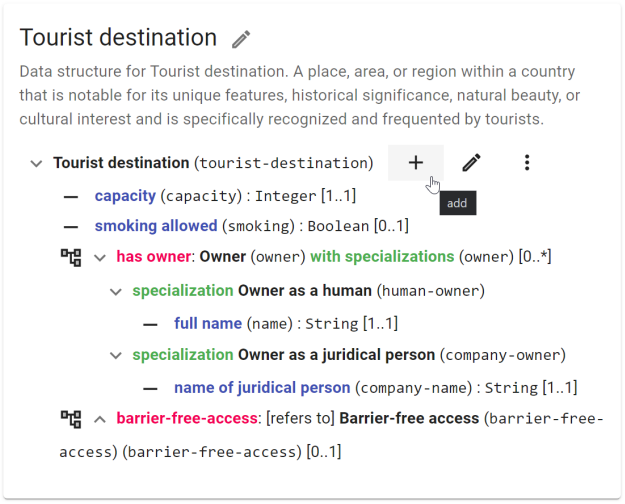

Figure 6. Example of structure for cataloging Tourist destinations. Primitive properties are in **blue** with minus symbol, associations in **red** and objects in **black** with a tree symbol. You can see that *Owner* object can be specialized if necessary, this is denoted by **green**, and Barrier-free access is a structure defined in another SDS.

We can then translate this graphical language into JSON Schema and XSD. We are also experimenting with generating other formats, such as CSVW by mapping individual objects to tables and attributes and associations into columns*. For seamless integration with linked data, Dataspecer can generate transformation scripts or mappings that can be used to convert data from the technical format into RDF (lifting) and vice versa (lowering). Currently, we support JSON-LD contexts for JSON, which can be used for lifting from JSON to RDF, and XSLT+SAWSDL for bi-directional XML transformations (Requirement 10). These technologies are supported by their respective communities, and they have proven to be sufficient for our use cases. The support for other mapping languages, such as RML[15], is planned as future work.

In Dataspecer, if one creates multiple SDSes from the same semantics, it is then possible to reference one structure from another (Requirement 9). In this context, by reference we mean a simple link to the corresponding technical artifacts of other SDSes. This allows data modelers to first create small technical SDSes and then use them to build larger ones.

Support for FAIR Principles

Dataspecer support (Requirement 12) for creating FAIR SDSes (Poveda-Villalón et al. (2020)) is only partially implemented, and the rest is our future work. In most cases, being FAIR means properly registering the SDS in the corresponding catalogs and providing the necessary metadata by the user, which might be considered out of the scope of the tool. However, the tool should guide the user on what to do. It is important to note that Dataspecer does not aim to serve as an SDS catalog, but only as a tool for editing specifications. Below is the current state of the implementation and future work:

Findable: Each specification, as well as individual artifacts, can have a persistent identifier (e.g., via `w3id.org`) with basic machine-readable metadata using standard vocabularies such as Dublin Core and DSV. This metadata can be set directly in the tool. Other recommended metadata, such as license, creators, and versions, must currently be filled out manually without any support. We also plan to provide instructions for registering the resulting vocabulary in Linked Open Vocabularies (LOV) and `prefix.cc`, making the specifications discoverable through established catalogs.

Accessible: Dataspecer generates a ZIP file containing the specification that can be easily deployed to any web server, such as Apache, Nginx, or GitHub Pages. The specification includes DSV metadata to find various files from the specification. Part of our future work is generating web server configurations for proper content negotiation, such as `.htaccess` files, files for w3id redirection, and other instructions for deployment.

Interoperable: Interoperability is achieved by utilizing a machine-processable representation of the knowledge contained in the data specifications, in open data formats (RDF Turtle, JSON-LD), using well-defined vocabularies (DSV, RDFS, OWL). All information about the reuse of existing vocabularies and the profiling of terms is also represented in a machine-readable form.

Reusable: Dataspecer generates rich, human- and machine-readable documentation, including diagrams, definitions of concepts, usage notes, and links to reused terms. Designers are encouraged to enrich the documentation with their own examples and other relevant information. Specifications use well-defined vocabularies, as mentioned above. Dataspecer-created specifications with DSV metadata can be reused as a base for additional application profiles, e.g., using Dataspecer itself. However, as mentioned above, proper support for specifying licenses and other metadata is still under development. Since reusable specifications should also capture provenance and the rationale for including individual concepts, part of our future work is to automatically track changes made to the SDS and publish them, thereby helping consumers better understand and use the specification.

Dataspecer Development

Dataspecer is actively being developed on GitHub† in collaboration with the Faculty of Mathematics and Physics of Charles University and the Digital Information Agency of the Czech Republic [16] and is publicly available as open source software (`https://github.com/dataspecer/dataspecer`). Due to its extensibility, we involve our computer science students in the development process. Usually, students provide their own frontends, model specifications, and generators for creating specific artifacts, such as the OpenAPI generator (Akhvlediani (2024)), the LLM assistant (Prokop et al. (2025)), or a mockup application generator. At the time of writing, we

*The details of the transformation is out of scope of this paper.

†At the time of writing, we have 12 contributors, of whom 2 are active, along with 4 other students working on their own projects within Dataspecer. So far, the project has received 33 stars and has more than 3,600 commits.

have two active developers and 4 students working on extensions. All issues are tracked on GitHub in English.

Dataspecer is implemented in TypeScript and most of the code is part of the main repository as a monorepo. Frontends use Vite + React, and the backend runs on Bun[17] + Express.

For simplicity, we publish all core functionality as a minimalist single-container Docker image which uses a bundled SQLite database, making it easily deployable with the docker command `docker run -p3000:80 ghcr.io/dataspecer/ws` and accessing `http://localhost:3000`. Detailed instructions are provided in the repository README.

Related Work

The OSLO (Open Standards for Linked Organizations, Buyle et al. (2016)) and SEMIC[18] (Semantic Interoperability Community) toolchains are similar, well-established approaches for creating SDSes, including application profiles, in complex ecosystems. OSLO, initiated by the Flemish Government in Belgium, aims to standardize data exchange within local administrations, while SEMIC, under the Interoperable Europe programme, builds on top of OSLO and targets cross-border data interoperability within the EU.

These toolchains begin with Enterprise Architect[19], which is used for vocabulary management in UML following their strict set of rules[20] to automatically generate *data specifications* - documents targeting domain experts, developers, and machines. These data specifications may include RDFS representations of the vocabulary, SHACL shapes, HTML documentation, and diagrams. The SHACL shapes are effectively used to validate the correctness against the AP, but do not provide machine-readable semantic information about how individual APs alter the original definition. These toolchains are implemented as a series of CI/CD pipelines to read the source Enterprise Architect file, generate individual artifacts, and, depending on the context, publish them.

From a functionality perspective, they meet our requirements, covering use cases for vocabulary and application profile authoring. However, we identify several ideological drawbacks that limit the usability of these toolchains:

1. The toolchains are **complex to set up** due to Git and CI/CD, as they mostly target large organizations. They depend on Enterprise Architect, which is paid and **proprietary software**.

2. The toolchains are **difficult to operate** because (a) the Enterprise Architect editor is not suited to their specific needs, and (b) the configuration is spread across various pipeline and git configuration files.

3. The toolchains do not provide any suggestions when modeling and do not check consistency and validity immediately, making them **difficult to develop and iterate SDSes quickly** (Requirement 5).

Furthermore, as both toolchains focus primarily on semantic interoperability, they lack support for technical schemas, such as the ability to design structures that meet developers' needs (Requirement 7 and 8).

On the other hand, we see a significant benefit in Git/GitHub/GitLab integration, as features such as collaboration, user management, versioning, and change proposition are already implemented and widely understood by users. However, the integration setup should not restrict smaller developers and open-source communities.

An alternative is the Finnish tool **Tietomallit**[21], an all-in-one web application for semantic modeling and publication of specifications. Users can create conceptual models directly in the web interface and collaborate with others. They distinguish between core vocabularies and their profiles that reuse terms from them. As Tietomallit is also a publication platform, SDSes are not generated per se but are directly made public. However, users can generate individual artifacts, including OWL, JSON-LD context, or JSON schema. The generation of JSON Schemas has minimal customization as it is simply derived from the conceptual model. Thus, it is expected that the data in JSON matches the implicit conceptual model derived from semantics, which conflicts with Requirement 7 and 8. The tool itself is currently not reusable outside the Finnish use case because of dependencies on the Finnish environment.

