PKBD.Onto: A Plugin for Ontological Schemas Generation

Nikita Dorodnykh[0000-0001-7794-4462], Aleksandr Yurin[0000-0001-9089-5730] and Anastasia Vidiya

Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of Russian Academy of Sciences, Lermontov St. 134, Irkutsk, Russia
tualatin32@mail.ru, iskander@icc.ru

Abstract. The use of Semantic Web technologies (including ontologies) for intelligent systems and knowledge bases engineering is a widespread practice, it is true especially for tasks of conceptualization and formalization. However, tools and approaches used for these tasks in most cases provide only a manual manipulation of concepts and relationships. In this regard, the use of various information sources for automated ontology engineering is relevant. One of these sources is spreadsheets. In this paper, we propose an approach for the automated creation of ontological schemas based on the analysis and transformation of spreadsheets data. The feature of our approach is the original relational canonicalized form of spreadsheets. This form is used for preprocessing spreadsheets and unifying the input data. The proposed approach is implemented in the form of a plugin (PKBD.Onto) for Personal Knowledge Base Designer - software for prototyping rule-based expert systems. The main stages of the approach, the architecture and functions of the plugin, and the case study are also described.

Keywords: Spreadsheets, Canonical Spreadsheet, Ontological Schema, OWL, Model Transformation, Code Generation

1 Introduction

The use of Semantic Web technologies, including ontologies [8], for intelligent systems and knowledge bases engineering is a widespread practice. In most cases, ontologies and special software (e.g., Protégé, ONTOedit, Menthor Editor, Semaphore Ontology Editor, OntoStudio, WebOnto, Fluent Editor, etc.) are used by analysts and domain experts for tasks of knowledge conceptualization and formalization. However, these tools provide a weak integration with external information sources (e.g., databases, texts, tables, conceptual models, etc.) in terms of importing domain concepts and relationships. This fact reduces the efficiency of the ontology engineering process. One of the information sources that can be used for the automated creation of ontologies is spreadsheets. Today, a large volume of arbitrary tables has been accumulated worldwide [9] and presented in the spreadsheet-like formats (HTML,

EXCEL, and CSV). Arbitrary tables are a valuable data source in business intelligence and data-driven research.

In our previous papers [5, 14] we proposed an approach for automated analysis and transformation of spreadsheets into conceptual domain models in the form of UML class diagrams. In this paper, we propose to apply this approach for ontological schemas generation (ontologies at the TBox level) in the OWL2 DL format [6]. A feature of the proposed approach is the use of the original canonicalized form for representation of spreadsheets, which provides the unification of input data.

Our approach is implemented in the form of the plugin, namely, PKBD.Onto, for Personal Knowledge Base Designer (PKBD) [15] – software for prototyping rule-based expert systems. A case study for the proposed approach and the plugin description are also presented.

2 Background

2.1 Method for Spreadsheets Transformation

A big volume of arbitrary tables (e.g. cross-tabulations, invoices, roadmaps, and data collection forms) circulates in spreadsheet-like formats. Spreadsheets are characterized by a wide variety and heterogeneity of layouts, styles and content. This diversity determines the two groups of solutions for their processing:

- ad-hoc solutions, which are oriented on certain layout or domain;
- universal solutions that use pre-processing arbitrary spreadsheets to transform them into some unified canonicalized format for further automated processing [12, 13].

This work uses the results of the project for the development of a framework for creating systems of data extraction from arbitrary spreadsheets. It is a reason why we use the TabbyXL [11] canonical spreadsheet format designed for the analysis of spreadsheets from Industrial Safety Inspection (ISI) reports [14].

We use the following spreadsheet structure in a canonicalized form:

$$CS^{CSV} = \{D, RH, CH\}, \quad (1)$$

where D is a data block that describes literal data values (named "entries") belonging to the same data type (e.g., numerical, textual, etc.); RH is a set of row labels of the category; CH is a set of column labels of the category. The values in cells for heading blocks can be separated by the "|" symbol to divide categories into subcategories. Thus, the canonical table denotes hierarchical relationships between categories (headings).

The analysis of canonicalized spreadsheets is carried out line by line. At the same time, cells can contain several values (concepts) with the separator ("|"). A cell value with the separator ("|") is interpreted as a hierarchy of classes (concepts) or attributes (properties).

