ORM Diagram as an Intermediate Model for OWL Ontology Engineering: Protégé ORM Plugin Implementation

D V Litovkin\(^1\), A V Anikin\(^1\), O A Sychev\(^1\) and T Petrova\(^2\)

\(^1\) Volgograd State Technical University, 28, Lenina Avenue, Volgograd, 400005, Russia
\(^2\) Volgograd State Socio-Pedagogical University, 27, Lenina Avenue, Volgograd, 400066, Russia

E-mail: anton@anikin.name

Abstract. OWL2, a widely-used ontology-representation language, is poorly perceived by humans because OWL2 statements have a low level of abstraction. To solve this issue, various OWL2 ontology editors are used, which allow to group statements and represent them using some visual notation. ORM-diagram is a good candidate for an intermediate model for authoring and understanding of OWL2-ontology as Object-Role Modelling notation supports visually distinguishable constructs, has the high expressive capabilities, and implements the node-link paradigm and the attribute-free approach. A Protégé plugin allowing to create an ORM2-diagram using the live error checking approach was implemented. The plugin allows us to form a valid object-oriented diagram model in computer memory using a widely known ontology authoring tool Protégé.

1. Introduction

A conceptual model is a concise and precise consolidation of all goal-relevant structural and behavioural features of the SUI (system under investigation) presented in a predefined format [1].

In this research, we focus on structural conceptual models (also named domain models, information models, semantic data models). More specifically, this kind of model addresses issues such as: (i) general notions of types and their instances; (ii) distinctions among sorts of types (e.g., kinds, roles, phases, mixins) and their admissible relations; (iii) objects, their intrinsic properties (attributes and associations); (iv) distinctions among sorts of relational properties [2].

An explicit and formal representation of a conceptual model is termed an ontology. The ontology should be understandable for both humans and computers. Currently, OWL2 is the most widely used language to represent ontologies. However, it is poorly perceived by humans because OWL2 statements have a low level of abstraction so a large number of them are required to describe a domain. To solve this issue, various OWL2 ontology editors are used, which allow to group statements and represent them using some visual notation.

Since structural conceptual model by their nature have a graph structure (these are various objects and relations between them), node-link paradigm is a promising approach for ontologies visualisation and authoring [3, 4].

2. Visual tools for ontology editing
The first category of visual tools for ontology authoring consists of tools that display OWL2 statements without their transformation: Ontodia, WebVOWL / ProtegeVOWL, OWLGrEd Editor, and Eddy. The best expressive capabilities has Eddy tool [5] that implements Graphol notation. Its disadvantages include the complexity of Graphol-diagrams and its inability to construct diagrams out of existing ontologies: Eddy tool can only generate an OWL-ontology out of Graphol-diagram. OWLGrEd Editor [6] allows visualising an existing ontology and generating a new ontology using UML notation. However, some OWL2 statements cannot be represented in OWLGrEd Editor.

The second category consists of tools allowing to define high-level statements and transform them into an OWL2 ontology. The examples include Menthor Editor [7] and OntoUML Lightweight Editor. They implement an ontologically well-founded language for Ontology-driven Conceptual Modelling: OntoUML. OntoUML is designed to comply with Unified Foundational Ontology (UFO) so it provides expressive and precise constructs allowing ontology engineers to capture their domains of interest. However, OntoUML uses only three visual elements for the whole variety of supported statement types: class (attributes and operations sections are not used), association relation, and generalisation relation. Therefore, it is difficult to visually distinguish and recognise types, attributes and associations in OntoUML-diagrams.

In contrast to OntoUML, Object-Role Modelling notation supports visually distinguishable constructs, exceeds the expressive capabilities, and implements the attribute-free approach [8].

3. Using ORM for conceptual modelling and ontology engineering

ORM2 notation allows to visualise types, taxonomic relations, properties (attributes and associations (predicates)), and various constraints for taxonomic relations and relational properties [9]. So, ORM-diagram is a good candidate for an intermediate model for authoring and understanding of OWL2-ontology.

Previously existing DogmaModeler tool [10] was able to build an ORM2-diagram and map it to an OWL2-ontology. However, it is currently not available. A notable example of available tools implementing ORM2 notation is NORMA [11], an open-source plugin for Microsoft Visual Studio IDE. It allows building ORM2-diagrams with live error checking and generating DDL code to create the relational schema. However, it is not convenient for non-IT specialists because it requires installing the complex and cumbersome environment Microsoft Visual Studio.