Focusing solely on individual use cases introduced in Motivation and Requirements, to our knowledge, there are no other tools that focus on AP creation as this problem; although important, it is relevant only for larger communities.

Protégé (Musen (2015)) is the de facto standard tool for the creation of RDF vocabularies and OWL ontologies. The desktop client provides mature OWL editing with plugin support for visualizations, reasoning, and documentation generation, effectively supporting our first use case with established Semantic Web practices. However, its functionality is limited to OWL capabilities; therefore, it lacks support for application profiles. Although a web version (WebProtégé) exists, it offers very limited functionality without basic features such as visualization or generation customization.

The Linked Data Modeling Language **LinkML** (Moxon et al. (2021)) is a modeling language and a tool to generate various artifacts from a YAML document containing easy-to-understand semantic descriptions of concepts, their relations, and other metadata. It supports various technical formats and can also produce transformations between them. However, its target audience is developers who need to generate various artifacts quickly based on a conceptual model of their data. The tool also binds the resulting schema structure to the conceptual model that the user must design, forcing the user to contemplate the specific schema rather than the concepts that the schema describes. The schema may then need to be manually mapped onto an ontology. Thus, it does not use the existing ontology model as a basis. Given the wide range of supported technologies and their use, we plan to implement a LinkML model generator into Dataspecer so that the resulting structure is also instantly usable in these tools.

All mentioned tools are compared in Table 1.

The term *profile* is used in a wide range of contexts, such as in healthcare's HL7 FHIR (FHIR Infrastructure Work Group (2023)); however, its core idea remains consistent. Profiles define how to use a combination of base standards

Table 1. Comparison of Dataspecer with tools mentioned in Related Work with respect to fulfillment of requirements defined in Motivation and Requirements. ◆ indicates only partial fulfillment of the requirement while ✓ complete fulfillment.

Dataspecer	OSLO/SEMIC toolchains	Tietomallit	Protégé	LinkML
Requirement 1 - creation of web vocabularies in RDFS+OWL				
✓	✓	✓	✓ richer OWL support	✓
Requirement 2 - reuse of existing vocabularies on the web with user guidance				
✓	✗ only by manual IRI input	✓	✓	✗
Requirement 3 - customization of generated documents and technical artifacts				
✓	✓	✗	✗	✗
Requirement 4 - possibility to extend the tool to generate additional types of artifacts				
✓ by coding	✓ by scripting CI pipelines	✓ by coding	✓ by plugins	✓ by coding
Requirement 5 - creation of APs with reuse of existing APs with user guidance				
✓	◆ no user guidance	✓	✗	✗
Requirement 6 - generate technical schemas such as XSD, JSON Schema, ...				
✓	✗	◆ JSON and OpenAPI only	✗	✓
Requirement 7 - freedom to derive different structure than the implicit from the semantics				
✓	✗	✗	✗	✗
Requirement 8 - freedom to model only part of the structure or add additional concepts				
✓	✗	✗	✗	✗
Requirement 9 - support for referencing other schemas				
✓	✗	✗	✗	✗
Requirement 10 - generate a mapping between the structure and the semantics				
✓	✗	◆ JSON-LD only	✗	✓
Requirement 11 - simpler, user-friendly language to create the structure				
✓	✗	✗	✗	✓
Requirement 12 - contains machine readable metadata and supports publishing FAIR specifications				
◆ *not all FAIR principles*	✗	✗	✗	✗

for a given environment (International Organization for Standardization and International Electrotechnical Commission (1998)). The closest use to our context is the use of UML profiles (OMG (2015)), more specifically, UML Class model profiles in software engineering. It is a mechanism for customizing the UML language for a specific domain while still being UML compliant. This matches our notion of a profile as a way to customize a base standard for a specific use case while still being compliant with the base standard.

It is important to note that Application Profiles and UML Class profiles work on different meta-levels. UML profiles operate at the metamodel level. This allows developers to create a domain-specific modeling language that remains technically valid within any standard UML tool. Application Profiles operate at the model level, which allows designers to create domain-specific specifications.

An example of a UML profile is OntoUML (Guizzardi (2005)), a language for conceptual modeling based on UFO (Guizzardi (2005)) It defines profiles such as a *role* or a *phase*, which are stereotypes that can be applied to UML classes. In ontology development, we can use these stereotypes to enrich the semantics of our concepts. For example, the class *Student* is clearly a subclass of the class *Person*. Since the subclass *Student* meets UFO's criteria of being a role, we can say it is a *role of the Person class* instead of a *subclass of the Person class*. From this point of view, the *Class Profile* can be viewed as a stereotype of a *UML Class*[7]. Application profiles then operate similarly, but at a lower meta-level. For example, in the context of the application profile DCAT-AP for European data catalogs, there is a class profile of the class Dataset. This means that its instances will still be instances of the original class Dataset, but with the knowledge of DCAT-AP, they can be interpreted more precisely, as Datasets in the context of European data catalogs.

Evaluation: Improvement of DCAT Application Profiles

In this section, we take a closer look at DCAT and its application profiles, examining in detail the challenges they face and how Dataspecer would address them if DCAT and its profiles were created in it. This thus demonstrates that (i) we have successfully met all the requirements and (ii) the proposed solution is capable of addressing various real-world use cases. We cover all three use cases from Motivation and Requirements as the DCAT hierarchy (Figure 1) goes from the most generic vocabulary to the specifications concerning technical interoperability in JSON and XML.

DCAT - Data Catalog Vocabulary

Formally, DCAT is not just a vocabulary; it is also a Default Application Profile (DAP). It is a vocabulary because it defines new domain-specific terms in the `http://www.w3.org/ns/dcat#` namespace. It is an application profile because it reuses terms from existing vocabularies such as DCMI Metadata Terms, SKOS (Miles and Bechhofer (2009)), FOAF, PROV-O (Lebo et al. (2013)), the Time Ontology (Cox and Little (2022)), etc., to be used in the data catalogs context.

Explicit term reuse The first identified issue is the lack of a machine-processable representation of which terms are actually being reused in an AP. In the DCAT Vocabulary, classes like `dcat:Resource` or `dcat:Dataset` and properties like `dcat:keyword` are defined. Moreover, DCAT specifies the reuse of classes like `skos:Concept` or `dct:PeriodOfTime`, and properties like `time:hasBeginning` or `dct:title`. The fact that these terms are reused in DCAT is represented only by mentioning them in the `dcat-external.ttl`[22] file linked from Section 6.1 of the specification. Moreover, processing this file along with other vocabularies adds DCAT-specific annotations directly to the external terms like

Title

Vocabulary:	DCMI Metadata Terms
URI	http://purl.org/dc/terms/title
Label	Title
Definition	A name given to the resource.
Type of Term	Property
Has Range	http://www.w3.org/2000/01/rdf-schema#Literal
Subproperty of	Title (http://purl.org/dc/elements/1.1/title)

Figure 7. `title` property in DCMI Metadata Terms

`dct:title`, which creates confusion in other use cases of these vocabularies.

§ 6.5.1 Property: title

RDF Property:	dcterms:title
Definition:	A name given to the record.
Range:	rdfs:Literal

Figure 8. Catalog Record title in DCAT 3

Terminological changes to profiles Another issue in DCAT is the lack of a machine-readable representation of context-specific adjustments to the reused classes and properties from other vocabularies. As one representative example, we take the very commonly reused property `dct:title`. In its original vocabulary, DCMI Metadata Terms, it is defined as can be seen in Figure 7, with the label `Title` and the definition `A name given to the resource.`

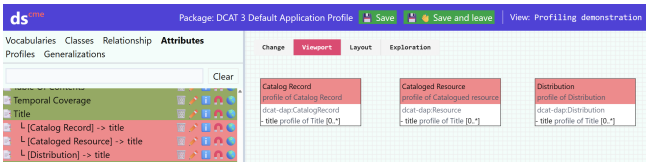

Figure 9. Various titles in DCAT 3 DAP in Dataspecer. All titles are profiles of a single `dct:title`, but in different contexts - in this case in different classes. For each class, Dataspecer allows the definition of title to be altered to better suit the given context.

