OntoGrapher: a Web-based Tool for Ontological Conceptual Modeling

Alice Binder, Petr Křemen

Faculty of Electrical Engineering, Czech Technical University

Abstract. Conceptual models have traditionally been tools for sharing understanding of system's structure and behaviour with others. Although this is still the main use-case, some works have already shown usefulness of their machine readable form to derive data schemata, system documentation, as well as semantic vocabularies of the given system or domain. In this work we present OntoGrapher, a visual web-based tool for conceptual modeling based on the OntoUML conceptual modeling language. The tool accepts and produces machine readable outputs in the form of OWL ontologies and SKOS thesauri. We show its main features, benefits as well as use-cases in which the tool has been successfully applied and include a hands-on demo of the tool. Finally, we perform user testing evaluating the tool. A roadmap for the tool's future development is laid out based on the testing results and already planned features.

1 Introduction

Conceptual modeling is a traditional discipline of system and data engineering. Although there exist various languages for conceptual modeling (like E-R models [4], Archimate [5], or UML [13]), most of them are aimed at visual representation of the conceptualization to be shared among domain experts, IT experts and other stakeholders. Complex conceptual models are prone to errors which are difficult to detect (e.g. rigid types `Person`, `Company` being subtypes of anti-rigid/contingent type `PropertyOwner`). To tackle such problems, OntoUML, an ontology-based conceptual modeling language has been introduced [8]. Although a few OWL serializations for OntoUML have been introduced [3], to the best of our knowledge there is no tool allowing to create interlinked OntoUML models using SKOS [10] and OWL [12] as their native formats. This significantly limits the knowledge reuse by machines and integration into the current linked data stack. Now, let's briefly introduce the overall scenario where the need for such tool a comes from.

1.1 Scenario: supporting public sector data management in Czechia

The Ministry of the Interior of the Czech Republic and the Department of the eGovernment Chief Architect is responsible for the design and optimization of

digital services as well as for the coordination of the design of the data and information architecture across the Czech government. As part of its agenda, it maintains the necessary eGovernment legislation comprising thirteen key acts. These include e.g. the Basic Registry Act No. 111/2009 Coll., the Act No. 365/2000 Coll., on information systems of the public administration, and Act No. 106/1999 Coll. on Free Access to Information. In 2018, they created a non-legal document Information Concept of the Czech Republic, a vision of the Czech public sector data and services, which is further detailed in several other documents – the National Architecture Plan and the National Architecture Framework and also the eGovernment vocabulary. The latter lists selected concepts from the aforementioned laws, as well as from the Czech norms, together with custom concepts defined by the Department of the Chief Architect.

The terminology in the public sector often stems from the legislation, which contains many well defined terms. However many others are not well defined, yet heavily used. Open data stand on the opposite side. Often, they are just selections of data existing in governmental information systems - yet the connection between the open data, the data in the original information systems and the requirements given by legislation is missing. This complicates checking how well they reflect the legislation, whether there is a novel requirement to publish new open data, or what is the correspondence to the data in the original information systems.

To support answering such questions related to auditability of open data, a data schema management process has been designed, as depicted in Figure 1.

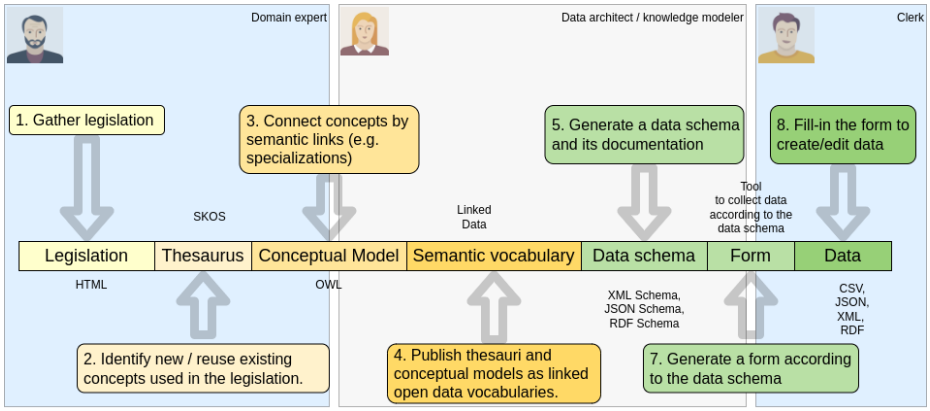

Fig. 1. From legislation to data schemas.

