Towards a Uniform User Interface for Editing Data Shapes

Ben De Meester, Pieter Heyvaert, Anastasia Dimou, and Ruben Verborgh

IDLab, Department of Electronics and Information Systems, Ghent University – imec
{ben.demeester,pheyvaer.heyvaert,anastasia.dimou,ruben.verborgh}@ugent.be

Abstract. Data quality is an important factor for the success of the envisaged Semantic Web. As machines are inherently intolerant at the interpretation of unexpected input, low quality data produces low quality results. Recently, constraint languages such as SHACL were proposed to assess the quality of data graphs, decoupled from the use case and the implementation. However, these constraint languages were designed with machine-processability in mind. Defining data shapes requires knowledge of the language's syntax – usually RDF – and specification, which is not straightforward for domain experts, as they are not Semantic Web specialists. The notion of constraint languages is very recent: the W3C Recommendation for SHACL was finalized in 2017. Thus, user interfaces that enable domain experts to intuitively define such data shapes are not thoroughly investigated yet. In this paper, we present a non-exhaustive list of desired features to be supported by a user interface for editing data shapes. These features are applied to *unSHACLed*: a prototype interface with SHACL as its underlying constraint language. For specifying the features, we aligned existing work of ontology editing and linked data generation rule editing with data shape editing, and applied them using a drag-and-drop interface that combines data graph and data shape editing. This work can thus serve as a starting point for data shape editing interfaces.

Keywords: Data Graph Validation, Data Shapes, SHACL

1 Introduction

High quality Linked Data is an important factor for the success of the envisaged Semantic Web. As machines are inherently intolerant at the interpretation of unexpected input, low quality data produces low quality results. In the context of Linked Data, data quality can be measured by assessing the *data graph* – represented using the Resource Description Framework (RDF) [1] – against a set of *constraints* [2]. This set of constraints forms the *data shape*. When it is assessed that the data graph does not adhere to the data shape, a set of *constraint violations* is returned. These constraint violations in turn can guide a user how to refine the data graph, and thus, improve its quality. For example, for a certain use case, an important set of constraints of the data graph is that every person has a first name, last name,

and birth date specified. When a data graph describing people that contains a person without first name is assessed, a constraint violation is returned pointing to that person. By adding the first name, the data graph is of higher quality for that use case.

Ease of editing of the data shape is important. As quality is commonly defined as "fitness for use" [3], data shapes can be different for different use cases; thus, data shapes need to be easily editable. Hard-coded validation approaches – as opposed to declarative approaches – are not easily editable, as each change of the data shape would require a new development cycle of the software tool [4]. Recently, *constraint languages* were proposed as a declarative means to define data shapes: this eases editing as it decouples the definition of the data shapes from the implementations that validate data graphs. Of these constraint languages, the Shapes Constraints Language (SHACL) became a W3C Recommendation in 2017 [5]. Constraint languages allow domain experts – that are familiar with the use case at hand – to define the data shapes, and modify these at any time, without making changes in the implementation of the validation tool.

However, constraint languages were designed with machine-processability in mind. Similarly to manually editing other declarative, machine-processable rules – e.g., rules for linked data generation using the RDB to RDF Mapping Language (R2RML) [6] –, manually editing data shapes using declarative constraint languages requires a substantial amount of human effort [7]. When data shapes cannot be created and edited easily, uptake of using data shapes and thus improving the quality of data graphs might be inhibited: intuitive user interfaces are needed. As constraint languages for describing Linked Data quality are only recently being standardized, features for user interfaces that enable domain experts to intuitively define such data shapes, are not thoroughly investigated yet.

In this paper, we introduce a non-exhaustive list of desired features for a user interface for editing data shapes. They are addressed by a data shape editor, which aims to support domain experts to improve Linked Data quality. After discussing the related work at Section 2, we list our features at Section 3. We show how these features were applied to the *unSHACLed* editor at Section 4. Finally, we conclude at Section 5.