In DCAT, the `dct:title` property is reused in multiple contexts. First, for titles of Cataloged Resources, with an unchanged definition, but a slightly modified label `title` - note the lowercase `t`. Second, it is used for the titles of Catalog Records, with the modified label, but also with a modified definition, as can be seen in Figure 8. Third, it is used for the titles of Distribution, with the modified label and yet another definition.

Here, we can already start to see the issue - one property, `dct:title`, has a different label when reused in DCAT than it has in its original vocabulary. Moreover, it also has different, and even multiple, definitions in DCAT, based on

§ 2.4.5 title

Profiled relationship IRI(s)	dcterms:title
IRI	https://mff-uk.github.io/specifications/dcat-dap/#CatalogRecord.title
Label	title
Definition	A name given to the record.
Domain	Catalog Record
Range	Literal [0..*]
Profiles	property Title (dcterms:title) Definition: *A name given to the resource.*

Figure 10. Catalog Record title in DCAT 3 DAP from Dataspecer

the class with instances of which it is used. However, this information is not represented in any machine-processable form and may even confuse the users of the specification, such as developers trying to build a UI for an application processing DCAT.

Dataspecer's approach Dataspecer natively supports term profiling in its Conceptual Model Editor (Figure 9), transparently indicates this information in the generated specification (Figure 10), and represents this information in a machine-processable form in an attachment to the specification, using the Data Specification Vocabulary (DSV) (Figure 11). This is shown in our Dataspecer-created version of DCAT 3, the *DCAT 3 Default Application Profile (DCAT-DAP)*[23]. DCAT-DAP consists of 34 class profiles, 89 property profiles, and, e.g., 60 terminological changes in the labels of terms from the reused vocabularies, including the DCAT Vocabulary itself. 63 term labels were reused as-is. Representing the term reuse in a machine-readable way then allows the editor to more easily check these inconsistencies and confirm that they are intentional or fix the erroneous ones.

Dataspecer also uses the Profiles Vocabulary (Car (2019)) to identify and describe all the specification artifacts so that they can be discovered. The necessary metadata is stored embedded in the specification document, allowing applications to automatically discover all relevant artifacts of the specification, making the specification more FAIR. The Profiles vocabulary represents the specification as an instance of `prof:Profile`, and the artifacts, including the human-readable specification, are instances of `prof:ResourceDescriptor`, connected using `prof:hasResource`. The description of each artifact then includes its format, the standard used, and its role. The profiles then point to other profiles they reuse.

Requirement 1 and 2 were fulfilled by creating the SDS of DCAT. It reused several vocabularies such as `skos` or `dcterms`. Requirement 3 allowed us to customize the final document to fit the format of DCAT specifications, such as customizing the ordering and contents of chapters or customizing logos.

DCAT-AP for European Data Catalogs

Continuing the case from DCAT - Data Catalog Vocabulary, we now focus on DCAT-AP, the DCAT application profile for data portals in Europe, published by the European Commission and maintained by the SEMIC initiative, and

```
<#CatalogRecord.title>
  a dsv:TermProfile,
   ↪  dsv:DatatypePropertyProfile;
  dsv:domain :CatalogRecord;
  dct:isPartOf dcat-dap:;
  skos:prefLabel "title"@en;
  skos:definition "A name given to the
   ↪  record."@en;
  dsv:cardinality cardinality:0n;
  dsv:property dct:title;
  dsv:datatypePropertyRange rdfs:Literal.
```

Figure 11. Catalog Record title in DCAT 3 DAP DSV representation

also the most prominent application profile of DCAT. Naturally, it profiles DCAT, namely its default application profile, but also adds its own supporting vocabulary[24] for new terms such as *distribution availability* and *applicable legislation*. Again, we show that the current official representation is insufficient, mainly from the machine-processability point of view, and how it can be represented better using Dataspecer[25]. The discoverability of specification artifacts is handled by the Profiles Vocabulary metadata, but there are additional issues with the specification contents and relation to the reused specifications.

Explicit term reuse In DCAT-AP, the list of reused terms can be extracted from the specification document by processing it as RDFa (Adida et al. (2015)). However, this yields the context-specific definition and the usage note as direct properties of the reused terms, leading to the same confusion as above with DCAT. The reuse information is then reflected in the SHACL shapes used for validation of DCAT-AP compliant data. However, SHACL shapes are generated from the AP definition, and they are not something to be used as a definition itself. An example of a limitation of relying on SHACL for AP definition is a situation where we have one class used in two contexts in an AP. In DCAT-AP, this may be a Distribution representing a file download, which has a different set of mandatory properties than a Distribution representing a Data Service. Both are reusing the same class `dcat:Distribution`, and therefore, all SHACL shapes targeting this class will apply to all instances of the class, regardless of the different usage contexts. This then forces the specification editors to change their AP due to a limitation in a data validation language, which is not systematic. Moreover, there is no support for term profile hierarchies. There is no machine-processable indication that, for example, the reuse of `dct:title` on DCAT-AP Catalogue Record[26] is actually a reuse of how that property is reused for the Catalog Record in DCAT[27], not a reuse of how it was originally defined by DCMI Metadata Terms[28].

Terminological changes to terms In DCAT-AP, the same property is reused in multiple contexts, e.g., for Dataset, Data Service, Catalog, Distribution, and Category Scheme, having a more precise label and definition for each context. In DCAT-AP 2.1.1, this was represented using RDFS directly on the original property IRI, causing confusion, as even with just DCAT-AP processed, a developer gets multiple labels for one property and no means of distinguishing the contexts. In DCAT-AP 3.0.0, the different contexts are manifested in the generated SHACL shapes, however, a machine-readable AP definition is missing.

Even in cases where no AP-specific changes are defined, in cases where one term reuses multiple terms at once, it is necessary to explicitly represent from which term the terminology should be reused. Otherwise, it would be unclear which label and definition should be reused.

Context-specific Property Domains and Ranges When reusing properties, it is common for an AP that the property has an AP-specific domain and range. For example, `dct:title` is used for titles of Cataloged resources and Catalog records in DCAT, yet it does not have a domain where it is defined. Again, there is no machine-readable representation of this fact. In DCAT-AP, this information manifests itself only in the generated SHACL shapes. Regarding AP-specific ranges, for example, DCAT-AP restricts the range of `adms:sample` to `dcat:Distribution`, while originally, it is defined as `adms:Asset`. Dataspecer allows specification editors to explicitly represent each property reuse context along with the domain and range specific to the context.

Cardinality Constraints A part of property reuse in an AP are cardinality constraints. For example, in DCAT-AP, the reuse of `dct:license` on Distribution is restricted to `0..1`, meaning that there may be at most one license for a Distribution. Moreover, in DCAT-AP-CZ, a Czech profile of DCAT-AP, it is further restricted to `1..1`, meaning that there must be one license for each distribution.

Dataspecer's approach All the above mentioned issues are addressed by DSV and are fully supported in Dataspecer. A separate term profile is created for each usage context, each carrying its own label, definition, and usage note, if changed, or may restrict cardinality, or domain and range. Profiles can then profile other profiles, creating a term profile hierarchy. Since all profiles are considered separate entities, they do not unintentionally interfere with each other, which avoids the mentioned limitations of SHACL. However, for simple cases where collisions would not occur, it is possible to generate the SHACL shapes as well.

DCAT-AP consists of 34 class profiles, 131 property profiles, and, e.g., 42 terminological changes in the labels of terms of the reused vocabularies and application profiles. 123 term labels are reused as-is.