The process is based on open semantic web standards, an obvious benefit of which are the (linked) open data principles, so that institutions can communicate not only more efficiently, but also more transparently with citizens and organizations. In order to do so, the meaning of open data (its entities, relations

and attributes) needs to be conveyed to the consumers as well. Various legal concepts, requirements, processes, and relationships or distinctions between the aforementioned concepts have to be abided by.

The process expects involvement of domain experts who are able to identify key concepts, knowledge modelers who are able to design a formal ontology on their top, publish it as linked data and create data schemas out of it. Finally, clerks help to fine-tune the automatically generated forms and curate the open data collected through these forms.

For the rest of the paper, let's zoom in to the details of the conceptual model creation with OntoGrapher.

2 OntoGrapher

With the requirements mentioned in section 1 in mind, we introduce OntoGrapher as a tool for clerks with background in conceptual modeling (or domain experts keen to get it), but not necessarily versed in OWL[1]. In particular, OntoGrapher has the following goals:

- offer a collaborative web-based conceptual modeling environment for domain experts,
- accept SKOS terminologies on input and produce OWL ontologies on output without requiring users to understand either of these standards,
- track the source of entities, relations and attributes in conceptual models stemming from multiple vocabularies (e.g. laws).

As a result of these requirements, OntoGrapher is implemented as a web application with the following key features:

Interaction with online services OntoGrapher uses and manipulates data from triple stores with RDF4J APIs, authenticates the user with OpenID Connect-compatible servers, and is designed to be an online application from the ground-up. This streamlines the overall process of managing, sharing and publishing vocabularies, since users don't have to import them into a dedicated desktop application and any changes are stored on a server - ready to be published whenever the user is ready to do so.

Artifact export In addition to modifying thesauri and ontologies in-place, the application is capable of exporting the view of the model (as arranged by the user) in either PNG or SVG formats.

Customizable visualization options i.e. concepts can be visualized using SKOS preferred or alternative labels. Also, diagrams can be visualized in E-R model syntax or a more compact UML-like syntax.

[1] see the web page of the acknowledged grant project, `https://data.gov.cz/english/` or the project's GitHub repositories at `https://github.com/opendata-mvcr`. An up-to-date Czech version is available at `https://data.gov.cz`.

OntoUML Support The tool supports categorization of concepts using OntoUML [8] stereotypes to give basic ontological distinctions to the concepts.

Data validation To validate the model against OntoUML constraints and quality rules (e.g. presence of a label in a predefined language, minimal length of definition, etc.), the user can take advantage of a validation service. The service checks the vocabularies' data consistency.

2.1 Workflow

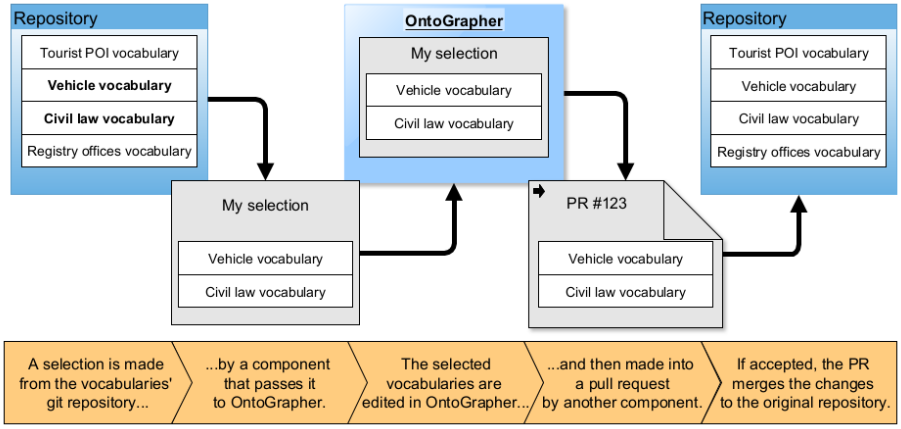

Fig. 2. Example workflow of modifying the central repository's vocabularies. First, the user selects vocabularies they want to edit (or create). The selection is then passed to OntoGrapher. After the desired changes have been made, the resulting data is introduced as a PR to the repository, requiring manual review.