2 Related Work

After discussing how data shapes are commonly defined (Subsection 2.1), we discuss Semantic Web editors and their interfaces (Subsection 2.2). We close at Subsection 2.3.

2.1 Defining Data Shapes

Firstly, ontologies were interpreted as integrity constraints [8, 9]. However, as the semantics of ontologies operate under the Open World Assumption, and the semantics of data shapes under the Closed World Assumption, this requires an alternative interpretation of the existing semantics of ontology languages [10], e.g., the Web Ontology Language (OWL) [11]. Then, query-based approaches, such as

RDFUnit [2], emerged where data shapes are defined using SPARQL queries. Finally, more generic constraint languages were defined, such as Shape Expressions (ShEx) [12] and SHACL [5]. These constraint languages provide a declarative approach for defining data shapes without creating ambiguous semantics for ontologies. There exist more than eighty different types of constraints [13], where each constraint language supports a subset of these *constraint types* [13].

2.2 Editors

No extensive research was conducted into features for editing data shapes so far, given the very recent standardization of constraint languages for Linked Data. However, we argue that existing work into visualizing other types of RDF structures is also relevant, namely, (i) ontology editors, and (ii) editors for Linked Data generation rules. On the one hand, we can take ontology editors into account, similar as to related work where ontologies are interpreted as integrity constraints. On the other hand, editors for Linked Data generation rules are also related to data shape editors: both define the shape of the data graph; however, they have different directions [14]. For Linked Data generation rules, the data graph is the *target*: the generation rules *define* the data graph. For constraint languages, the data graph is the *source*: the constraints *limit* the data graph. Thus, besides only discussing the existing work around constraint editors, we also discuss ontology editors and Linked Data generation rules editors.

Data Shape Editors *Textual* data shape editors such as SHACL4P [15] – a textual editor as plug-in for Protégé – are not intuitive for editing data shapes [16]. Defining data shapes requires knowledge of the syntax – usually RDF – and the specification of the constraint language. This is not straightforward for domain experts as they are not necessarily Semantic Web specialists. Other data shape editors such as the work of Şimşek et al. for validating Schema.org annotations [17], are typically *form-based* or *use-case specific*. Form-based approaches enforce a linear workflow and thus hinder the user to keep an overview of the relation between the data graph and data shapes during editing. Use-case specific approaches are not reusable and depend on a specific constraint language.

Ontology Editors Visual ontology editors have largely been divided into three groups: *graph-based*, *indented-tree-based* [18], and *UML-based*. Graph-based visual notations and implementations for OWL include Graffoo [19] and AVOnEd [20]. Indented tree-based editors include the Protégé OWL plugin [21], the editor of the NeOn Toolkit [22], Eddy [23] with Graphol [24] as visual notation, and TopBraid Enterprise Vocabulary Net. In general, the indented-tree-based visualization is perceived more organized and familiar to novice users, whilst the graph-based visualization is perceived to be more controllable and intuitive without visual redundancy, particularly for ontologies with multiple inheritance [18]. UML-based editors include OWLGrEd [25], Menthor that uses OntoUML [26] as visual notation that can be seen as an abstraction over, a.o., OWL, and the work by De Paepe et al. [27].

Linked Data Generation Rule Editors

Editors for Linked Data generation rules can largely be divided into two groups: *form-based*, and *graph-based*. An alternative graphical user interface using a block metaphor to edit Linked Data generation rules is Juma [28]. Form-based approaches [7, 29] – just as for data shape editors – enforce a linear workflow and hinder keeping an overview during editing. Graph-based approaches includes SQuaRE [30], Map-On [31], and the RMLEditor [32]. Heyvaert et al. proposed a set of desired features for linked data generation rules editors [33], namely: (R1) *independent of the underlying linked data generation rules language*, (R2) *support multiple data sources*, (R3) *support heterogeneous data formats*, (R4) *support multiple ontologies*, (R5) *support multiple alternative modeling approaches*, (R6) *support non-linear workflows*, and (R7) *independent of mapping execution*. These features helped guide the interface of the RMLEditor, which was evaluated to be better than related editors during a user study [32].