DCAT-AP-CZ

The Czech Digital and Information Agency also uses Dataspecer for the definition of DCAT-AP-CZ[29], the Czech application profile of DCAT-AP, and subsequent profiles of DCAT-AP-CZ for specific purposes. For example, in all these profiles, the class `dcat:Dataset` is reused, but each time in a different context, and with changed definitions along the way. At the same time, users are implicitly expected to comply with all restrictions from the entire application profile hierarchy, i.e., to stay compliant with what DCAT says, what DCAT-AP says, and what DCAT-AP-CZ says. These links are present in various forms in the human-readable specifications. However, it is quite demanding for the user of the specification to manually navigate the whole hierarchy for each class and each property reused in the

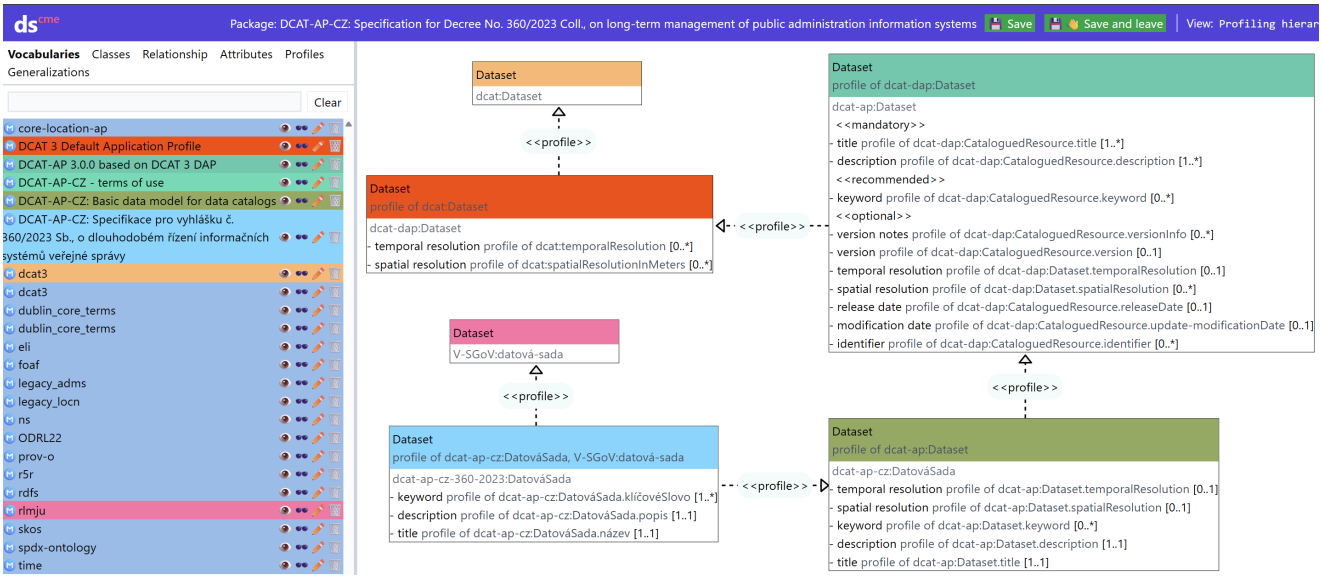

Figure 12. 5-level term profile hierarchy for `dcat:Dataset` example in Dataspecer - the relations represent class profiling

specification, and, without Dataspecer, there is no machine-processable way of helping the users. When implemented properly, the term reuse hierarchy can look like in Figure 12. Although it may seem complex, it just sheds light on existing relations among the data specifications and the terms reused in them. Note that the relations between the classes are the *term profiling* relations. At the top we have the **orange DCAT Vocabulary Dataset**, profiled by the **red DCAT-DAP Dataset**, profiled by the **teal DCAT-AP Dataset**. Next is the **green DCAT-AP-CZ Dataset**. Finally, there is the **blue Dataset** in one of the DCAT-AP-CZ profiles. Note that the blue Dataset also profiles the **pink Dataset**, a DCAT-independent definition of a Dataset in SGoV.

Profiling of Multiple Terms There are situations like the one described in the previous case in which a term in an AP reuses multiple terms at once. Another example can be found in GeoDCAT-AP[30] Section A.7.16, where it is suggested that a *responsible party* be double typed as `prov:Agent` and `foaf:Agent`. The reason for this may be that the AP editors want to make sure that the compliant data is usable to some extent by users who know only some of the reused terms. The situation is similar to defining an `owl:unionOf` class, but not defining an IRI for it, and requiring the data instances to be explicitly typed by all the reused classes. This situation is not systematically covered in current specifications, not even in human-readable documents. Dataspecer supports the representation of profiling of multiple terms and reflects it in the generated artifacts, including human-readable documentation (Figure 13).

DCAT-AP-CZ consists of 28 class profiles, 71 property profiles, and, e.g., 73 terminological changes in the labels of terms of the reused vocabularies and application profiles. They are, however, mostly caused by the addition of Czech translations of terms not already available in Czech from the reused vocabularies and application profiles. 26 term labels are reused as-is.

§ 2.7 Dataset

Profiled class IRI(s)	dcat:Dataset https://slovník.gov.cz/veřejný-sektor/pojem/datová-sada
IRI	https://ofn.gov.cz/dcat-ap-cz-360-2023/draft/#DatováSada
Label	Dataset
Definition	A conceptual entity that represents the information published.
Profiles	class profile Dataset (https://ofn.gov.cz/dcat-ap-cz#DatováSada) class profile Dataset (https://mff-uk.github.io/specifications/dcat-ap#Dataset) Definition: *A conceptual entity that represents the information published.* class profile Dataset (https://mff-uk.github.io/specifications/dcat-dap#Dataset) class Dataset (dcat:Dataset) Definition: *A collection of data, published or curated by a single source, and available for access or download in one or more representations.* class Dataset (https://slovník.gov.cz/veřejný-sektor/pojem/datová-sada) Definition: *Dataset is a set of related data.*

Figure 13. Dataset in DCAT-AP-CZ-360, a class profile of two distinct Dataset classes, merging 6 reuse contexts. (1) Dataset in DCAT-AP-CZ, profile of Dataset in DCAT-AP, profile of Dataset in DCAT-DAP, profile of Dataset in DCAT vocabulary, and (2) Dataset in SGoV

Evaluation: Other Specifications Created in Dataspecer

In this section, we will briefly report on various other specifications created in Dataspecer that were successfully deployed.

Case Study: Czech Formal Open Standards (FOSes)

In this subsection, we focus on technical interoperability. The specifications described here are developed using Dataspecer at the Czech Digital and Information Agency (DIA), which also maintains the Czech Government Semantic Vocabulary (SGoV). The specifications are then published as Formal Open Standards (FOSes), which are mandatory to use for the publication of open data by public administration in Czechia. The goal of these specifications is to provide technical artifacts such as XML Schema, JSON Schema, and transformations to RDF, so that data providers unaware

of semantic web technologies can focus on compliance with the technical schemas in languages they understand, while maintaining the interoperability of their data as RDF. This is achieved by deriving a technical data structure from SGoV, maintaining the mapping of the technical constructs to the semantic constructs, and automatically translating the data structure to multiple target technical artifacts at once (Requirement 11). For XML, these are an XSD annotated with SAWSDL (Lausen and Farrell (2007)), a lifting XSLT transformation producing the compliant data in RDF, and a lowering transformation consisting of a SPARQL query selecting the relevant RDF subset, and an XSLT transformation to transform the RDF/XML SPARQL query result to a form compliant with the XSD. For JSON, the artifacts include a JSON Schema for validation, and a JSON-LD context for mapping the data to RDF.

Examples of Dataspecer generated specifications targeting XML include *Information system record replacing an officially certified signature*[31], and *Common parts for electronic documents*[32] reused (Requirement 9) in specifications *Documents from forms for electronic submission under the Digital Services Act*[33], *Digital act certificate*[34], and *Scheme for a machine-readable layer of static text components and static combined text and image components pursuant to §69a of Act No. 499/2004 Coll.*[35] - all available in Czech.

Examples of Dataspecer generated specifications targeting JSON include updated versions of specifications describing the *Czech Register of rights and obligations*[36], technical parts of a specification for the representation of semantic vocabularies by public administration[37], and the specification for the lists of *Voluntary associations of municipalities*[38].

Some of these specifications, such as *Czech Register of rights and obligations*, already existed before Dataspecer was developed, so we recreated them one-to-one in our tool. Even though some structures were complex, we did not find any issues recreating them in the structural editor, meaning resulting JSON Schemas matched those designed by hand.

