Describing Reasoning Results with RVO, the Reasoning Violations Ontology

Bojan Božić, Rob Brennan, Kevin C. Feeney, and Gavin Mendel-Gleason

Knowledge and Data Engineering Group,
Trinity College Dublin, College Green, Dublin 2, Ireland
{bojan.bozic,rob.brennan,kevin.feeney,mendelgg}@scss.tcd.ie

Abstract. This paper presents a new OWL RL ontology, the Reasoning Violations Ontology (RVO), which describes both ABox and TBox reasoning errors produced by DL reasoners. This is to facilitate the integration of reasoners into data engineering tool-chains. The ontology covers violations of OWL 2 direct semantics and syntax detected on both the schema and instance level over the full range of OWL 2 and RDFS language constructs. Thus it is useful for reporting results to other tools when a reasoner is applied to linked data, RDFS vocabularies or OWL ontologies, for example for quality evaluations such as consistency, completeness or integrity. RVO supports supervised or semi-supervised error localisation and repair by defining properties that both identify the statement or triple where a violation is detected, and by providing context information on the violation which may help the repair process. In a case study we show how the ontology can be used by a reasoner and a supervised repair process to accelerate high quality ontology development and provide automated constraint checking feedback on instance data. RVO is also being used to enable integration of reasoning results into multi-vendor data quality tool chains within the ALIGNED H2020 project.

Keywords: Ontology Engineering, Data Integrity, Consistency Checking, Reasoning

1 Introduction

A common task performed with Semantic Web reasoners is the detection and reporting of errors or inconsistencies found in an ontology. This task frequently occurs within the ontology authoring, interlinking, classification, quality analysis and evolution phases of the linked data lifecycle [1]. However, to manage the entire data lifecycle or even the full range of activities within a single lifecycle stage, typically requires many tools to be integrated into a tool-chain. Current research on standard mechanisms for linked data tool-chain integration is still immature. However, the use of ontologies to support the interchange of data for tool-chain integration has been explored [5]. The existing gap in related work, as we show in Section 2 is the absence of a complete ontology for modelling reasoning errors. This would enable the integration of standardised ontology violation

detection services into linked-data management tool-chains. Other relevant work in the field focuses on pitfalls, prevention of mistakes and general ontology design and structural issues. The two goals that have driven this research are firstly, to produce a service which will assist in high-quality ontology development by identifying reasoning violations and producing semi-automated repair recipes. Secondly, to produce a service which can detect constraint violations on instance data according to a schema (ontology), and produce semi-automated repair recipes. In order to support these services, we need a rich, highly structured, general purpose way of expressing reasoning violations. This paper presents the RVO (Reasoning Violations Ontology). It describes OWL and RDF(S) reasoning errors, in two categories: those involving classes, properties and axioms (schema / TBox) and those involving instances (ABox). This ontology is used by a custom reasoner implemented in SWI-Prolog, the Dacura Quality Service [7], to report the errors that it detects to other linked data lifecycle tools. Our first client application is the Dacura Schema Manager [3], which can consume RVO and presents the reasoner output with filtered, categorised, detailed error information, as well as information about the source of the error. The principal contributions of this paper are: a description of the published RVO ontology, a discussion of the integration of RVO in our own data lifecycle web platform Dacura and documentation of the violation identification process for ontology developers. The rest of the paper is structured as follows: Section 2 is dedicated to related work. Section 3 presents the RVO ontology and provides insights into its design. Section 4 validates the ontology in a toolchain integration case. Finally, Section 5 concludes the paper and provides an answer to the research question as well as an outlook on future work.

2 Related Work

To the best of our knowledge, there are no ontologies that have been developed for the specific purpose of describing reasoning violations. However, similar problems have been addressed from differing perspectives. One of the most relevant contributions is OOPS! [10] which is a tool with a catalogue for validating ontologies by spotting common pitfalls. The catalogue contains 41 pitfalls which the tool checks for. The ontology can be inserted directly in a textbox or referenced by URI. Although, OOPS! identifies many common pitfalls, it detects design flaws rather than logical errors and does not use an ontology for error reporting. Other research [4] has identified the types of flaws that can occur in the object property box and proposed corresponding compatibility services. However, this work is very specific and focuses on properties and their compatibility. Our approach addresses a far broader palette of violations, across the ABox and TBox, incorporating class and property violations. In [12], a very similar approach to OOPS! was proposed, covering logical and non-logical anti-patterns, but it is quite limited as it covers only 4 logical and 3 non-logical anti-patterns as well as 4 guidelines. The work presented in [6] will be combined with our Reasoning Violations Ontology in order to extend Linked Data Quality in an ALIGNED