At its core, OntoGrapher takes SKOS thesauri (representing identified concepts for which conceptual models do not exist yet) and OWL ontologies (representing already created conceptual models), visualizes them, and manipulates them. The changes that users make are saved in real-time to a triple store. In addition, OntoGrapher commits its own specific data (element positions within the graph, diagram names, etc.). OntoGrapher reads the SKOS thesauri and OWL ontologies from the central repository for Czech government vocabularies[2]. The appendix talks about the vocabulary data structure in more detail.

OntoGrapher modifies these vocabularies in a *workspace*, which is a copy of a selection of vocabularies, so that changes in the workspace do not modify the original repository. Workspaces can have multiple vocabularies in them, which is useful for modifying vocabularies and their complements together (for example a

[2] `https://slovník.gov.cz`

Civil law vocabulary and a Marriage registry vocabulary). An example workflow with workspaces is described in Figure 2.

3 UFO integration

Of particular note is the interplay between OntoGrapher and UFO (Unified Foundational Ontology; the foundational ontology on which OntoUML is based) [7]. This is manifested in the (types of) data OntoGrapher works with. More specifically, all of the concepts created and managed in OntoGrapher are of types equivalent to UFO types thanks to the basic top-level ontology the concepts are ultimately subclasses of. OntoGrapher takes advantage of this by, for example, distinguishing types visually by shape[3].

OntoGrapher, as of the time of writing, supports concepts with at most two stereotypes[4], though at least one is required for the validity of the model (depending on the stereotype) as determined by the validation server; (i) Type stereotypes, which define the ontological nature of the concept - e.g. `Object Type`, `Event Type` - and (ii) OntoUML stereotypes serving to validate the conceptual model itself (e.g. rigidity). There are plans to support more stereotypes with more varied inferences in the future (see the Future work section below).

To improve the modeling experience, OntoGrapher offers only valid types of relationships to the user upon relationship creation, based on the stereotypes of the concepts involved in the given relationship.

4 Demo and the User Interface

A demo instance of OntoGrapher can be run as described at `https://ontographer.github.io/ontographer-demo/`. The website contains instructions to build and run the example as well as a walk-through of the prepared scenario. In this scenario, the user has been given a single vocabulary to edit - a vocabulary representing a law regulating the operation of vehicles on public roads. The aim of the editing effort is to clean up the vocabulary and fix any inconsistencies that arise. At the end, they should have a vocabulary that is visually legible, consistent, and comprehensive.

Figure 3 shows the main elements of the user interface. At all times, the user is presented with the canvas where the concepts and relationships are visualized along with a panel showing the vocabularies and concepts within the workspace as well as any concept search results.

An important distinction of concepts is between write-enabled concepts and read-only concepts. The editability of concepts is determined by the way their vocabularies entered the workspace - vocabularies that the workspace itself was created around are write-enabled, but the user can also search and introduce

[3] An example of this is shown in Figure 4.
[4] By stereotypes, we mean types defined in the aforementioned top-level ontology.

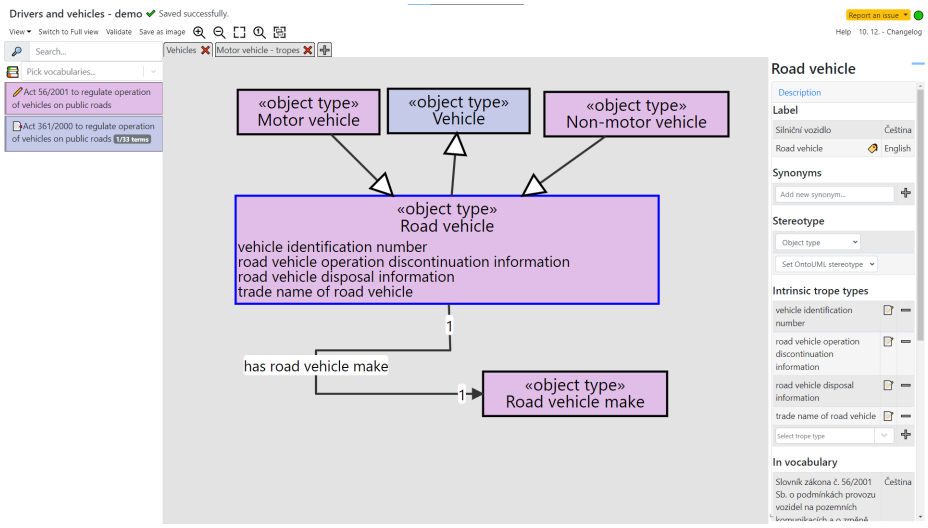

Fig. 3. Screenshot of OntoGrapher with the "Drivers and vehicles" workspace (from the demo) open with Compact view selected. The left panel lists all vocabularies and concepts in the model, while the right panel details a single selected concept. The tabs above the work area list diagrams. Concepts are color-coded to their respective vocabularies. The first vocabulary in order of appearance on the left panel is write-enabled and the second is read-only (as depicted by pictographs next to their titles).

concepts from the central repository that are outside the workspace. The consequence of the design of workspaces as described in section 2.1 is that these concepts cannot be edited.