In our approach we use the following heuristic-based rules [14] for the transformation of arbitrary spreadsheets to a canonicalized form:

Rule 1: IF *RH* corresponds only one *CH* THEN *RH* transformed to a class with properties from *CH*.

Rule 2: IF *RH* corresponds only one *CH* and at the same time *RH* contains two values with the separator ("|") THEN *RH* transformed to a class with properties from *CH* and with an additional property "Name" that corresponds to *RH*-2.

Rule 3: IF *RH* contains two values with the separator ("|") and they correspond to two *CH* values with the separator ("|"), THEN *RH* transformed to the first class, *CH* transformed to the second class and a relationship stated between them.

Rule 4: IF *RH* corresponds to three *CH* values with the separator ("|"), THEN *RH* transformed to the first class with properties *CH*-1, and *CH* 2 and 3 transformed to the second class and a relationship stated between them.

All obtained parent-child relationships are interpreted as the association and the cardinality of "1..*" is determined by default.

By default attribute values are set based on the *D* column.

The main results of this algorithm are fragments of conceptual models. These fragments need to aggregate, including operations for clarifying the names of concepts, their properties and relationships, and also their possible merging and separation.

The following rules used for automatic aggregation of conceptual models fragments:

Rule 1: Merge two classes when they have equal names from duplicate fragments of class diagrams.

Rule 2: Merge two classes when they have the same structure, i.e. when sets of attributes are equal. In this case, only the first class with this structure stays in the model.

Rule 3: Merge two classes when they have similar names. The resulting fragments of class diagrams can describe the same objects or processes. We suggest using a simple string comparison method based on the Levenshtein distance [10] to determine the similarity between two names of classes. If the distance is less than or equal to three, then we assume the classes to be similar. Note that this is not enough, so we also look at the structure of classes (names of attributes must partly match).

Rule 4: Create a new association between two classes if homonymous classes and attributes exist. In this case, a name in one class is equivalent to the attribute name in another class. At the same time, the attribute of the same name is removed.

Rule 5: Remove duplicate associations between classes.

Manual merging and separation operations are performed by using PKBD.

2.2 PKBD: a Tool for Knowledge Base Engineering

We used PKBD when solving problems of knowledge bases of expert systems engineering, in particular, in the field of ISI [1]. PKBD is implemented as a desktop application designed for non-programmers. The main purpose of PKBD is to prototype knowledge bases that use the formalism of logical rules.

One of PKBD features is a support of the Rule Visual Modeling Language (RVML) [7]. RVML is considered as a UML extension. Other PKBD features are the following:

- a modular architecture that provides the ability to add modules for supporting knowledge programming languages. Currently, CLIPS and Drools are supported;
- integrability with conceptual modeling tools when importing and exporting concepts and relationships.

The PKBD architecture determines the interaction of the following main software components:

- a knowledge base management module, it provides storage of projects in the EKB format (the proprietary XML-like format);
- a user interface subsystem includes the following modules: software wizards for manipulating knowledge base elements, a GUI generation, a Tiny RVML editor;
- a subsystem for supporting programming language modules, it provides connection and disconnection of modules, access to their functions for generating program codes;
- a module of integration with conceptual models sources: IBM Rational Rose, StarUML, XMind, CMapTools, and TabbyXL;
- a rule engines control module provides activation of rule engine for testing knowledge bases;
- a module of interaction with the web-based software called Knowledge Base Development System (KBDS) [3].

Main functions of PKBD are:

- designing elements of rule bases (fact templates, facts, and rules) by non-programmers using a set of wizards and defined sources of conceptual models;
- checking the integrity of the developed knowledge bases (syntactic and semantic control);
- representing knowledge base elements using RVML;
- generating knowledge base codes in the CLIPS format;
- testing developed knowledge base codes (logical inference) using the integrated CLIPS rule engine;
- integrating with CASE-tools: IBM Rational Rose, StarUML, XMind, and CMapTools, regarding import and transformation of conceptual models in order to highlight the main entities (concepts) and relationships for creating knowledge base drafts;
- integrating with TabbyXL [11] in terms of import and transformation of canonical spreadsheet tables in order to highlight the main entities (concepts) and relationships for creating knowledge base drafts;
- interacting with the KBDS service.