Since the SGoV domain is large with complex business processes that depend on various actors, when an error happens, such as a bug in the conceptual model, the whole process may jam. For such cases, Dataspecer allows working in a mode where consistency can be violated. This feature enables faster deployment of SDSes, with the technical debt in the form of inconsistent mappings to be fixed in the future.

To date, we have not encountered any other issues in collaboration with the authors of FOSes that would indicate any flaws in our proposed architecture and methodology.

Dogfooding - Data Specification Vocabulary

Data specifications produced by Dataspecer are represented using the Data Specification Vocabulary (DSV) (Klímek et al. (2025b)), and, more specifically, its Default Application Profile (DSV-DAP) (Klímek et al. (2025a)). In the spirit of dogfooding, those specifications themselves are developed in Dataspecer. This means that it is clear which terms of which vocabularies are and are not used by the profile, what the cardinalities of the reused properties are in which contexts, what the main and supportive classes are, and all this information is published as machine-readable, enabling further reuse and profiling.

Impact in Other Areas

Dataspecer is currently being used to build an application profile ecosystem in the domain of Czech research data repositories, part of the EOSC-CZ initiative[39]. Contrary to the DCAT-AP-based ecosystem of European public administrations, where standardized metadata flows from local data catalogs to regional catalogs, national catalogs, and, finally, to the Official portal for European Data[40], there is no such metadata harvesting hierarchy established in the international EOSC community. However, a national hierarchy is being established in Czechia. The Czech Core Metadata Model (CCMM) profiles DCAT-AP and Datacite, identifying core metadata for the domain of research data in general[41]. It is built with profiling in mind, so that domain-specific data repositories can define their CCMM application profiles, ensuring semantic-level data interoperability. On the technical level, the XSDs with SAWSDL are being developed. Dataspecer is the tool of choice in the project.

Evaluation: Productivity and Usability

In this section, we evaluate the productivity gains of using Dataspecer for designing Semantic Data Specifications (SDSes) and we assess the system's usability using the System Usability Scale (SUS; Brooke et al. (1996)).

Dataspecer supports three main use-cases: (i) vocabulary, (ii) Application Profile (AP), and (iii) technical schema creation. In this evaluation, we chose to opt out of evaluating the first use-case in isolation, as it is not the main focus of our tool to replace existing vocabulary editors, and there are already mature tools available for this purpose. Regarding the definition of an application profile, we compare the workflow in Dataspecer to a workflow in the SEMIC toolchain and to the manual approach. Next, we compare the time required to create technical schemas in Dataspecer with the time required to create them manually.

Application Profile Definition Process Comparison

In Evaluation: Improvement of DCAT Application Profiles, we characterized our use cases DCAT, DCAT-AP and DCAT-AP-CZ in terms of the number of class profiles, property profiles, and terminological changes to labels. For simplicity, we omitted the numbers of changes in definitions, usage notes, domains, ranges, and cardinalities, but these are also often made in the process of defining a new AP. Each class profile and property profile means that it needs to be created, an IRI needs to be assigned to it, it should be linked to the reused term in the reused vocabulary or AP, and then the terminological changes, if necessary, are made. In the case of property profiles, context-specific domains, profiles, and cardinalities are specified.

In this section, we compare the process of creating one class profile manually, in the SEMIC toolchain, and in Dataspecer, to demonstrate the difference in required effort. This effort can then be scaled to the number of class profiles and property profiles in the APs.

Creating a class profile of a class or another class profile manually means:

1. Assigning an IRI

2. Copying the profiled class or class profile IRI to maintain the link
3. Copying and optionally modifying the label
4. Copying and optionally modifying the definition
5. Copying and optionally modifying the usage note
6. Manually adding the facts in the textual part of the human-readable documentation
7. Manually adding the facts in the diagram part of the human-readable documentation
8. Manually adding facts in a machine-readable file

Creating a class profile of a class or another class profile in the SEMIC toolchain means:

1. Creating a UML class in the Enterprise Architect project
2. Assigning an IRI
3. Copying the profiled class or class profile IRI to maintain the link
4. Copying and optionally modifying the label
5. Copying and optionally modifying the definition
6. Copying and optionally modifying the usage note
7. Representing the facts as tagged values of a UML class in the source Enterprise Architect project

After the whole AP is created in the SEMIC toolchain, the toolchain needs to run to generate the human-readable documentation and validating SHACL shapes.

Creating a class profile of a class or another class profile in Dataspecer means:

1. Clicking the "profile" button on the profiled class or class profile, automatically generating the profile IRI and maintaining the link to the profiled class or class profile
2. Optionally modify the pre-filled label, definition, and usage note.

After the whole AP is created in Dataspecer, the resulting human-readable and machine-readable files are generated automatically. In addition, it is possible to "autoprofile" a vocabulary or application profile, automatically creating profiles of all terms in the "reuse as-is" fashion, leaving only the required changes to the user.

In the case of creating a property profile, the domain, range, and a compatible cardinality need to be manually defined in addition to what is necessary when creating a class profile. In the manual approach and the SEMIC toolchain approach, these steps are again manual. Using Dataspecer, the compatible cardinality is automatically pre-filled, but domains and ranges need to be specified manually as well. Overall, when the described effort is multiplied by the higher 10s of profiles present in the DCAT-based use cases, substantial effort is saved by using Dataspecer.

Technical Schema Design Process Comparison

In the rest of this chapter, we determine the relative times it takes to create technical schemas, specifically JSON Schemas, in Dataspecer compared to the manual approach. Our goal is to demonstrate that Dataspecer is useful and efficient for technical schema authoring, even for small SDSes without major requirements. Users can therefore use Dataspecer from the very beginning and thus create higher-quality specifications instead of postponing its use only for more complex cases.

By manual approach, we mean writing JSON Schema documents directly in an IDE of the user's choice, without any support for concept suggestion or automatic generation of chunks of the schema. In order to produce a schema in Dataspecer, the user needs to first create a vocabulary, then its application profile, which can be "autoprofiled", and then derive the technical schema from it. We compare these two workflow approaches. The Dataspecer approach produces JSON Schema, but also other artifacts, such as human-readable documentation or RDFS serialization of the vocabulary. These, however, are not part of the evaluation.

We selected JSON Schema as the target format for the evaluation because SDSes in our evaluation are JSON-based. However, the results of the evaluation are generalizable to other technical formats.

Evaluation Methodology and Cost Model

Directly measuring the time required to create large, real-world SDSes is impractical due to the significant duration of such tasks and the difficulty of controlling variables across multiple participants. Therefore, we adopt the cost per operation based evaluation framework. We first measure the time required to perform atomic modeling tasks using both manual and Dataspecer-based approaches. We then estimate the total time required to create the selected real-world SDSes by summing the costs of the individual tasks, weighted by their count in the specifications.

We selected the following real-world SDSes:

1. A FOS for publishing dissertation theses topics by universities[42]. It is based on part of SGoV that models the university domain and defines two data structures: a simple overview of a topic and a topic detail.

2. A collection of 19 FOSes[43] (including the above mentioned tourist destinations) for local administrations. They form a mutually interrelated system of data specifications, with more generic FOSes reused by more specific ones. They are based on SGoV and also on Schema.org, Registered Organization Vocabulary (Archer et al. (2013)), and several other vocabularies.

3. DCAT-AP-CZ - a Czech profile of DCAT-AP.

4. A collection of 13 data specifications for the *Czech Register of Rights and Obligations*. Each defines a single data structure, but the structures intersect on different semantic concepts, hence having a significant overlap.

5. A collection of data specifications that define data structures for data exchange among services in a service-oriented architecture of a study information system of our university. The specifications are based on semantic models designed in a commercial UML modeling tool. Each is based on one or more semantic models and defines various data structures.

These SDSes are sorted by their complexity, from the least complex to the most complex. We define complexity as the

Table 2. Relative cost for conceptual modeling tasks in Dataspecer. These tasks are necessary for the initial definition of the domain vocabulary and its associated Application Profile (AP).