In typical usage, therefore, a given workspace has at least one vocabulary designated as write-enabled (all its concepts are write-enabled) and can have other vocabularies that are read-only (all their concepts can be pulled into the model only as read-only). This means that they can be present in diagrams and manipulated visually, but their details or relationships can only be viewed, not edited. The demo uses this to find possibly redundant concepts that already exist in another vocabulary.

In addition to manipulation of their visual representation, write-enabled concepts allow editing of their various details, such as synonyms, and creation of relationships originating from those concepts. Concepts can of course also be deleted or created as part of a selected vocabulary.

The editing of concept and relationship attributes is done with clicking on the given element, which brings up another panel detailing all information about the element. The user can edit/view cardinalities, labels, display label (a selected pref/altLabel that a concept/relationship is displayed under in OntoGrapher), intrinsic tropes, stereotypes, and more. They can also see the list of all relationships involved with a concept (including outside the workspace) and filter through them or select which concepts related to a concept should be shown.

This enables showing only intrinsic tropes of a concept easily in the demo scenario.

Model elements are arranged into one or more *diagrams*, which are distinct "sections" of the entire model, containing some or all of the concepts and relationships within. It is not necessary for a concept in the model to be in any diagram; also, a single concept can be in multiple diagrams, as it is just an OntoGrapher-specific construct. Multiple diagrams are used in the demo to organize the large and complicated model into smaller sections.

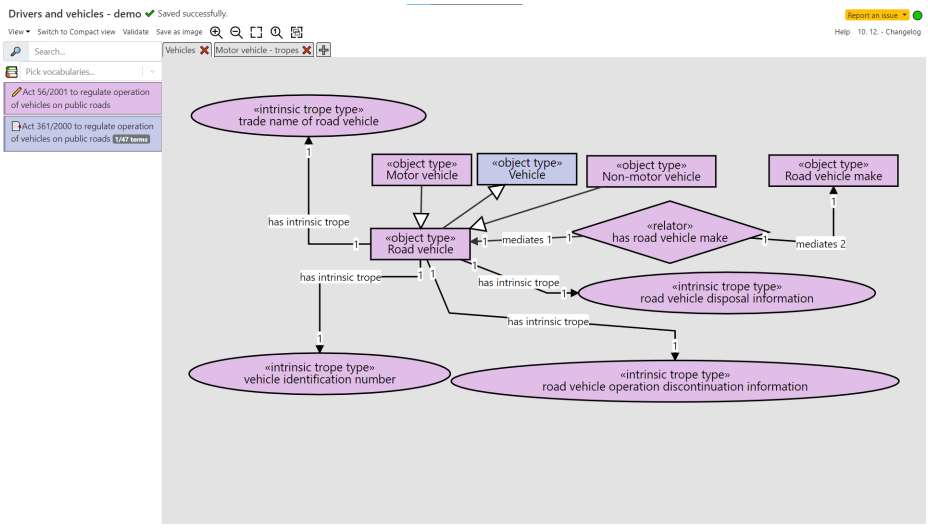

Fig. 4. Screenshot of OntoGrapher with the "Drivers and vehicles" workspace open with Full view selected. The data shown is the exact same as from Figure 3 - intrinsic tropes and relators are presented as concepts of their own, reflecting faithfully the underlying data structure.

There are two available views with which to visualize the diagram. *Full view* shows all connections and concepts as they are present in the data; i.e. all connections and concepts of all stereotypes. This is useful for experienced modelers, but may be undesirable to those who do not necessarily need the complete "behind-the-scenes" description of the model and want to focus on more conceptual modeling that OntoGrapher can provide. *Compact view* shrinks reified relationships into individual edges and aggregates concept tropes into the concepts themselves rather than displaying them as separate concepts connected to the given concept. Figures 3 and 4 display the same model in Compact and Full view, respectively.