We used PKBD as an open software platform and developed a PKBD.Onto plugin. This plugin implements our approach for ontological schemas generation in the OWL2 DL format.

3 Proposed Approach

3.1 Method

The method for generating ontological schemas is based on principles of a model transformation. A model transformation is one of the key concepts in Model-Driven Engineering (MDE) [2].

From a formal point of view our method can be represented as a chain of horizontal exogenous transformations:

$$T : CS^{CSV} \rightarrow CM^{XML} \rightarrow OS^{OWL}, \quad (2)$$

where CS^{CSV} is a source spreadsheet presented in a canonicalized form and saved in CSV format using TabbyXL. The structure of a canonical spreadsheet is described in Section 2.1; CM^{XML} is a conceptual model resulted from spreadsheet transformation, which is a form for the internal representation of domain concepts and relationships for PKBD; OS^{OWL} is a target ontological schema in the OWL2 DL format.

Using (2), let's describe CM^{XML} in more detail:

$$CM^{XML} = \langle C, DT, RL \rangle,$$

where C is a set of classes; DT is a set of datatypes; RL is a set of relationships between C. Let's refine C from (3) as follows:

$C = \{c_1 ... c_n\}, c_i = \langle name_i, AT_i \rangle, i = \overline{1,n}$, when $name_i$ is a class name; AT_i is a set of class attributes, $AT_i = \{a_{i,1},...,a_{i,k}\}$, $a_{i,j} = \langle name_j, type_j, value_j \rangle, j \in \overline{1,k}$, when $name_j$ is an attribute name; $type_j$ is an attribute datatype, $type_j \in DT$; $value_j$ is a possible attribute value.

$RL = \{rl_1 ... rl_n\}, rl_i = \langle name_i, type_i, lhs_i, rhs_i \rangle, i = \overline{1,n}$, when $type_i$ is a relationship type (inheritance, dependency, association, aggregation, composition, realization); $name_i$ is a relationship name; lhs_i is a left side of a relationship, $lhs_i = \langle name_{lhs}, cd_{lhs}, c_j \rangle$, when $name_{lhs}$ is a name of a class role at the left relationship side, cd_{lhs} is a cardinality of the left relationship side, c_j is a link of a class at the left relationship side, $c_j \in C$; rhs_i is a right side of a relationship, $rhs_i = \langle name_{rhs}, cardinality_{rhs}, c_k \rangle$, when $name_{rhs}$ is a name of a class role at the right relationship side, cd_{rhs} is a cardinality of the right relationship side, c_k is a link of a class at the right relationship side, $c_k \in C$. Wherein, $cd_{lhs}, cd_{rhs} = \{0,0..1,0..*,1,1..*\}$.

Using (2), let's describe OS^{OWL} in more detail:

$$OS^{OWL} = \langle C, OP, DP, DT \rangle,$$

when C is a set of classes; OP is a set of object properties; DP is a set of datatype properties; DT is a set of XML Schema datatypes. A detailed description of the OWL 2 DL specification is given in [6].

Analysis and transformation of source spreadsheets (CS^{CSV}) and formation of a conceptual model (CM^{XML}) are discussed in detail in [5, 14]. In this paper, we will describe in detail how to obtain ontological schemas (OS^{OWL}). For this, using (2), let's describe a transformation operator (T):

$$T = \langle T_{CS-CM}, T_{CM-OSM}, T_{OSM-OS} \rangle,$$

$$T_{CS-CM} : CS^{CSV} \to CM^{XML}, T_{CM-OSM} : CM^{XML} \to OSM,$$

$$T_{OSM-OS} : OSM \to OS^{OWL},$$

where T_{CS-CM} is a set of rules for transformation of a source spreadsheet in the CSV format into a conceptual model, for example, a UML class diagram; T_{CM-OSM} is a set of rules for transformation of a conceptual model into an ontological schema model; T_{OSM-OS} is a set of rules for transformation of an ontological schema model into OWL ontology code at the TBox level.

Wherein: OSM is an ontological schema model designed for a unified representation and storage of knowledge extracted from various information sources. This model abstracts from features of knowledge representation languages and their dialects used for the implementation of ontologies (e.g., OWL, RDFS, etc.).