Task	Manual	Dataspecer
Class creation	*n/a*	86 ± 13
Attribute definition	*n/a*	89 ± 10
Association	*n/a*	122 ± 22
Generalization (Extension)	*n/a*	56 ± 18

Table 3. Relative cost for technical schema design tasks. The manual baseline involves direct JSON Schema authoring, while the Dataspecer workflow involves deriving structures from the conceptual model.

Task	Manual	Dataspecer
Schema root initialization	125 ± 27	29 ± 5
Attribute (standard)	146 ± 8	39 ± 6
Attribute (IRI type)	174 ± 5	42 ± 7
Association to class	159 ± 15	37 ± 7
Reference to other schema	160 ± 16	89 ± 16
Class extension	149 ± 27	78 ± 13
Choice between classes	151 ± 20	52 ± 10
@id, @type attributes	151 ± 16	*n/a* (auto)

average number of technical schemas in which the concept is used. While the first three SDSes represent relatively isolated domains with few schemas, the final two cases involve high levels of "semantic entanglement", where manual tracking of dependencies when updating concepts becomes cognitively demanding.

Atomic Task

We conducted a study with 8 participants to measure the duration of the atomic operations. The participant group consisted of a mix of 4 regular Dataspecer users and 4 individuals familiar with semantic technologies but new to the tool. Notably, two participants had previously participated in the manual design of Czech FOSes, which allowed them to measure time more objectively based on their prior practical experience. To ensure that the results reflect steady-state professional productivity rather than the initial learning curve, participants first completed a set of practice tasks. They were then asked, when reporting task completion time, to adjust the value if they believed they were still learning during the tasks and had not yet reached full productivity.

Their goal was to create the JSON Schema from the given domain description and measure individual tasks, such as creating a class in Dataspecer or typing an attribute in an IDE.

The time measurements were normalized to an average value of 100 per task to define a *relative cost*. These cost values represent the cognitive and temporal effort required for specific actions. The results for conceptual modeling and schema design are detailed in Table 2 and Table 3.

All results obtained from the survey, including the questionnaires administered to the participants, are archived on Zenodo[44].

Productivity Discussion

Using the atomic costs, we estimated the total effort for the five complex SDSes (Table 4). For the simplest case (SDS 1), the manual approach appears more efficient. This is expected, as the overhead of creating a vocabulary in Dataspecer is not yet offset by the speed of schema generation in a simple domain.

However, as complexity increases, the productivity of Dataspecer improves significantly relative to the manual baseline. Specifically, Dataspecer becomes productive starting from the second case (SDS 2) for JSON Schema generation. By SDS 5, the Dataspecer approach is nearly twice as productive (Ratio 56%).

Table 4. Estimated total cost for the five selected SDSes. The ratio compares the effort of creating a full SDS in Dataspecer against creating only a JSON Schema manually.

SDS	Manual	Dataspecer	Ratio
1.	3,800 ± 200	4,600 ± 400	(123 ± 13) %
2.	25,500 ± 1,300	24,200 ± 2,100	(95 ± 9) %
3.	7,300 ± 200	6,100 ± 400	(83 ± 6) %
4.	35,400 ± 1,100	27,300 ± 2,100	(77 ± 6) %
5.	17,600 ± 600	9,800 ± 600	(56 ± 4) %

This means that the tool's ability to suggest concepts and automate syntactic correctness offsets the effort of conceptual modeling. In manual workflows, duplicating similar attributes was noted as a speed advantage, but this advantage is lost in complex domains where consistency and cross-referencing become the primary bottlenecks.

It is important to note that, although the initial task was to create JSON Schemas only, the Dataspecer workflow inherently produced other artifacts, such as the documentation of both the conceptual model and the JSON Schema and RDFS Vocabulary serialization. It checked the consistency of the created artifacts, and in case of future changes, it would ensure that the changes are propagated across all artifacts and that the resulting specification remains consistent.

The respondents preferred our approach due to its elimination of the need to address the syntactic correctness of the resulting schema and due to the suggestion of concepts that can be used for modeling. Therefore, they were not required to keep the entire model in their heads. In addition, they appreciated the ability of the tool to mitigate the risks of errors from repetitive tasks.

In contrast, for particular schemas with numerous, very similar attributes, respondents appreciated the capability of the manual approach to easily duplicate similar attributes, thus significantly accelerating the schema development process. In relation to more complex and less frequent operations, the cost between Dataspecer and the conventional methods equalized, as participants had to recall the operation and confirm its accurate functionality.

Usability Analysis

During the evaluation, we asked our respondents the 10 standard SUS questions since their task was to effectively design full SDS in order to obtain JSON Schema. The final score is **58**, which is below the typical threshold of **68** considered to be the minimal score to be *acceptable*, even though most of our respondents consider themselves experts or at least advanced users in the area of semantic modeling.

The main issue we see from the usability point of view is the notion of AP or DAP, as Dataspecer, to some extent, distinguishes between the vocabulary and AP instead of having a single conceptual model. The worst-rated question was *Q10: I needed to learn a lot of things before I could get going with this system*, with an average score of **2.6** out of 5 (1 means *strongly disagree*, 5 means *strongly agree*). This is understandable, as the users still need basic knowledge of vocabularies, reuse, and technical schemas. The best-rated question was *Q5: I found the various functions in this system were well integrated*, with an average score of **4.2**.

Conclusions and Future Work

In this paper, we introduced *Dataspecer*, an open-source, modular web application for authoring and managing Semantic Data Specifications (SDSes), covering conceptual modeling of (i) vocabularies, (ii) their application profiles with explicit support for reuse, and (iii) the derivation of technical schemas in various structures. The resulting SDSes contain rich metadata, human-readable documentation with diagrams, and machine-readable artifacts depending on the target technology: RDFS+OWL, JSON Schema with context, XSD, XSLT transformations, CSVW, and data samples.

We successfully demonstrated the applicability of Dataspecer on DCAT and its complex ecosystem of application profiles. The tool is currently used in Czechia for developing the government specifications ecosystem and in several other projects.

Although Dataspecer already covers the core lifecycle of practical SDS authoring, ongoing development and research focus on the following key areas:

- Supporting users in making their specifications more FAIR by checking all the necessary requirements and providing them with instructions on how to properly publish the specifications.
- Versioning, change management, and propagation of changes (evolution) across profile hierarchies, including publishing those changes in open standards with supporting artifacts, such as migration scripts.
- Git and GitHub/GitLab integration. These platforms are well-tailored for large communities with support for issues, pull requests, user management, and CI/CD, allowing specification creation processes to be tailored to domain needs.
- Capturing provenance information about critical decisions made during development, including their management and reasoning, to make them more **R**eusable from a FAIR perspective.
- Developing open standards for SDS description to enhance adoption and FAIRness while also contributing to standardization and methodology development.

Acknowledgements

The work was supported by the Charles University GAUK project no. 262823, by project **National Repository Platform for Research Data** no. CZ.02.01.01/00/23_014/0008787, and by the SVV project number 260821.