During the process of modeling, several concepts and relationships may have been created, edited and deleted. To ensure validity of the conceptual models, OntoGrapher uses a validation service to check for consistency of the result-

ing model. The validation service uses SHACL[1] to check for consistency of completeness of labels/definitions, etc. Furthermore, it evaluates the OntoUML constructs by translating them to SHACL constraints. If there are any inconsistencies, OntoGrapher highlights the offending concepts so that the user can easily remedy any issue. An example constraint t7 from [8]:

Example 1.

$$\neg\exists x, y(Rigid(x) \wedge AntiRigid(y) \wedge x \sqsubseteq y) \tag{1}$$

is expressed by the rule

```
j-sgov-obecný:m6
  a              sh:NodeShape ;
  sh:severity    sh:Violation ;
  sh:message     "A Rigid type (e.g. Kind) must not be specialized
                    from an Anti-rigid one (e.g. Role)."@en ;
  sh:targetClass z-sgov-pojem:rigid;
  sh:property    [ sh:path                rdfs:subClassOf ;
                   sh:qualifiedValueShape
                      [ sh:class z-sgov-pojem:anti-rigid ] ;
                   sh:qualifiedMaxCount   0 ] .
```

Rigid(x) and AntiRigid(x) are expressed in the data as classes defined in the `z-sgov-pojem` top-level ontology for the central repository, which describes all basic classes for concepts as well as connections between concepts.

5 User testing

In order to systematically assess the user experience of OntoGrapher, user testing has been devised. The test consists of three parts: first, an assessment of the participant's experience with OntoGrapher, Semantic Web technologies and conceptual modeling, then, a short scenario (about 10-15 minutes; different to the one in the Demo section) is presented in which participants complete tasks that demonstrate a vertical slice of the application's functions. Lastly, participants' impressions are collected. The last part also contains questions through which the System Usability Scale [2] can be computed.

Six participants have been recruited through notices in communication channels of the previously mentioned grant project's members. The number of participants who claimed to have at least some knowledge of OntoGrapher, Semantic Web technologies, or conceptual modeling, is 4, 4, and 5 respectively.

The results of the testing gave OntoGrapher a System Usability Scale score of 73; as the average SUS score is 68 [2], this means that according to this scale, OntoGrapher is above average in user interface usability. All participants, with the exception of one participant's attempt to complete one task, managed to complete all six tasks. 27 percent of task attempts were successful, but required some help from the provided "cheat sheet" of available actions or took what they considered an unreasonable amount of time figuring out the process. The rest of

the attempts were reported to have been successful without any issues. Table 1 describes the tasks and attempts in more detail.

Task description	**Done with no difficulty**	**Done with difficulty**	**Couldn't do it**
Put concepts from the selected vocabularies on the canvas	6	0	0
Search for concepts outside the selected vocabularies	5	1	0
Create a relationship and adjust cardinalities	4	1	1
Create a concept and adjust concept tropes	3	3	0
Save an image of the canvas	6	0	0
Create a diagram and find concepts through a concept's relationships	4	2	0

Table 1. Results of the user testing. Each participant has been given a task, after which they are asked if (and how easily) they completed them. The numbers refer to the amount of participants (out of 6) that answered a given way for a given task.

The most mentioned issues were with the way the creation of concepts and relations worked, which two thirds found unintuitive in their impressions after the testing. Another problem was that participants weren't absolutely sure what was being asked of them in the testing. In terms of functions the participants would have liked the most, the most requested one is the ability to undo changes.

Something to note is that this testing strategy only measures the usability of the user interface without considering the usefulness of the application in real-life scenarios - this testing strategy only measures the usability of the application with regards to the creation of the OntoUML-based conceptual model over multiple vocabularies without considering the broader scenario of Figure 1. The whole scenario is currently under testing - together with other tools in the stack by the Ministry of Interior eGovernment department and other governmental partners.

6 Related work

This application is not the first effort made to simplify the process of modeling OntoUML conceptual diagrams. A significant contribution in this field is

by Guizzardi and his team at NEMO[5] with the OntoUML lightweight editor (OLED)[6]. The project was first released in 2015, along with an accompanying paper [6].