2.3 Discussion

An intuitive and use-case independent solution is lacking: current data shape editing approaches either depend on the constraint language or enforce a linear workflow. This leads to two features: *Independence of constraint language* and *Non-linear workflows*. For applying these features to a visual editor, we can observe that on the one hand, constraint languages have following overlapping semantic constructs: (i) the data graph, (ii) the data shape, and (iii) the returning constraint violations (when available). On the other hand, graph-based visualizations are the most widely used across all editing approaches.

3 Desired Features for Data Shape Editing

In this section, we list the desired features for data shape editors. We extend the features extracted from the related work by adapting the features for linked data generation rules editors as proposed by Heyvaert et al. [33]. We adapt these features given previously researched parallels between ontologies and data shapes on the one hand, and between ontologies and linked data generation rules on the other hand.

F1. Independence of constraint language Data shape editors should not confront domain experts with writing the textual syntax of a specific constraint language: domain experts are not necessarily Semantic Web experts, and constraint languages are built with machine-processability in mind. For instance, writing Turtle files to describe data shapes would be a laborious process for domain experts. Moreover, the visualization of the constraints should be independent of the underlying constraint language: generic (graphical) symbols can be used to (partially) hide language-specific textual syntax, as constraint languages have overlapping semantic constructs.

F2. Support multiple data sources Data shape editors should support domain experts in defining data shapes referring to multiple data sources at the same time. Since Linked Data is commonly used as an integration approach, the data that is needed for a specific use case might originate from multiple data sources. For

instance, two data graphs contain information about employees and projects, respectively. Defining the data shape of the connection between employees and their corresponding projects is only possible when definitions can be specified across the two data sources.

F3. Support different serializations Data shape editors should not restrict domain experts to specific serializations of the data source nor the constraint language. A data graph can be serialized in different ways without changing the actual data or structure (e.g., RDF/XML vs Turtle). Moreover, a data shape constraints the data *model* and not the *format*. This is adapted from Heyvaert's original definition, as the scope of our work is Linked Data *validation*. As opposed to Linked Data *generation*, no different data formats need to be supported.

F4. Support multiple ontologies Data shape editors should support domain experts in defining data shapes for data graphs annotated with multiple ontologies at the same time. Complementary or overlapping ontologies are available, and different use cases require different ontologies. For instance, a user's friends are described with foaf, and his interests with schema.org. Depending on the use case, data shapes are needed for only the foaf-annotations, only the schema.org-annotations, or the combination of the two.

F5. Multiple alternative modeling approaches Data shape editors should enable and support multiple alternative modeling approaches and allow domain experts to choose the most adequate one for their needs. Two modeling approaches are put forward, given that a validation assessment is related to both the data graph and the data shape. Either (i) define a data shape based on a predetermined model of a use case, and afterwards validate a data graph (*model-driven*, e.g., a description is given that "every user needs a birthdate"), or (ii) define a data shape corresponding to a given a data graph, (*data-driven*, e.g., an example is given of a valid data graph describing a user). This requirement is adapted from Pinkel et al. [7], given the parallels between ontologies and linked data generation rules.

F6. Non-linear workflows Data shape editors should allows domain experts to keep an overview of the relationship between the data graph and data shapes, by providing non-linear editing. Defining data shapes involves multiple factors – the use case at hand, data graphs, and data shapes – that have an influence on each other. A linear workflow separates these factors and obscures their relationships, by using ordered steps that need to be followed by the domain experts. For instance, when adding constraints to the data shape, domain experts might find that either the data graph has violations, or the data shape is wrongly defined. The non-linear workflow allows to integrate information about the data graph into the data shape without the need to redo (parts of) the definition process.

F7. Independence of execution Data shape editors should allow importing and exporting the data shapes specified by the domain experts, as editing data shapes lies beyond the scope of their execution. Further processing or execution – outside the data shape editor – improves the data shapes' interoperability and reusability. For instance, when the data shapes are created locally using an editor, they can be executed on a server without further need of the editor.