So, using sets of transformation rules (T_{CM-OSM} and T_{OSM-OS}), ontological schemas generation (OS^{OWL}) includes four main stages.

Stage 1: Analysing and transforming an XML structure of PKBD internal knowledge representation for conceptual models. This stage involves extracting elements, their properties, and relationships from an XML tree (the depth-first search for elements).

Stage 2: Forming an ontological schema model. The main objective of this stage is obtaining typical ontological fragments in the form of a set of classes and their relationships (object and datatype properties), which describe a certain domain and based on the extracted XML elements.

Stage 3: Generating an ontological schema code in the OWL format based on an ontological schema model.

Transformations themselves can be described using special transformation languages, for example, Transformation Model Representation Language (TMRL) [4]. In this work, we use a general-purpose language to implement transformations. Moreover, all transformations can be represented in tabular form (Table 1).

Table 1. Main correspondences between elements of a conceptual model, an ontology schema model, and OWL constructions.

CM	OSM	OWL
Model	Ontology	owl:Ontology
Class	Class	owl:Class
Generalization (class)	Class (superclass)	rdfs:subClassOf
Class (name)	Class (name)	rdf:about
Association	Relationship	owl:ObjectProperty
AssociationEnd (class)	Rhs	owl:ObjectProperty (rdfs:domain)
AssociationEnd (class)	Lhs	owl:ObjectProperty (rdfs:range)
Attribute	Property	owl:DatatypeProperty
Attribute (name)	Property (name)	owl:DatatypeProperty (rdfs:domain)
Attribute (value)	Property (value)	owl:DatatypeProperty (rdfs:range)
Attribute (description)	Property (description)	rdfs:comment

Stage 4: Editing an obtained ontological schema. This stage is additional and represents a refinement (modification) of OWL code obtained with the aid of various ontological modeling tools, for example, Protégé and others.

So, the main result of these stages is a set of ontology classes and their properties, which define an ontological schema at the TBox level.

3.2 PKBD.Onto: a Plugin for PKBD

The PKBD.Onto plugin is implemented in the form of a Dynamic Link Library (DLL) that is dynamically connected via a unified PKBD API.

The unified PKBD API for supporting integration modules with external software in terms of import and export contains three functions:

- getting a description of DLL including name and version ("DllInfo" function);
- getting a detailed description of DLL ("About" function);
- executing a main function of DLL, while a conceptual model in the PKBD format, a resulting file name, and a list of possible parameters are passed as a parameter ("Execute" function).

In the PKBD.Onto plugin architecture (Fig. 1) can be distinguished following components:

- supporting a PKBD format of conceptual models, which provides access and manipulation of model elements;
- transforming the input model to the OWL2 DL format;
- transforming the input model to a set of linked data in the RDF format (can be viewed as a mean for obtaining a set of specific facts).

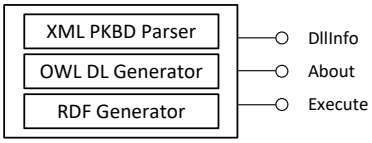

Fig. 1. A PKBD.Onto plugin architecture.

3.3 Case Study

Currently, PKBD is used in the educational process at Irkutsk National Research Technical University (IrNRTU), Institute of Information Technology and Data Science. Therefore, as an example, let's consider the educational task of developing an ontological schema fragment.

Information on minerals in the form of arbitrary spreadsheets is used as source data (Fig. 2). To unify the input data, a source arbitrary spreadsheet was preprocessed and a canonical spreadsheet resulted (Fig. 3).

Next, the canonical spreadsheet is analyzed using PKBD, in particular, by the PKBD.Onto plugin. Conceptual model elements are extracted as a result of this analysis. These elements can be visually represented as an RVML schema (Fig. 4). The obtained model requires modification, namely, all minerals were aggregated into a "Diamond" class (template), which must be renamed to "Mineral".