In the paper, the authors describe the various features of OLED, such as patterns and anti-pattern checking, OCL constraints[14], model verbalization, code generation, and more. As far as capabilities, then, the toolset of OLED is unparalleled compared to OntoGrapher, especially as OLED (and its offshoot, Menthor Editor [11]) was published before OntoGrapher was ever conceived.

However, as the contributions in the repository mostly ended after 2016 and the tool is incompatible with the latest Java version (our testing confirmed that it runs on Java 8, but not 16), the tool seems to be all but abandoned. In addition, while the number of features is impressive, the application's user interface is often unresponsive while using them and sometimes crashes entirely without an error prompt (even with models with as little as two classes, our testing showed).

The NEMO team has largely moved on to making an OntoUML plugin for the software engineering application Visual Paradigm[7]. This allows them to focus on implementing OntoUML-specific features directly without having to create a new modeling application base. As a result, the modeling experience is much smoother and more stable than with OLED.

The plugin offers modeling assistance in UML diagrams, for example filtering invalid stereotype relationships or validating models and diagrams. The models can be imported from JSON or exported to JSON or gUFO (a Turtle ontology file which imports and uses the gUFO base ontology to represent OntoUML constructs). Thanks to Visual Paradigm's ability to store its projects in an online database, users can collaborate on the model (the database contents are proprietary to Visual Paradigm - the aforementioned exports have to be generated by the client locally).

Another plugin for OntoUML modeling is for the OpenPonk modeling platform[8], developed by the OpenPonk team itself. The plugin can be downloaded directly with the application. While the application is competent with regards to modeling other classes of diagrams, the OntoUML plugin is quite limited; it offers the basic stereotypes and relationships with validity and anti-pattern checking, but the model itself cannot be exported to a format that can then be processed by anything other than OpenPonk.

OntoGrapher fulfills slightly different needs that the aforementioned solutions currently cannot; perhaps two of the most important are that the application should be connected to online services, where its results can be shared or published directly without the need to export, and have the ability to receive and

[5] Ontology and Conceptual Modeling Research Group - website at `https://nemo.inf.ufes.br/en/`

[6] The project's repository is at `https://github.com/nemo-ufes/ontouml-lightweight-editor`.

[7] Visual Paradigm's website is at `https://www.visual-paradigm.com/`. The plugin is available at `https://github.com/OntoUML/ontouml-vp-plugin/`.

[8] OpenPonk's website is at `https://openponk.org/`.

manipulate vocabularies with the wider Czech eGovernment initiative's specifications, which are a expansion of OntoUML with SKOS thesauri and other ontological categories.

7 Conclusions and future work

The user testing and general responses from the users of the application revealed that there are still some issues with regards to the user experience. In addition, as the tool is still in its prototyping phase, there are still many features to implement and many design details to improve in order for the tool to progress to a beta stage. Listed below are several examples of planned features for OntoGrapher:

Support for other formal ontologies The tool is currently tied in part to the basic top-level ontology of the Czech government vocabularies' repository, which prevents it from being applied in other domains.

Ontology Design Patterns We are working towards implementing an Ontology Design Pattern (ODP) language in OntoGrapher. ODPs help engineers create models more quickly, avoid certain frequently made inefficiencies and better prepare their models for any future extension or revision. The field of ODP languages, however, only a few years old and thus support of ODP usage with methodological approaches, classification, categorization, standardized distribution or implemented tools is still developing, so implementation of a language requires development of rigorous theoretical foundations first.

Change tracking Seeing the changes made to the vocabularies in the application in a clear and concise way would allow reviewers to work and identify problems faster. In subsequent development, it could also allow rollback of specific changes without affecting the rest of the vocabulary/vocabularies.

Modeling guidance While the application does feature validation with an accompanying SHACL constraint-checking server, the conflict-resolution and user guidance features are limited to non-existent. In addition, the user has to manually call the validation themselves in order to see possible violations. Some of this could be covered by the implementation of ODPs, but a feature wherein the validation errors can be resolved automatically/with a simple wizard or the user is guided through modeling with automatically setting correct cardinalities/relationships or warning about semantically invalid/anti-pattern sections (or closing the ability to create such sections in the first place) would also prove useful.

We believe that OntoGrapher is worth deploying in other contexts, since it enables a workflow in which domain experts do not have to worry themselves with the specifics of SKOS or OWL, but can focus directly on the creation of conceptual models in their domain without extensive help from conceptual modeling experts while keeping machine readability of the resulting models.