project use case. We have also published differences between SHACL and RVO in the deliverable [2] The Shapes Constraint Language (SHACL) introduced in [13], is a language for describing and constraining the contents of RDF graphs. As part of its ongoing development, through the W3Cs RDF Data Shapes Working Group, it is defining a standard error reporting format. Our ontology can be considered as an extension of SHACLs error reporting, as it can express a superset of the violations that can be expressed in SHACL, also while SHACL detects bad triples which caused an error, RVO is able to detect a whole subgraph which was involved in producing the violation. We plan to link RVO to SHACL errors through reuse of their predicates once the format achieves standardisation and stability. Another W3C vocabulary is EARL[1] which can be used for validation results. Although it has been defined in the context of validating accessibility tools, it contains several terms for describing validation results in RDF. There are also some publications about preventing errors in ontology development, such as [11]. They are extremely useful for defining best practices and fueling the discussion about ontology engineering style and error prevention, but they provide no insights into error reporting for existing ontologies. Closer to that is a publication about debugging OWL ontologies [8]. They have integrated a number of simple debugging cues generated from the description logic reasoner, Pellet[2], in the hypertextual ontology development environment, Swoop[3].

3 The Reasoning Violations Ontology (RVO)

The purpose of RVO is to enable a reasoner to describe reasoning errors detected in an input ontology in order to facilitate the integration of reasoners into semantic web toolchains. It is defined as a simple OWL 2 ontology that is amenable to RDFS-based interpretations or use as a linked data vocabulary without any dependence on reasoning. In future, an RDFS version of the ontology is planned, in order to support interpretation by RDFS reasoners. A permanent identifier for the ontology has been registered with the W3C permanent identifier community group. The full source of the ontology is published online[4] and meta-data have been added to facilitate LODE-based [9] documentation generation[5]. This ontology is used to describe RDF and OWL reasoning violation messages in the Dacura Quality Service [7]. These are generated by running an RDF/RDFS/OWL-DL reasoner over an RDF-based ontology model and allowing the Dacura quality service to report any integrity violations detected at schema or instance level. These violations report areas where the input model is logically inconsistent or breaks RDFS/OWL semantics or axioms. Violations may be reported as based on open world or closed world assumptions. The open world is the default OWL semantics and can typically only detect

[1] https://www.w3.org/TR/EARL10-Schema/
[2] https://github.com/complexible/pellet
[3] https://github.com/ronwalf/swoop
[4] https://w3id.org/rvo
[5] http://www.essepuntato.it/lode/closure/reasoner/https://w3id.org/rvo

a limited number of problems due to incomplete knowledge. The closed world interpretation assumes that you have provided all relevant aspects of the model and is able to detect a much wider range of violations, e.g. missing or misspelled term definitions. This is often useful during ontology development or in a system that interprets OWL as a constraint language.

3.1 The Ontology

The ontology can be divided into two layers. The top layer consists of the base classes and their properties and the bottom layer is a vocabulary which defines the hierarchical structure of violations identified so far (see 3.2).

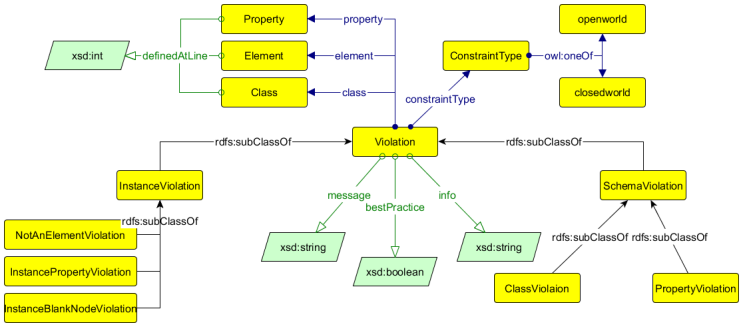

Fig. 1. Base Classes and Properties in RVO.

Figure 1 shows the top tier of the ontology which represents general metadata about a Violation as well as related properties, elements and classes. Class violations are used for reporting issues regarding the TBox and instance violations ABox in general. Therefore, class violations are reported when e.g. property domains are missing, subsumption errors are detected, or class and property cycles are found. Instance violations show instances which are not elements of valid classes, cardinalities which are incorrect, property constraints that are violated, literals and objects which are confused, etc.