4 Application to unSHACLed

In this section, we show the feasibility of supporting all proposed features by applying them to *unSHACLed*, which we introduce in Subsection 4.1. Then, we elaborate on how each desired feature is supported by unSHACLed in Subsection 4.2. A screencast demonstrating unSHACLed can be found at https://w3id.org/imec/unshacled/screencast/voila2018, a working demo is available at https://w3id.org/imec/unshacled/app, and the source code can be found on https://github.com/dubious-developments/UnSHACLed.

4.1 unSHACLed

We present unSHACLED: a prototype interface for a visual data shape editor configured to use SHACL as underlying constraint language. In unSHACLed, domain experts are able to drag-and-drop loaded data graphs, loaded data shapes, and add data shapes using templates. They can edit both the data graphs and data shapes based on visual feedback on whether the loaded data graphs conform with the data shapes, and export the resulting data shape. Its User Interface (UI) consists of three elements (see Fig. 1): an *Overview Sidebar* (Fig. 1, a), an *Action Toolbar* (Fig. 1, b), and an *Editing Area* (Fig. 1, c).

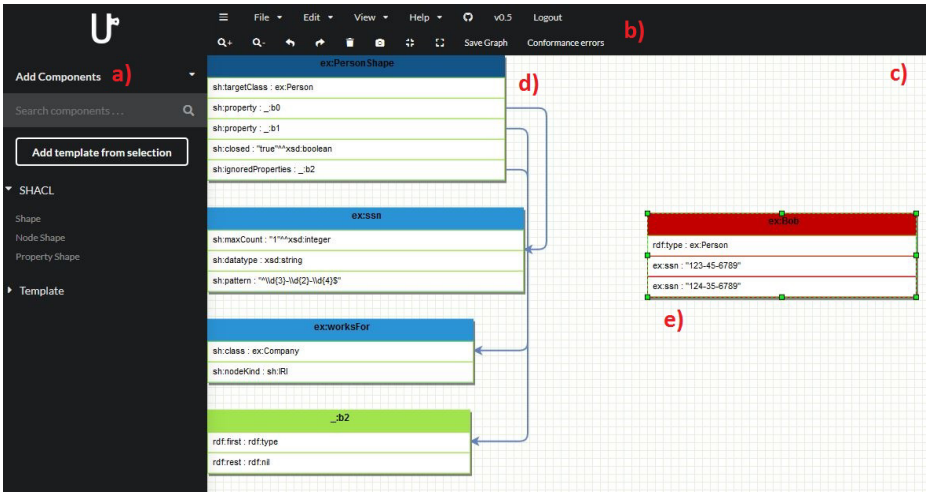

Fig. 1: The unSHACLED UI, consisting of an *Overview Sidebar* (left), an *Action Toolbar* (top), and an *Editing Area* (middle-right)

Overview Sidebar The *Overview Sidebar* shows a set of templates for data shapes. Templates can be added: certain structures are more likely to occur, depending on the context in which the editor is used. As unSHACLed is configured for editing data shapes using SHACL, the default templates are currently *Node shapes* or *Property shapes* (constraining resources or the relations(s) between resources, respectively).

Action Toolbar The *Action Toolbar* consists of two rows: general application actions, and specific editing area actions. General application actions include importing and exporting data graphs and data shapes (either from local or remote files, or synced with your GitHub account) into the current workspace – visible in the Editing Area –, and user management. Editing area actions include zooming in and out, under/redo, removing definitions, taking a screenshot, and executing the validation assessment.

Editing Area The *Editing Area* provides a visual drag-and-drop interface for managing, connecting, and editing data shapes (Fig. 1, d). We chose a graph-based interface, due to its widespread usage across editing approaches. Each resource within either a data graph or data shape is depicted using a diagrammatic symbol: a rectangle with colored header. There is a visual distinction between the data graph and the data shapes via different colors for the resource rectangle's header. For editing the individual constraint rules (Fig. 1, e), each rectangle contains an editable list of rules in the form of text fields.