4. Protégé ORM Plugin Implementation

The authors propose to create an authoring tool for building ORM2-diagrams and mapping them to OWL2-ontologies as Protégé plugin. This solution has several advantages. Protégé is a widely used, well-maintained, free ontology editor. It allows visualising ontologies with GUI, inferring statements with reasoners, displaying inferred statements, detecting contradictory statements, and saving ontologies in different notations. The Protégé plugin can get access to these features, and ontology engineers can use the ontology-modelling environment that is familiar to them. The current implementation of Protégé ORM plugin, implemented by the authors, allows creating ORM2-diagrams containing only basic elements of ORM2 notation. This subset of elements is used during the initial steps of Object-Role Modelling.

4.1. ORM2-diagram as graph

When constructing an ORM2-diagram, the following restrictions must additionally be met:

- all components $et_{name}$ from $et$ and $vt_{name}$ from $vt$ must be unique and not empty;
- for every $r$, there is only one edge $ra$ incident with it;
- for any $pr$, at least one $r$ from $pr$ must be incident with $et$;
- the component $r_{name}$ from $u_{pr}$ must not be empty;
- components $r_{name}$ from $b_{pr}$ must be unique; one of $r_{name}$ from $b_{pr}$ may be empty;
- deleting any node $et$, $vt$ or $pr$ causes deleting edges $ra$ or $si$ incident with it.


ORM2-diagram using only basic elements can be denoted as graph:

\[ ORM_2_G = \langle OT, PR, RA, ST \rangle, \]

where \( OT, PR, RA, ST \) are defined in Table 1.

### Table 1. ORM2-diagram elements.

| Elements | Visual representation |
|----------|-----------------------|
| \( OT \) – set of nodes denoting Object Types | ![Person](image) |
| \( ET \) – set of nodes denoting Entity Types | ![Gender](image) |
| \( VT \) – set of nodes denoting Value Types | ![is tenured](image) |
| \( PR \) - set of nodes denoting Predicates | ![worksFor/employs](image) |
| \( RA \) – set of association edges between nodes \( OT \) and \( R \), i.e. Role Association is | ![Person](image) ![Company](image) |
| \( ST \) – set of edges denoting Subtyping relationships, i.e. | ![Person](image) ![Male](image) |

4.2. **Protégé plugin structure**

Figure 1 presents the structure of Protégé plugin at the components level. The component ORM2_Diagram implements an object-oriented representation of ORM2-diagram in computer memory and encapsulates rules of creation ORM2-diagram as a graph. Every element of ORM2-diagram (see above) is implemented with a separate class (see figure 2, ORM2_Diagram package) that controls: a) the ability to create an element with specified components; b) the ability to link elements to each other; c) the presence of incidence between elements. These capabilities are used by the ORM_Diagram class (see figure 2, ORM2_Diagram package) for the correct creation, removal, and linking of diagram elements and also for the validation of the diagram as a whole.
Figure 1. The structure of Protégé plugin for ORM2-diagrams authoring at the components level.

Component JGraphX is a Java Swing Library version of mxGraph. mxGraph is a diagramming library that allows implementing interactive graphs. This library was chosen for the following reasons:

- it allows to create a graph interactively and programmatically;
- it uses Java Swing library, which is necessary for compatibility with Protégé;
- it allows you to create nodes of different shapes, sizes and styles;
- it allows you to create edges with different styles, including end decorations;
- it allows to set control points for edge. These are intermediate points along the edge that the edge is drawn as passing through. The use of control points is sometimes referred to as edge routing;
- it allows to validate the connection of edges to nodes;
- it is free;
- it is supported by developers (the last update released on June, 19 2020).

The ORM2_Presenter component is an intermediate layer between the ORM2_Diagram component and the JGraphX library. Each subclass of ORM_ElementPresenter class corresponds to a particular element of the ORM2-diagram and encapsulates the rules for its rendering using JGraphX library.

The mxGraph_Extension component contains the My_mxGraph and My_mxMultiplicity classes, subclassing mxGraph and mxMultiplicity classes from the JGraphX library. The My_mxGraph and My_mxMultiplicity classes are necessary for integration of the JGraphX library with the ORM2_Presenter component and indirectly with the ORM2_Diagram component.

The ORM2/owl2_Mapper component maps an ORM2-diagram to an OWL2-ontology and vice versa.

This structure of the Protégé plugin allows:

- to minimise linkage between components because each component is connected to no more than two other components, i.e. the structure is multilayered;
- encapsulating the rules for constructing ORM2-diagrams, the rules for rendering their elements using the JGraphX library, and the rules for mapping an ORM2-diagram in an OWL2-ontology (and vice versa) in separate components in order to modify the rule groups independently.