Appendix: Vocabulary description

As mentioned in section 2.1, the two main elements representing the model are (i) SKOS concepts (more specifically, `skos:Concept` instances[9]) and (ii) relationships (compiled as OWL restrictions on certain concepts with a certain relationship type and of a certain cardinality, when applicable).

All concepts and all connections also should have a type that is from or is a subclass of the basic top-level ontology to obey validity rules. This ontology mirrors OntoUML's classes and extends it with connections and other classes.

The following examples clarify the details of this arrangement. All of the examples will be represented in Turtle, with syntax highlighting for `owl`, `skos` and the `z-sgov-pojem` basic top-level ontology, which mirrors OntoUML's classes and relationship types.

Example 2. Let's start with one concept, a Record of birth, which will be defined in the Birth registry vocabulary.

```
gov-birth-registry:record-of-birth a skos:Concept,
    z-sgov-pojem:typ-objektu;
    skos:inScheme gov-birth-registry:scheme;
    rdfs:subClassOf gov-registry-offices:registry-record;
    skos:prefLabel "Record of birth"@en;
    skos:description "A record of a birth of a person
        stored in the birth registry office."@en.
```

This describes the concept as an Object type (with `z-sgov-pojem:typ-objektu`) and as belonging to the `gov-birth-registry` vocabulary (with a `gov-birth-registry:scheme` SKOS scheme). The concept is a subclass of Registry record - another concept[10].

Example 3. We could assign a trope to this concept, like so:

```
gov-birth-registry:record-of-birth
    z-sgov-pojem:má-vlastnost
        gov-birth-registry:datetime-of-birth.
```

`gov-birth-registry:datetime-of-birth` is an Intrinsic trope type class concept connected to our working concept via `z-sgov-pojem:má-vlastnost` (has intrinsic trope) - a connection defined in the basic ontology `z-sgov-pojem`.

Example 4. However, we would also like to describe a more complex connection, with cardinalities and detailed information about the nature of the connection. This is a common pattern with these vocabularies, and is expressed with Relator types and OWL restrictions in the following way:

[9] `skos:` prefix denotes the namespace `http://www.w3.org/2004/02/skos/core#`

[10] All concepts must have at least a preferred label, a SKOS scheme, and a skos:Concept type to be recognized as a concept. Anything else is, from the standpoint of OntoGrapher, optional. Note that having these required attributes does not make a concept or its connections inherently valid with respect to the validation rules.

```
# "Documents birth event" Relator type
gov-birth-registry:documents-birth-event a skos:Concept,
    z-sgov-pojem:typ-vztahu;
    skos:inScheme gov-registry-office:scheme;
    skos:prefLabel "documents birth event"@en;
    skos:altLabel "documents"@en.

# Connection from "Documents birth event" to "Record of birth"
gov-birth-registry:documents-birth-event rdfs:subClassOf
    [rdf:type owl:Restriction;
    owl:onProperty z-sgov-pojem:má-vztažený-prvek-1;
    owl:allValuesFrom gov-birth-registry:record-of-birth],
    [rdf:type owl:Restriction;
    owl:onProperty
        [owl:inverseOf z-sgov-pojem:má-vztažený-prvek-1];
    owl:allValuesFrom gov-birth-registry:documents-birth-event],
    [rdf:type owl:Restriction;
    owl:onProperty z-sgov-pojem:má-vztažený-prvek-1;
    owl:onClass gov-birth-registry:record-of-birth;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger],
    [rdf:type owl:Restriction;
    owl:onProperty
        [owl:inverseOf z-sgov-pojem:má-vztažený-prvek-1];
    owl:onClass gov-birth-registry:documents-birth-event;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger].

# Connection from "Documents birth event" to "Birth of person"
gov-birth-registry:documents-birth-event rdfs:subClassOf
    [rdf:type owl:Restriction;
    owl:onProperty z-sgov-pojem:má-vztažený-prvek-2;
    owl:allValuesFrom gov-civil-law:birth-of-person],
    [rdf:type owl:Restriction;
    owl:onProperty
        [owl:inverseOf z-sgov-pojem:má-vztažený-prvek-2];
    owl:allValuesFrom gov-birth-registry:documents-birth-event],
    [rdf:type owl:Restriction;
    owl:onProperty z-sgov-pojem:má-vztažený-prvek-2;
    owl:onClass gov-civil-law:birth-of-person;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger],
    [rdf:type owl:Restriction;
    owl:onProperty
        [owl:inverseOf z-sgov-pojem:má-vztažený-prvek-2];
    owl:onClass gov-birth-registry:documents-birth-event;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger].
```

The example expresses its cardinalites and relationships through OWL restrictions, using a Relator type *Documents birth event* concept mediating between the two involved concepts. The `z-sgov-pojem:typ-vztahu` type (Relator) mirrors the relator class from OntoUML. The *Birth of person* concept is from the Civil law vocabulary. Figure 5 show the model from these examples as presented

in OntoGrapher in Compact mode (more on this in section 4). The concepts are color-coded based on their vocabulary.