Notes

1. https://interoperable-europe.ec.europa.eu/collection/iopeu-monitoring/european-interoperability-framework-detail
2. https://semiceu.github.io/style-guide/1.0.0/
3. https://www.w3.org/TR/ld-bp/#VOCABULARIES
4. http://xmlns.com/foaf/spec/
5. https://www.dublincore.org/specifications/dublin-core/dcmi-terms/
6. https://www.w3.org/TR/shacl/
7. https://interoperable-europe.ec.europa.eu/collection/semic-support-centre/application-profiles-what-are-they-and-how-model-and-reuse-them-properly-look-through-dcat-ap
8. https://lov.linkeddata.es/dataset/lov
9. http://data.europa.eu/r5r/
10. https://dataspecer.com
11. https://github.com/dataspecer/dataspecer
12. https://respec.org/docs/
13. https://speced.github.io/bikeshed/
14. https://handlebarsjs.com/
15. https://rml.io/specs/rml/
16. https://www.dia.gov.cz/
17. https://bun.sh/
18. https://semiceu.github.io/toolchain-manual/
19. https://sparxsystems.com/
20. https://semiceu.github.io/style-guide/1.0.0/gc-conceptual-model-conventions.html
21. https://tietomallit.suomi.fi/
22. https://www.w3.org/ns/dcat-external.ttl
23. https://mff-uk.github.io/specifications/dcat-dap/
24. https://data.europa.eu/r5r/
25. https://mff-uk.github.io/specifications/dcat-ap/
26. https://semiceu.github.io/DCAT-AP/releases/3.0.0/#CatalogueRecord.title
27. https://www.w3.org/TR/vocab-dcat-3/#Property:record_title
28. https://www.dublincore.org/specifications/dublin-core/dcmi-terms/#http://purl.org/dc/elements/1.1/title
29. https://ofn.gov.cz/dcat-ap-cz/
30. https://semiceu.github.io/GeoDCAT-AP/releases/3.0.0/#detailed-usage-notes-and-examples
31. https://ofn.gov.cz/záznam-informačního-systému-nahrazující-úředně-ověřený-podpis/
32. https://ofn.gov.cz/společné-části-elektronických-dokumentů/
33. https://ofn.gov.cz/dokumenty-z-formulářů-ZoPDS/
34. https://ofn.gov.cz/osvědčení-digitálního-úkonu/
35. https://ofn.gov.cz/metadata-dokumentů-veřejnoprávních-původců/
36. https://ofn.gov.cz/registr-práv-a-povinností/
37. https://ofn.gov.cz/slovníky/
38. https://ofn.gov.cz/dobrovolné-svazky-obcí/

39. https://www.eosc.cz/en
40. https://data.europa.eu
41. https://eosc-cz.github.io/CCMM/en/
42. https://ofn.gov.cz/témata-dizertačních-prací/
43. https://data.gov.cz/ofn/
44. https://doi.org/10.5281/zenodo.18907258

References

Adida B, Herman I, Birbeck M and McCarron S (2015) RDFa Core 1.1 - Third Edition. W3C Recommendation, W3C. URL https://www.w3.org/TR/2015/REC-rdfa-core-20150317/.

Akhvlediani A (2024) *Expanding the Dataspecer Tool for Streamlined API Creation and Management*. Master's Thesis, Charles University, Faculty of Mathematics and Physics. URL http://hdl.handle.net/20.500.11956/193756.

Archer P, Papantoniou A and Meimaris M (2013) Registered Organization Vocabulary. W3C Note, W3C. URL https://www.w3.org/TR/2013/NOTE-vocab-regorg-20130801/.

Atkinson C and Kühne T (2003) Model-driven development: a metamodeling foundation. *IEEE Softw.* 20(5): 36–41. URL https://doi.org/10.1109/MS.2003.1231149.

Baker T and Coyle K (2009) Guidelines for Dublin Core™ Application Profiles. https://www.dublincore.org/specifications/dublin-core/profile-guidelines/.

Bayerlein B, Schilling M, Birkholz H, Jung M, Waitelonis J, Mädler L and Sack H (2024) Pmd core ontology: Achieving semantic interoperability in materials science. *Materials & Design* 237: 112603. DOI:https://doi.org/10.1016/j.matdes.2023.112603. URL https://www.sciencedirect.com/science/article/pii/S0264127523010195.

Brooke J et al. (1996) SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189(194): 4–7.

Buyle R, De Vocht L, Van Compernolle M, De Paepe D, Verborgh R, Vanlishout Z, De Vidts B, Mechant P and Mannens E (2016) Oslo: Open standards for linked organizations. In: *Proceedings of the international conference on electronic governance and open society: Challenges in Eurasia*. pp. 126–134. DOI: 10.1145/3014087.3014096. URL https://dl.acm.org/doi/abs/10.1145/3014087.3014096.

Car N (2019) The Profiles Vocabulary. W3C Working Group Note, W3C. URL https://www.w3.org/TR/2019/NOTE-dx-prof-20191218/.

Champin PA, Longley D and Kellogg G (2020) JSON-LD 1.1. W3C Recommendation, W3C. URL https://www.w3.org/TR/2020/REC-json-ld11-20200716/.

Clark K, Williams G, Torres E and Feigenbaum L (2013) SPARQL 1.1 protocol. W3C recommendation, W3C. https://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/.

Cock JD, Dhondt E, Fragkou P, Klímek J and Sofou A (2025) DCAT-AP 3.0.1. Technical report, European Commission. URL https://semiceu.github.io/DCAT-AP/releases/3.0.1/.

Corcho O, Fernández-López M, Gómez-Pérez A and López-Cima A (2005) Building legal ontologies with methontology and webode. In: *Law and the semantic web: legal ontologies, methodologies, legal information retrieval, and applications*. Springer, pp. 142–157. URL https://link.springer.com/chapter/10.1007/978-3-540-32253-5_9.

Corcho Associate professor O, Poveda-Villalón PhD student M and Gómez-Pérez Full professor A (2015) Ontology engineering in the era of linked data. *Bulletin of the Association for Information Science and Technology* 41(4): 13–17. URL https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/bult.2015.1720410407.

Cox S and Little C (2022) Time Ontology in OWL. Candidate Recommendation, W3C. URL https://www.w3.org/TR/2022/CRD-owl-time-20221115/.

Doan A, Halevy AY and Ives ZG (2012) *Principles of Data Integration*. Morgan Kaufmann. ISBN 978-0-12-416044-6. URL http://research.cs.wisc.edu/dibook/.

Espinoza-Arias P, Garijo D and Corcho O (2021) Crossing the chasm between ontology engineering and application development: A survey. *Journal of Web Semantics* 70: 100655. DOI:https://doi.org/10.1016/j.websem.2021.100655. URL https://www.sciencedirect.com/science/article/pii/S1570826821000305.

FHIR Infrastructure Work Group (2023) Profiling – fhir v5.0.0. https://hl7.org/fhir/profiling.html. URL https://hl7.org/fhir/profiling.html. Normative specification page, part of the *Fast Healthcare Interoperability Resources (FHIR)* standard.

Garijo D and Poveda-Villalón M (2020) Best Practices for Implementing FAIR Vocabularies and Ontologies on the Web. In: *Applications and practices in ontology design, extraction, and reasoning*. IOS Press, pp. 39–54. URL http://dx.doi.org/10.3233/SSW200034.

Gora M (2025) *Searching classes in the Wikidata ontology*. Master's Thesis, Charles University, Faculty of Mathematics and Physics. URL http://hdl.handle.net/20.500.11956/197453.

Guarino N, Oberle D and Staab S (2009) What is an ontology? In: *Handbook on Ontologies*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-92673-3, pp. 1–17. URL https://doi.org/10.1007/978-3-540-92673-3_0.

Guha R and Brickley D (2014) RDF Schema 1.1. W3C Recommendation, W3C. https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

Guizzardi G (2005) *Ontological foundations for structural conceptual models*. PhD Thesis, University of Twente. URL https://ris.utwente.nl/ws/portalfiles/portal/6042428/thesis_Guizzardi.pdf.

Heery R and Patel M (2000) Application profiles: Mixing and matching metadata schemas. *Ariadne* 25. URL http://www.ariadne.ac.uk/issue/25/app-profiles/.

Hyland B, Atemezing GA and Villazón-Terrazas B (2014) Best Practices for Publishing Linked Data. W3C Note, W3C. URL https://www.w3.org/TR/2014/NOTE-ld-bp-20140109/.

International Organization for Standardization and International Electrotechnical Commission (1998) Information technology — Framework and taxonomy of International Standardized Profiles — Part 1: General principles and documentation framework. Technical Report ISO/IEC TR 10000-1:1998 (Ref. 30726), ISO/IEC. URL https://www.iso.org/standard/30726.html.

Jordan M, Schönhals S and Auer S (2025) Enhancing interoperability in digital calibration data exchange: A case for ontology development. *Measurement: Sensors* 38: 101459. DOI:https://doi.org/10.1016/j.measen.2024.101459. URL https://www.sciencedirect.com/science/article/pii/S2665917424004355. Proceedings of the XXIV IMEKO World Congress.

Kellogg G and Tennison J (2015) Model for Tabular Data and Metadata on the Web. W3C Recommendation, W3C. URL https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/.