3.2 Error Classes

We have organised reasoning errors in violation classes and put them in a hierarchical structure. The top level differentiation of violations is instance or schema, depending on whether the violation occurred in the ABox (instances) or TBox (schema). Here is an overview of all errors currently documented:

- **Instance**
  - InstanceBlankNode

  - NotAnElement: NotRestrictionElement, ObjectInvalidAtClass, EdgeOrphanInstance, DataInvalidAtDatatype (NotBaseTypeElement)
  - InstanceProperty: NotPropertyDomain, InvalidEdge, NotFunctionalProperty, LocalOrphanProperty, NotInverseFunctionalProperty
- **Schema**
  - ClassViolation: NotUniqueClassLabel, NotUniqueClassName, NotDomainClass, ClassCycle, NoImmediateClass (NotSuperClassOfClass), OrphanClass (NotIntersectionOfClass, NoSubclassOfClass, NotUnionOfClass)
  - PropertyViolation: PropertyCycle, NotUniquePropertyName, SchemaBlankNode, PropertyTypeOverload, PropertyAnnotationOverload, OrphanProperty (NotSubpropertyOfProperty), PropertyDomain (InvalidDomain, DomainNotSubsumed, NoExplicitDomain), PropertyRange (InvalidRange, RangeNotSubsumed, NotExplicitRange)

3.3 Example of RVO in Use - Class Violation

In our example a reasoning error is asserted first in JSON as raw data and then converted to RDF triples using RVO in order to be consumed in Dacura. The example shows a ClassViolation which is a SchemaViolation and more specifically a ClassCycleViolation. Such specific violation detection results make it possible to provide exact suggestions to ontology developers or repair agents and trigger ontology improvements.

```
{ "rdf:type": "ClassCycleViolation",
 "bestPractice": { "type": "xsd:boolean",
  "data": "false"},
 "message": "Class UnitOfSocialOrganisation has a class cycle with path: [
     TemporalEntity, UnitOfSocialOrganisation]",
 "path": [
  "seshat:TemporalEntity",
  "dacura:UnitOfSocialOrganisation"
 ],
 "class": "seshat:UnitOfSocialOrganisation"
}
```

The response RDF graph provides a much better way to interpret the results: The instance is called _example1 and is a ClassCycleViolation. bestpractice is false, so it is an error rather than a warning. The message provides a summary of the cause of the violation, but the important parts are the next two properties. The path property marks all classes which were involved in the cycle and the class property marks the class where the cycle has been detected. Another important feature is that RVO provides us direct links to the Seshat ontology (an ontology which models the Seshat Global History Databank[6]) and hence to the OWL classes from the external ontology which were involved in the violation process.

4 Validation through Integration into ALIGNED

The Dacura Toolchain Case Study covers toolchaining and reporting to users. At this point we want to have a closer look into supervised ontology repair and publishing.

[6] https://evolution-institute.org/project/seshat/

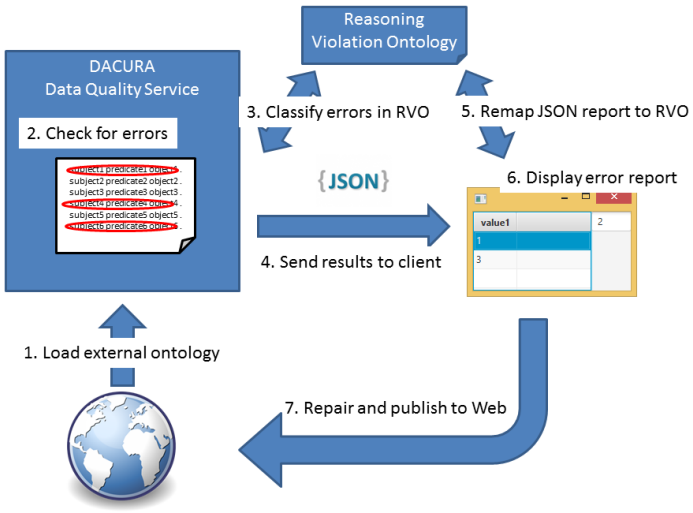

Fig. 2. Dacura Ontology Repair Use Case

In the use case scenario in Figure 2, the Dacura Shema Service gets an external ontology which is loaded by the Dacura Schema Manager and needs to be validated into the knowledge base (step 1). The ontology is then send to DQS and checked for errors by the DQS reasoner (step 2), whose rules comply with the violation classes of RVO. The detected violations are sent back to the client (step 3). The results are classified and assigned classes from RVO (step 4). Dacura then integrates the report in the UI and presents it to the user (step 5). Finally, the user can repair the ontology in an editor and republish it to the Web (step 6).

So how did RVO help in this specific scenario? After the Data Quality Service checked the external ontology for errors, it used the RVO structure to provide information about a specific violation by creating individuals. This provided us with a classification of DQS' results. RVO has been used by the client for classification of the errors and for providing relevant information about violations to the user. An additional benefit would be to archive the history of errors in a knowledge base and be able to query for certain occurrences of violations for an ontology.