B	C	D	E	F	G	H	I
Mineral	**Color**	**Hardness (Moos)**		**Transparency**		**Syngony**	
		Density	**Value**	**Bandwidth**	**Refractive index**	**System**	**Appearance of crystals**
Diamond	Transparent	3,47-3,55	10	Transparent	2,417-2,419	Cubic	Octahedral
Emerald	Green	2,69-2,78	7,5-8	Transparent, semi-transparent	1,56-1,6	Hexagonal	Massive
Turquoise	Blue-green	2,6-2,8	5-6	Non-transparent	1,61-1,65	Triclinic	Massive
Spinel	Red, pink	3,57-3,72	7,5-8	Transparent	1,71-1,76	Cubic	Octahedral
Chrysolite	Green	3,2-3,4	6,5-7	Transparent	1,64-1,70	Rhombic	Prismatic-pyramidal
Аметист	Violet	2,63-2,65	7	Transparent, semi-transparent	1,543-1,554	Trigonal	Rhombohedral
Amethyst	Green	2,9-3	6-6,5	Non-transparent	1,62	Monoclinic	Massive
Topaz	Pink, yellow	3,49-3,57	8	Transparent	1,606-1,638	Rhombic	Prismatic
Opal	Yellow, red	1,96-2,2	5,5-6,5	Semi-transparent	1,44-1,46	Amorphous	Massive
Tourmaline	Red,green	3,02-3,26	7-7,5	Transparent, semi-transparent	1,616-1,652	Trigonal	Prismatic

Fig. 2. An example of a source arbitrary spreadsheet (before preprocessing).

DATA	RowHeading	ColumnHeading
Transparent	Diamond	Color
3,47-3,55	Diamond	Hardness (Moos)|Density
10	Diamond	Hardness (Moos)|Value
Transparent	Diamond	Transparency|Bandwidth
2,417-2,419	Diamond	Transparency|Refractive index
Cubic	Diamond	Syngony|System
Octahedral	Diamond	Syngony|Appearance of crystals
Green	Emerald	Color
2,69-2,78	Emerald	Hardness (Moos)|Density
7,5-8	Emerald	Hardness (Moos)|Value
Transparent, semi-transparent	Emerald	Transparency|Bandwidth
1,56-1,6	Emerald	Transparency|Refractive index
Hexagonal	Emerald	Syngony|System
Massive	Emerald	Syngony|Appearance of crystals

Fig. 3. A fragment of a canonical spreadsheet.

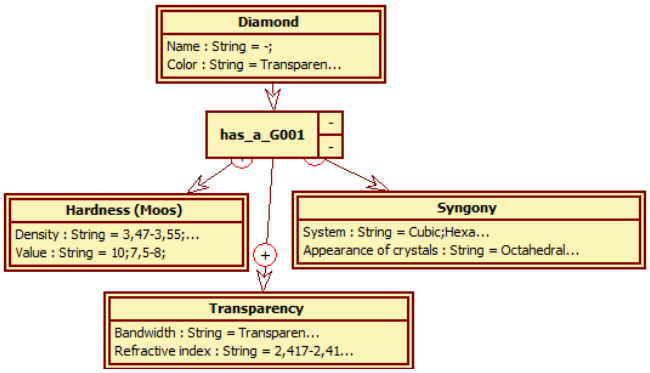

Fig. 4. A conceptual model in the form of a RVML schema resulted from the analysis of a canonical spreadsheet.

Based on the modified conceptual model (Fig. 4), we generated the code of the ontological schema in the OWL format. Then, this code can be verified in Protégé (Fig. 5).

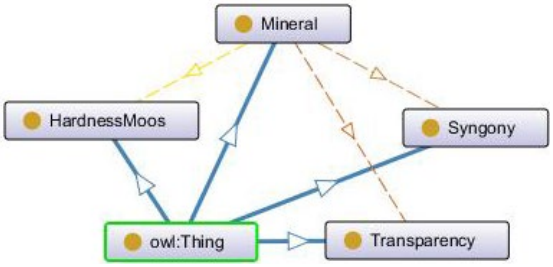

Fig. 5. A fragment of the ontological schema (Protégé).

4 Conclusions

In this paper, we describe a method and tool for ontological schemas generation (ontologies at the TBox level) in the form of a plugin for Personal Knowledge Base Designer. Spreadsheets reduced to a canonicalized form and saved in the CSV format were used as source data. Resulting OWL ontology codes are syntactically correct and can be evaluated by end-users.