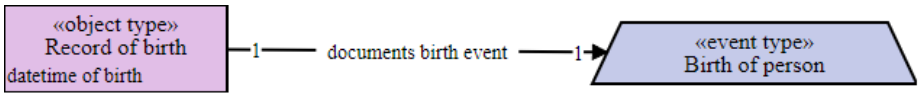

Fig. 5. Model from examples 2, 3, 4 in OntoGrapher's Compact view.

Note that the design of these vocabularies predate OntoGrapher entirely and the application's development does not influence the core nature of the vocabularies' architecture in any way; it is merely adapted to it. More details about the structure and the reasoning behind choosing it are described in a dedicated paper [9].

References

1. Shapes constraint language (SHACL). Tech. rep., W3C (Jul 2017), https://www.w3.org/TR/shacl/
2. Affairs, A.S.f.P.: System usability scale (sus) (Sep 2013), https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html
3. Barcelos, P.P.F., dos Santos, V.A., Silva, F.B., Monteiro, M.E., Garcia, A.S.: An automated transformation from ontouml to owl and swrl. In: Bax, M.P., Almeida, M.B., Wassermann, R. (eds.) ONTOBRAS. CEUR Workshop Proceedings, vol. 1041, pp. 130–141. CEUR-WS.org (2013), http://dblp.uni-trier.de/db/conf/ontobras/ontobras2013.html#BarcelosSSMG13
4. Chen, P.P.: The entity-relationship model - toward a unified view of data. ACM Trans. Database Syst. **1**(1), 9–36 (1976), http://dblp.uni-trier.de/db/journals/tods/tods1.html#Chen76
5. Group, T.: ArchiMate® 3.0 Specification. Van Haren Publishing (2016), https://books.google.cz/books?id=SmxpDAAAQBAJ
6. Guerson, J., Sales, T.P., Guizzardi, G., Almeida, J.P.A.: Ontouml lightweight editor: A model-based environment to build, evaluate and implement reference ontologies. 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop pp. 144–147 (2015)
7. Guizzardi, G.: Ontological foundations for structural conceptual models. Ph.D. thesis, University of Twente (Oct 2005)
8. Guizzardi, G., Fonseca, C.M., Benevides, A.B., Almeida, J.P.A., Porello, D., Sales, T.P.: Endurant types in ontology-driven conceptual modeling: Towards ontouml 2.0. In: Trujillo, J., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M. (eds.) Conceptual Modeling - 37th International Conference, ER 2018, Xi'an, China, October 22-25, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11157, pp. 136–150. Springer (2018). https://doi.org/10.1007/978-3-030-00847-5_12, https://doi.org/10.1007/978-3-030-00847-5_12

9. Kremen, P., Necasky, M.: Improving discoverability of open government data with rich metadata descriptions using semantic government vocabulary. https://doi.org/10.2139/ssrn.3303148, https://papers.ssrn.com/abstract=3303148
10. Miles, A., Bechhofer, S.: SKOS simple knowledge organization system reference. W3C recommendation, W3C (Aug 2009), https://www.w3.org/TR/2009/REC-skos-reference-20090818/
11. Moreira, J.L.R., Sales, T.P., Guerson, J., Braga, B.F.B., Brasileiro, F., Sobral, V.: Menthor editor: An ontology-driven conceptual modeling platform. In: JOWO@FOIS (2016)
12. Motik, B., Parsia, B., Patel-Schneider, P.F.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. W3c recommendation, W3C (oct 2009), http://www.w3.org/TR/2009/REC-owl2-syntax-20091027
13. OMG: OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1 (August 2011), http://www.omg.org/spec/UML/2.4.1
14. OMG: About the Object Constraint Language Specification Version 2.4 (February 2014), https://www.omg.org/spec/OCL/2.4/