In addition, the developed plugin contains extension points for adding more ORM-elements. These points are the abstract classes ORM_Element and ORM_ElementPresenter.
4.3. Protégé plugin for ORM2 diagrams authoring usage examples

In the recent version of the plugin the components ORM2_Diagram, ORM2_Presenter and mxGraph_Extension are implemented. The plugin allows using Protégé for creating ORM2-diagram composed of the basic elements. Figure 3 shows an example of ORM2-diagram created in the developed plugin. Figures 4 - 7 show examples of validation of some of the constraints explained above.

Figure 2. The class diagram of Protégé plugin for ORM2-diagrams authoring.

Figure 3. An example of ORM2-diagram created with Protégé ORM plugin.
Figure 4. An error occurred while trying to connect two Entity Types using Role Association.

Figure 5. An error occurred while trying to connect Value Type to Unary Predicate using Role Association.

Figure 6. An error occurred while trying to connect twice Entity Type and Unary Predicate using Role Association.
5. Conclusion
A Protégé plugin allowing to create an ORM2-diagram using the live error checking approach was implemented. As a result, a valid object-oriented diagram model is formed in the computer memory.

The structure of the developed Protégé plugin allows a) minimising the linkage between its components and b) modifying the rules of constructing and rendering diagrams independently of each other.

6. Future work
The future work will continue in two directions. First, the ORM2-OWL2_Mapper component will be implemented to map the ORM2-diagram to the OWL2-ontology and vice versa, using existing approaches [12, 13].

Second, we plan to implement more complex elements of ORM2-diagram: uniqueness constraints, mandatory role constraints, and set-comparison constraints. To implement them, we will need to extend the ORM2-diagram graph model and create additional classes in the ORM2_Diagram and ORM2_Presenter components using extension points.

7. Acknowledgements
This paper presents the results of research carried out under the RFBR grants 18-07-00032 "Intelligent support of decision making of knowledge management for learning and scientific research based on the collaborative creation and reuse of the domain information space and ontology knowledge representation model" and 20-07-00764 "Conceptual modelling of the knowledge domain on the comprehension level for intelligent decision-making systems in the learning".

References
[1] Robinson S, Arbez G, Birta L G, Tolk A and Wagner G 2015 Conceptual modeling: Definition, purpose and benefits 2015 Winter Simulation Conference (WSC) (IEEE)
[2] Guizzardi G 2005 Ontological foundations for structural conceptual models
[3] Litovkin D, Anikin A and Kultsova M 2019 Interactive visualization of ontology-based conceptual domain models in learning and scientific research Advances in Intelligent Systems and Computing pp 365-374
[4] Litovkin D, Anikin A and Kultsova M 2019 Semantic zooming approach to semantic link network visualization Communications in Computer and Information Science (Springer International Publishing) pp 81-95
[5] Lembo D, Pantaleone D, Santarelli V and Savo D F 2016 Easy owl drawing with the graphol visual ontology language Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning KR’16 (AAAI Press) pp 573-576
[6] Bārzdinš J, Bārzdinš G, Čeräns K, Liepiņš R and Sprogis A 2010 UML style graphical notation and editor for OWL 2 Lecture Notes in Business Information pp 102–114
[7] Moreira J L R, Sales T P, Guerson J, Braga B F B, Brasileiro F and Sobral V 2016 Menthor
editor: An ontology-driven conceptual modeling platform *Proceedings of the Joint Ontology Workshops 2016 (FOIS 2016), Annecy, France, July 6-9, 2016 (CEUR Workshop Proceedings)*

[8] Halpin T 2002 Metaschemas for ER, ORM and UML data models *Journal of Database Management* **13** 20-30

[9] Halpin T A 2015 Object-role modeling fundamentals: A practical guide to data modeling with ORM

[10] Jarrar M, Demey J and Meersman R 2003 On using conceptual data modeling for ontology engineering *Journal on Data Semantics* I pp 185-207

[11] Curland M and Halpin T 2011 The NORMA software tool for ORM 2 *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications* pp 190-204

[12] Hodrob R and Jarrar M 2012 On using a graphical notation in ontology engineering

[13] Keet C M 2007 Mapping the object-role modeling language ORM2 into description logic language DLRifd *CoRR* [abs/cs/0702089](http://dblp.uni-trier.de/db/journals/corr/corr07.html#abs-cs-0702089)