Klíma K, Taelman R and Nečaský M (2023) Ldkit: Linked data object graph mapping toolkit for web applications. In: *The Semantic Web – ISWC 2023*. Cham: Springer Nature Switzerland. ISBN 978-3-031-47243-5, pp. 194–210. DOI:10.1007/978-3-031-47243-5_11. URL https://link.springer.com/chapter/10.1007/978-3-031-47243-5_11.

Klímek J, Koupil P, Škoda P, Bártík J, Štěpán Stenchlák, Nečaský M and Holubová I (2023) Atlas: A toolset for efficient model-driven data exchange in data spaces. In: *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, pp. 4–8. DOI:10.1109/MODELS-C59198.2023.00009. URL https://ieeexplore.ieee.org/document/10350726.

Klímek J, Štěpán Stenchlák, Polický A, Škoda P and Nečaský M (2024) Towards Authoring of Vocabularies and Application Profiles using Dataspecer. In: Etcheverry L, Garcia VL, Osborne F and Pernisch R (eds.) *Proceedings of the ISWC 2024 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 23nd International Semantic Web Conference (ISWC 2024), Hanover, Maryland, USA, November 11-15, 2024*, *CEUR Workshop Proceedings*, volume 3828. CEUR-WS.org, pp. 1–5. URL https://ceur-ws.org/Vol-3828/paper33.pdf.

Klímek J, Štěpán Stenchlák and Škoda P (2025a) Data Specification Vocabulary - Default Application Profile (DSV-DAP). Technical report, Charles University, Department of Software Engineering. URL https://w3id.org/dsv-dap#.

Klímek J, Štěpán Stenchlák and Škoda P (2025b) Data Specification Vocabulary (DSV). Technical report, Charles University, Department of Software Engineering. URL https://w3id.org/dsv#.

Klímek J, Štěpán Stenchlák and Škoda P (2025c) Data Specification Vocabulary (DSV): Representation of Application Profiles of Semantic Data Specifications. In: *The 27th International Conference on Information Integration and Web Intelligence (iiWAS2025)*, Lecture Notes in Computer Science. Springer, pp. 221–236. DOI:10.1007/978-3-032-11976-6_16.

Köcher A, Markaj A and Fay A (2022) Toward a generic mapping language for transformations between rdf and data interchange formats. In: *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*. pp. 1–4. DOI:10.1109/ETFA52439.2022.9921513. URL https://ieeexplore.ieee.org/abstract/document/9921513.

Křemen P and Nečaský M (2019) Improving discoverability of open government data with rich metadata descriptions using semantic government vocabulary. *J. Web Semant.* 55: 1–20. URL https://doi.org/10.1016/j.websem.2018.12.009.

Lausen H and Farrell J (2007) Semantic Annotations for WSDL and XML Schema. W3C Recommendation, W3C. URL https://www.w3.org/TR/2007/REC-sawsdl-20070828/.

Lebo T, Sahoo S and McGuinness D (2013) PROV-O: The PROV Ontology. W3C Recommendation, W3C. URL https://www.w3.org/TR/2013/REC-prov-o-20130430/.

Masmoudi M, Ben Abdallah Ben Lamine S, Karray MH, Archimede B and Baazaoui Zghal H (2024) Semantic data integration and querying: A survey and challenges. *ACM Comput. Surv.* 56(8). DOI:10.1145/3653317. URL https://doi.org/10.1145/3653317.

Miles A and Bechhofer S (2009) SKOS Simple Knowledge Organization System Reference. W3C Recommendation, W3C. URL https://www.w3.org/TR/2009/REC-skos-reference-20090818/.

Moxon SAT, Solbrig H, Unni DR, Jiao D, Bruskiewich RM, Balhoff JP, Vaidya G, Duncan WD, Hegde H, Miller M, Brush MH, Harris NL, Haendel MA and Mungall CJ (2021) The Linked Data Modeling Language (LinkML): A General-Purpose Data Modeling Framework Grounded in Machine-Readable Semantics. In: Hastings J and Barton A (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*, *CEUR Workshop Proceedings*, volume 3073. CEUR-WS.org, pp. 148–151. URL https://ceur-ws.org/Vol-3073/paper24.pdf.

Musen MA (2015) The protégé project: a look back and a look forward. *AI Matters* 1(4): 4–12. DOI:10.1145/2757001.2757003. URL https://doi.org/10.1145/2757001.2757003.

Noy NF, McGuinness DL et al. (2001) Ontology development 101: A guide to creating your first ontology. URL https://protege.stanford.edu/publications/ontology_development/ontology101.pdf.

Nuyts E, Werbrouck J, Verstraeten R and Deprez L (2023) Validation of building models against legislation using shacl. In: *LDAC2023: Linked Data in Architecture and Construction Week*, volume 3633. CEUR, pp. 164–175. URL http://hdl.handle.net/1854/LU-01HCHF9JK71QP7RQCEZCQK49WC.

OMG (2015) *OMG Unified Modeling Language (OMG UML), version 2.5*. OMG.

Park Jr and Childress E (2009) Dublin core metadata semantics: An analysis of the perspectives of information professionals. *Journal of Information Science* 35(6): 727–739.

Perego A, Albertoni R, Browning D, Cox S, Winstanley P and Beltran AG (2024) Data Catalog Vocabulary (DCAT) - Version 3. W3C Recommendation, W3C. URL https://www.w3.org/TR/2024/REC-vocab-dcat-3-20240822/.

Poveda-Villalón M, Espinoza-Arias P, Garijo D and Corcho O (2020) Coming to terms with fair ontologies. In: Keet CM and Dumontier M (eds.) *Knowledge Engineering and Knowledge Management*. Cham: Springer International Publishing. ISBN 978-3-030-61244-3, pp. 255–270.

Prokop D, Stenchlák Š, Škoda P, Klímek J and Nečaský M (2025) Enhancing domain modeling with pre-trained large language models: An automated assistant for domain modelers. In: Maass W, Han H, Yasar H and Multari N (eds.) *Conceptual Modeling*. Cham: Springer Nature Switzerland. ISBN 978-3-031-75872-0, pp. 235–253. URL https://link.springer.com/chapter/10.1007/978-3-031-75872-0_13.

Stenchlák S, Nečaský M, Škoda P and Klímek J (2022) DataSpecer: A Model-Driven Approach to Managing Data Specifications. In: *The Semantic Web: ESWC 2022 Satellite Events - Hersonissos, Crete, Greece, May 29 - June 2, 2022, Proceedings*, *LNCS*, volume 13384. Springer, pp. 52–56. DOI: 10.1007/978-3-031-11609-4_10. URL https://doi.org/10.1007/978-3-031-11609-4_10.

Tan H, Kebede RZ, Moscati A and Johansson P (2024) Semantic interoperability using ontologies and standards for building product properties. In: *12th Linked Data in Architecture and Construction Workshop, Bochum, Germany, June 13-14, 2024*. CEUR-WS, pp. 23–35.

Thompson H, Beech D, Maloney M, Mendelsohn N, Sperberg-McQueen M and Gao S (2012) W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. W3C Recommendation, W3C. URL https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/.

Vogt L, Strömert P, Matentzoglu N, Karam N, Konrad M, Prinz M and Baum R (2025) Suggestions for extending the fair principles based on a linguistic perspective on semantic interoperability. *Scientific Data* 12(1). DOI:10.1038/s41597-025-05011-x.

Westermann T, Köcher A and Gehlhoff F (2025) Automated validation of textual constraints against automationml via llms and shacl. *arXiv preprint arXiv:2506.10678* .

Wilkinson MD, Dumontier M, Aalbersberg IJ et al. (2016) The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data* 3(1). URL http://dx.doi.org/10.1038/sdata.2016.18.

Wood D, Lanthaler M and Cyganiak R (2014) RDF 1.1 Concepts and Abstract Syntax. W3C recommendation, W3C. URL https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

Wright A, Andrews H, Hutton B and Dennis G (2022) JSON Schema: A Media Type for Describing JSON Documents. Internet-Draft draft-bhutton-json-schema-01, Internet Engineering Task Force. URL https://datatracker.ietf.org/doc/draft-bhutton-json-schema/01/. Work in Progress.