Text fields are used, as there exist more than eighty different types of constraints [13], where each constraint language supports a subset of these constraint types. Of all visual expressiveness variables, only the shape variable would have a large enough capacity to distinctively depict all different constraint types [34]. However, we argue that distinguishing between these shapes and remembering which shape relates to which constraint type would put at least the same amount of cognitive burden, as textually entering the constraint rule. Thus, the textual interface for editing the constraint rules allows flexibility in terms of constraint languages, without overburdening the user, similar to the interface of OWLGrEd [25]. Furthermore, the actual RDF syntax is simplified: each text field represents a predicate-object pair, stemmed on the Predicate Lists expressions of Turtle, the human-readable RDF serialization format [35].

unSHACLed allows domain experts to execute the validation assessment on the loaded sample data graph. This is automatically performed during editing, or can be triggered manually via the Action Toolbar. The results are visualized in the Editing Area by differently coloring the resource rectangle's header. This allows to easily detect constraint violations during the editing process. Thus, immediate feedback is available, visualized on the data graph in the Editing Area.

4.2 Implementing the Desired Features

We evaluated the application of our approach within unSHACLed by using the objectives-based evaluation method [36], i.e., a questions-oriented evaluation approach that can be performed internally by program developers. The UI of unSHACLed implements the desired features for data shape editors which were presented in Section 3. Below, we elaborate on how each feature is facilitated. A summary of which features are supported by which UI element can be found in Table 1.

Features	**UI Element**		
	Overview Sidebar	Action Toolbar	Editing Area
F1. Independence of constraint language	✓		✓
F2. Support multiple data sources		✓	✓
F3. Support different serializations		✓	
F4. Support multiple ontologies		✓	
F5. Multiple alternative modeling approaches			✓
F6. Non-linear workflows	✓	✓	✓
F7. Independence of execution		✓	

Table 1: The features for a data shape editor, denoting which features each element of unSHACLed's UI supports.

F1. Independence of constraint language The Overview Sidebar can be filled with constraint language specific templates, however, the interface of the Editing Area remains independent of the constraint language. The overlapping semantic constructs of the constraint languages are supported by the interface: the data graph and data shape are similarly visualized via the connected rectangles, with either predicate-object pairs or a set of constraint rules; the constraint violations are visualized on the data graph; the constraint rules can be edited via the textual input fields. Moreover, the actual validation engine is modularized, i.e., other validation systems can be plugged in to support different constraint languages.

F2. Support multiple data sources The Action Toolbar allows importing multiple data graphs and data shapes: all are loaded and visualized in the Editing Area. Validation is executed on the combination of all loaded data graphs, based on the combination of all loaded data shapes.

F3. Support different serializations The Action Toolbar allows importing data graphs and shapes from Turtle and RDF/XML, and the import engine is extensible for other serializations. In the future, alternative serializations, such as the JSON serialization for ShEx, could thus be included.

F4. Support multiple ontologies The textual input fields of the Editing Area are not restricted to specific ontologies and/or vocabularies, and allow changes at any time during the editing process.

F5. Multiple alternative modeling approaches By combining the data graph and data shape in the Editing Area, domain experts can follow different approaches when editing data shapes, i.e., either the model-driven or data-driven approach, or they can employ a hybrid approach: given the immediate feedback of constraint violations in the loaded data graph, constraint violations can be addressed immediately.

F6. Non-linear workflows

Domain experts keep the overview of the editing process by combining the data graph and shape with the Editing Area, and showing all three UI elements simultaneously. The different constraint aspects can be edited and reviewed without the need for tracing back previously completed steps.

F7. Independence of execution Domain experts export data shapes as RDF statements at any point during the editing process, as facilitated by the Action Toolbar. Subsequently, the definitions can be edited or executed without unSHACLed.