Table1 shows the validation of ALIGNED ontologies[7] which have all been developed by using different approaches (Protégé, RDF2RDF, human checks, etc.). We have used the Data Quality Service and RVO to validate several project ontologies and report the errors and warnings found in a first run. Although, this is only a case study and especially the validation and correction efforts

[7] `http://aligned-project.eu/data-and-models/`

Table 1. Ontology validation results.

	DIO	DLO	SLO	SIP	EIPDM	SDO	DIOPP
# of triples	263	127	56	126	147	102	145
# of errors	0	1	25	10	37	6	4
# of warnings	4	11	5	5	5	7	11
h for validation	1	1	1	1	1	1	1
h for correction	1	2	4	2	4	2	2

are estimated, the table signalises the potential for improvement of existing ontologies.

5 Conclusion and Future Work

In this paper we have shown that a dedicated reasoning error ontology improves error reporting with structured data, and integration of the ontology in the Dacura toolchain case study. The Reasoning Violations Ontology not only benefits the interpretation and further processing of reasoning errors in tools, platforms, and Web UIs which present results of the reasoning process or ontology validation, but can also be used as a common format to represent violations found in ontologies during the whole software toolchain process. We have shown an example of supervised ontology repair use case and explained the advantages of our approach. Furthermore, we have given some examples for reasoning violations and constructed RDF graphs to present the results. Our future work will continue with the integration of the ontology in the ALIGNED toolchain and linking of the ontology to SHACL constraints as well as using it together with RDFUnit. Finally, we plan to evaluate the benefits in a case study with ontology engineers and investigate their work with the ontology and our tools in order to improve the quality of their ontology or repair it.

Acknowledgement

This research has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 644055, the ALIGNED project (www.aligned-project.eu) and from the ADAPT Centre for Digital Content Technology, funded under the SFI Research Centres Programme (Grant 13/RC/2106) and co-funded by the European Regional Development Fund. References

References

1. Sören Auer, Lorenz Bühmann, Christian Dirschl, Orri Erling, Michael Hausenblas, Robert Isele, Jens Lehmann, Michael Martin, Pablo N Mendes, Bert Van Nuffelen, et al. Managing the life-cycle of linked data with the lod2 stack. In *The Semantic Web–ISWC 2012*, pages 1–16. Springer, 2012.

2. Rob Brennan, Bojan Božić, Monika Solanki, Dimitris Kontokostas, Andreas Koller, and Christian Dirsch. D2.7 meta-model phase 2. 2016.
3. Kevin Feeney, Gavin Mendel-Gleason, and Rob Brennan. Linked data schemata: fixing unsound foundations (submitted). *Semantic Web Journal - Special Issue on Quality Management of Semantic Web Assets*, 2015.
4. C Maria Keet. Detecting and revising flaws in owl object property expressions. In *Knowledge Engineering and Knowledge Management*, pages 252–266. Springer, 2012.
5. Carsten Keßler, Mathieu d'Aquin, and Stefan Dietze. Linked data for science and education. *Semantic Web*, 4(1):1–2, 2013.
6. Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd international conference on World Wide Web*, pages 747–758. ACM, 2014.
7. Gavin Mendel-Gleason, Kevin Feeney, and Rob Brennan. Ontology consistency and instance checking for real world linked data. In Anisa Rula, Amrapali Zaveri, Magnus Knuth, and Dimitris Kontokostas, editors, *LDQ@ESWC*, volume 1376 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
8. Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging owl ontologies. In *Proceedings of the 14th international conference on World Wide Web*, pages 633–640. ACM, 2005.
9. Silvio Peroni, David Shotton, and Fabio Vitali. Tools for the automatic generation of ontology documentation: A task-based evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 9(1):21–44, 2013.
10. María Poveda-Villalón, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. Validating ontologies with oops! In *Knowledge Engineering and Knowledge Management*, pages 267–281. Springer, 2012.
11. Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. Owl pizzas: Practical experience of teaching owl-dl: Common errors & common patterns. In *Engineering Knowledge in the Age of the Semantic Web*, pages 63–81. Springer, 2004.
12. Catherine Roussey, Oscar Corcho, and Luis Manuel Vilches-Blázquez. A catalogue of owl ontology antipatterns. In *Proceedings of the fifth international conference on Knowledge capture*, pages 205–206. ACM, 2009.
13. Arthur Ryman. Z specification for the w3c editor's draft core shacl semantics. *arXiv preprint arXiv:1511.00384*, 2015.