The PKBD.Onto plugin allows one to create rapid prototypes of spreadsheet-based ontologies for a specific domain. Modified and refined ontologies can be used for intelligent systems and knowledge bases engineering [1].

5 Acknowledgments

This work was financially supported by the Council for Grants of the President of Russia (grant No. MK-1647.2020.9), Program of the Fundamental Research of the Siberian Branch of the Russian Academy of Sciences, project no. IV.38.1.2 (reg. no.

AAAA-A17-117032210079-1), project no. IV.38.1.3 (reg. no. AAAA-A17-117032210077-7). Results are achieved using the Centre of collective usage «Integrated information network of Irkutsk scientific educational complex».

References

1. Berman, A.F., Nikolaichuk, O.A., Yurin, A.Yu., Kuznetsov, K.A.: Support of Decision-Making Based on a Production Approach in the Performance of an Industrial Safety Review. Chemical and Petroleum Engineering 50(1-2), 730–738 (2015). DOI: 10.1007/s10556-015-9970-x
2. Da Silva, A.R.: Model-driven engineering: A survey supported by the unified conceptual model. Computer Languages, Systems & Structures 43, 139–155 (2015). DOI: 10.1016/j.cl.2015.06.001
3. Dorodnykh, N.O.: Web-based software for automating development of knowledge bases on the basis of transformation of conceptual models. Open Semantic Technologies for Intelligent Systems 1, 145–150 (2017).
4. Dorodnykh, N.O., Yurin, A.Yu.: A domain-specific language for transformation models. CEUR Workshop Proceedings (ITAMS-2018) 2221, 70–75 (2018).
5. Dorodnykh, N.O., Yurin, A.Yu., Shigarov, A.O.: Conceptual Model Engineering for Industrial Safety Inspection Based on Spreadsheet Data Analysis // Communications in Computer and Information Science. Modelling and Development of Intelligent Systems (MDIS 2019) 1126, 51–65 (2020). DOI: 10.1007/978-3-030-39237-6_4
6. Grau, B.C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: OWL 2: The next step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web 6(4), 309–322 (2008). DOI: 10.1016/j.websem.2008.05.001
7. Grishenko, M.A. Dorodnykh, N.O., Nikolaychuk, O.A., Yurin, A.Yu.: Designing rule-based expert systems with the aid of the model-driven development approach. Expert Systems 35(5), 1–23 (2018). DOI: 10.1111/exsy.12291
8. Guarino, N.: Formal Ontology in Information Systems. In: the First International Conference on Formal Ontology in Information Systems (FOIS'98) 46, 3–15 (1998).
9. Lehmberg, O., Ritze, D., Meusel, R., Bizer, C.: A large public corpus of web tables containing time and context metadata. Proceedings 25th International Conference Companion on World Wide Web, 75–76 (2016). DOI: 10.1145/2872518.2889386
10. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Tech. Rep. 8, Soviet Physics Doklady (1966).
11. Shigarov, A.O., Khristyuk, V.V., Mikhailov, A.M.: TabbyXL: Software platform for rule-based spreadsheet data extraction and transformation. SoftwareX 10, 100270 (2019). DOI: 10.1016/j.softx.2019.100270
12. Shigarov, A.O., Mikhailov, A.A.: Rule-based spreadsheet data transformation from arbitrary to relational tables. Information Systems 71, 123–136 (2017). https://doi.org/10.1016/j.is.2017.08.004
13. Tijerino, Y.A., Embley, D.W., Lonsdale, D.W., Ding, Y., Nagy, G.: Towards Ontology Generation from Tables. World Wide Web 8(3), 261–285 (2005). DOI: 10.1007/s11280-005-0360-8
14. Yurin, A.Yu., Dorodnykh, N.O.: A Reverse Engineering Process for Inferring Conceptual Models from Canonicalized Tables. Proceedings of the 2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON) 485–490 (2020). DOI: 10.1109/SIBIRCON48586.2019.8958458

15. Yurin, A.Yu., Dorodnykh, N.O.: Personal knowledge base designer: Software for expert systems prototyping. SoftwareX 11, 100411 (2020). DOI: 10.1016/j.softx.2020.100411