5 Conclusion

This paper presents a non-exhaustive list of features for a visual data shape editor, and proves their feasibility by presenting how these features are applied to unSHACLed. The proposed seven desired features can guide the design and implementation of a uniform data shape editor. Ultimately, data shape editors that provide these features help domain experts to create data shapes without manually editing them, and without being restricted to the use case: unSHACLed does not require domain experts to manually write syntactically correct RDF triples, and is not limited to specific use cases.

This work is an initial step towards the novel field of data shape editing. First of all, an evaluation based on a user study is planned, to validate the suitability of the graphical representation of unSHACLed. Open issues include scalability: how the interface would handle large data shapes. We foresee the management of multiple data shapes (and accompanying data graph samples) by the use of multiple contexts within a workspace. Within a workspace, multiple data shapes can be loaded, without needing to visualize them simultaneously, and the user is able to switch between contexts to visualize different data shapes within one workspace. Moreover, we could apply different detail levels to provide interactive filtering.

Acknowledgements

The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), and the European Union. Ruben Verborgh is a postdoctoral fellow of the Research Foundation – Flanders.

References

1. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 Concepts and Abstract Syntax. World Wide Web Consortium (W3C), http://www.w3.org/TR/rdf11-concepts/ (2014).
2. Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., Zaveri, A.: Test-driven evaluation of linked data quality. In: Proceedings of the 23rd international conference on World Wide Web. pp. 747–757 (2014).
3. Juran, J.M.: Juran's Quality Control Handbook. Mcgraw-Hill, Texas, USA (1988).
4. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A Generic Language for Integrated RDF Mappings of

Heterogeneous Data. In: Proceedings of the 7th Workshop on Linked Data on the Web (2014).
5. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL). W3C, https://www.w3.org/TR/shacl/ (2017).
6. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. World Wide Web Consortium (W3C), http://www.w3.org/TR/r2rml/ (2012).
7. Pinkel, C., Binnig, C., Haase, P., Martin, C., Sengupta, K., Trame, J.: How to Best Find a Partner? An Evaluation of Editing Approaches to Construct R2RML Mappings. In: The Semantic Web: Trends and Challenges. pp. 675–690 (2014).
8. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. In: Proceedings of WWW 2007. pp. 74–89 (2009).
9. Tao, J., Sirin, E., Bao, J., McGuinness, D.L.: Integrity Constraints in OWL. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010). , Atlanta, Georgia, USA (2010).
10. Arndt, D., De Meester, B., Dimou, A., Verborgh, R., Mannens, E.: Using Rule Based Reasoning for RDF Validation. In: RuleML+RR (2017).
11. Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., Smith, M.: OWL 2 Web Ontology Language – Structural Specification and Functional-Style Syntax (Second Edition). World Wide Web Consortium (W3C), http://www.w3.org/TR/owl2-syntax/ (2012).
12. Prud'hommeaux, E., Labra Gayo, J.E., Solbrig, H.: Shape expressions: an RDF validation and transformation language. In: Proceedings of the 10th International Conference on Semantic Systems. pp. 32–40 (2014).
13. Bosch, T., Nolle, A., Acar, E., Eckert, K.: RDF Validation Requirements – Evaluation and Logical Underpinning. arXiv preprint arXiv:1501.03933. (2015).
14. Dimou, A., Heyvaert, P., De Meester, B., Verborgh, R.: What Factors Influence the Design of a Linked Data Generation Algorithm? In: Proceedings of the Workshop on Linked Data on the Web (LDOW 2018) (2018).
15. Ekaputra, F.J., Lin, X.: SHACL4P: SHACL constraints validation within Protégé ontology editor. In: 2016 International Conference on Data and Software Engineering (ICoDSE) (2016).
16. Dadzie, A.-S., Rowe, M.: Approaches to visualising linked data: A survey. Semantic Web. 2, 89–124 (2011).
17. Şimşek, U., Kärle, E., Holzknecht, O., Fensel, D.: Domain Specific Semantic Validation of Schema.org Annotations. In: Perspectives of System Informatics. PSI 2017. pp. 417–429 (2017).
18. Fu, B., Noy, N.F., Storey, M.-A.: Indented Tree or Graph? A Usability Study of Ontology Visualization Techniques in the Context of Class Mapping Evaluation. In: The Semantic Web – ISWC 2013. pp. 117–134 (2013).
19. Falco, R., Gangemi, A., Peroni, S., Shotton, D., Vitali, F.: Modelling OWL Ontologies with Graffoo. In: The Semantic Web: ESWC 2014 Satellite Events. pp. 320–325 (2014).
20. Hallay, F., Hartmann, S., Kewitz, N., Mertens, R.: An Aspect-Oriented Visual Ontology Editor with Edit-Time Consistency Checking. In: 2017 IEEE 11th International Conference on Semantic Computing (ICSC) (2017).

21. Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.A.: The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In: Proceedings of the 3rd International Semantic Web Conference (ISWC). pp. 229–243 (2004).
22. Erdmann, M., Waterfeld, W.: Overview of the NeOn Toolkit. In: Ontology Engineering in a Networked World. pp. 281–301. Springer Berlin Heidelberg (2011).
23. Lembo, D., Pantaleone, D., Santarelli, V., Savo, D.F.: Eddy: a graphical editor for OWL 2 ontologies. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence. pp. 4252–4253. , New York, New York, USA (2016).
24. Console, M., Lembo, D., Santarelli, V., Savo, D.F.: Graphol: Ontology Representation Through Diagrams. In: Informal Proceedings of the 27th International Workshop on Description Logics. pp. 483–495. , Vienna, Austria (2014).
25. Cerans, K., Ovcinnikova, J., Liepins, R., Sprogis, A.: Advanced OWL 2.0 Ontology Visualization in OWLGrEd. In: DB&IS (2012).
26. 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. In: 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop (2015).
27. De Paepe, D., Thijs, G., Buyle, R., Verborgh, R., Mannens, E.: Automated UML-Based Ontology Generation in OSLO2. In: The Semantic Web: ESWC 2017 Satellite Events – ESWC 2017. pp. 93–97 (2017).
28. Junior, C.A.C., Debruyne, C., O'Sullivan, D.: Juma: An Editor that Uses a Block Metaphor to Facilitate the Creation and Editing of R2RML Mappings. In: The Semantic Web: ESWC 2017 Satellite Events. pp. 87–92 (2017).
29. Sengupta, K., Haase, P., Schmidt, M., Hitzler, P.: Editing R2RML mappings made easy. In: Proceedings of the 12th International Semantic Web Conference (Posters & Demonstrations Track). pp. 101–104. , Sydney, Australia (2013).
30. Blinkiewicz, M., Bąk, J.: SQuaRE: A Visual Support for OBDA Approach. In: Semantic Technology – JIST 2016. pp. 47–55 (2016).
31. Sicilia, Á., Nemirovski, G., Nolle, A.: Map-On: A web-based editor for visual ontology mapping. Semantic Web Journal. 8, 969–980 (2017).
32. Heyvaert, P., Dimou, A., De Meester, B., Seymoens, T., Herregodts, A.-L., Verborgh, R., Schuurman, D., Mannens, E.: Specification and implementation of mapping rule visualization and editing: MapVOWL and the RMLEditor. Web Semantics: Science, Services and Agents on the World Wide Web. 49, 31–50 (2018).
33. Heyvaert, P., Dimou, A., Verborgh, R., Mannens, E., Van de Walle, R.: Towards a Uniform User Interface for Editing Mapping Definitions. In: Proceedings of the 4th Workshop on Intelligent Exploration of Semantic Data (2015).
34. Moody, D.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering. 35, 756–779 (2009).

35. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: Turtle - Terse RDF Triple Language. World Wide Web Consortium (W3C), http://www.w3.org/TR/turtle/ (2014).
36. Stufflebeam, D.: Evaluation Models. New Directions for Evaluation. 2001, 7–98 (2001).