e-Informatica Software Engineering Journal, Volume 13, Issue 1, 2019, pages: 63–103, DOI 10.5277/e-Inf190103

Representation of UML Class Diagrams in OWL 2 on the Background of Domain Ontologies

Małgorzata Sadowska[*], Zbigniew Huzar[*]

[*]*Faculty of Computer Science and Management, Wrocław University of Science and Technology*

m.sadowska@pwr.edu.pl, zbigniew.huzar@pwr.edu.pl

Abstract

Background: UML class diagrams can be automatically validated if they are compliant with a domain knowledge specified in a selected OWL 2 domain ontology. The method requires translation of the diagrams into their OWL 2 representation.
Aim: The aim of this paper is to present transformation and verification rules of UML class diagrams to their OWL 2 representation.
Method: The analysis of the results of the systematic literature review on the topic of transformation rules between elements of UML class diagrams and OWL 2 constructs. The purpose of the analysis is to present the extent to which state-of-the-art transformation rules cover the semantics expressed in class diagrams. On the basis of the analysis, new transformation rules expressing the semantics not yet covered but expected from the point of view of domain modelling pragmatics have been defined.
Results: The first result is the revision and extension of the transformation rules identified in the literature. The second original result is a proposition of verification rules necessary to check if a UML class diagram is compliant with the OWL 2 domain ontology.
Conclusion: The proposed transformations can be used for automatic validation of compliance of UML class diagrams with respect to OWL 2 domain ontologies.

Keywords: UML, OWL 2, transformation rules, verification rules

1. Introduction

In [1], we presented an idea of a method for semantic validation of Unified Modeling Language (UML) class diagrams [2] with the use of OWL 2 Web Ontology Language (OWL 2) [3] domain ontologies. While UML has been known for many years, OWL is a much younger formalism and its main purpose is to represent knowledge in the Semantic Internet. The choice of OWL is justified by the fact that knowledge, and in particular ontologies collected on the Internet, will be increasingly used in business modelling as the first stage of software development. The proposed approach [1] requires a transformation of an UML class diagram constructed by a modeller into its semantically equivalent OWL 2 representation. Despite the fact that there are many publications which define some UML to OWL 2 transformations, to the best of the authors' knowledge, no study has investigated a complete mapping covering all diagram elements emphasized by pragmatic needs. This paper seeks to contribute in this field with a special focus on providing a full transformation of elements of an UML class diagram which are commonly used in business and conceptual modelling. All the transformations described in this paper and the method of validation explained in [1], have been implemented in a tool whose prototype was presented in [4]. On the basis of the proposed UML–OWL transformations, the tool has been further extended. Currently, the tool offers validation of the modified diagram, and can automatically suggest how the diagram

Submitted: 6 June 2018; Revised: 7 October 2018; Accepted: 7 November 2018; Available online: 14 December 2018

should be corrected on the basis of the ontology. A necessary requirement before the UML class diagram can be validated with the use of OWL domain ontology is that the diagram and the ontology must follow one agreed domain vocabulary. Moreover, the domain ontology must be consistent because it is the knowledge base for the area.

Our research is limited to the static elements of UML class diagrams – the behavioural aspect represented by class operations is omitted. This is due to the fact that the semantics of UML operations cannot be effectively expressed with the use of OWL 2 constructs, which do not represent behaviour. In order to identify which transformation rules of UML class diagrams into OWL constructs have already been proposed, we have performed a systematic review of literature. The extracted rules have been analysed, compared and extended. The resulting findings of how to conduct the transformation of UML class diagram to its OWL 2 representation are described further in this paper. In the rest of this paper, OWL always means OWL 2, if not stated otherwise.

Besides the transformation rules, the method of semantic validation of UML class diagrams requires the so called verification rules. This aspect is an original element of this research. Transforming the UML elements to OWL may introduce some new properties that may be in conflict with the ontology. The verification rules are specified in the form of either OWL verification axioms or verification queries.

It is then checked that the verification axioms are not present in the domain ontology because, if they are included, the diagram is contradictory to the domain knowledge. In other words, we can say that the verification axioms detect if the semantics of the diagram transformation is compliant with the axioms included in the domain ontology. Considering the inverse transformation (from the ontology to the diagram), the presence of the verification axioms in the domain ontology means that the reengineering transformation would remain in conflict with the semantics of the UML class diagram. In [1], we presented some examples of the consequences of a reengineering transformation of OWL to UML which does not take into account the verification rules.

Verification queries are used for extracting information from a domain ontology, the kind of information that could not be provided through inspecting the class diagram itself. The domain ontology can have more information regarding the elements of a UML class diagram, which is not explicitly expressed on the diagram. For example, the ontology can contain information about individuals. To give a more detailed perspective, the verification queries are used for: (a) checking if the classes denoted as abstract in the UML class diagram do not have any individuals assigned in the OWL domain ontology, (b) verifying if the multiplicity (of both the attributes and the association ends) is not violated on the side of the OWL domain ontology, and (c) checking if the user-defined list of literals of the specified enumerations on the UML class diagram is compliant with those defined in the OWL domain ontology. Technically, all verification queries are defined with the use of SPARQL[1] language.

The verification of UML class diagrams with the use of the proposed method is possible thanks to the initial normalization of the domain ontology and the normalization of the transformation axioms. The concept of OWL ontology normalization is our proposition [5]. Any input OWL 2 DL ontology after normalization is presented in a new but semantically equivalent form because the normalization rules only change the structure but do not affect the semantics of axioms or expressions in the OWL 2 ontology. The normalized OWL 2 DL ontologies have a unified structure of axioms so that they can be algorithmically compared without the need to conduct additional complex calculations. The extensive details of conducting the transformation of OWL 2 ontologies to their normalized form are described in [5]. In the rest of the paper OWL domain ontology is understood as OWL domain ontology after normalization (it should be done only once). Before comparison of axioms, all transformation axioms are also normalized (tool automatically saves transformation axioms also in the normalized form so that no additional delay is needed in

[1]SPARQL Query Language: https://www.w3.org/TR/sparql11-overview/

the verification algorithm). For the purpose of being compliant with the literature and for the potential use of transformation axioms for other purposes, all transformation axioms presented in this paper are not normalized. On the other hand, due to the fact that verification axioms are our proposition preliminary designed to support verification of UML class diagrams, some rules for verification axioms are already defined in the normalized form in order to reduce the number of unnecessary redundant verifications, the rest rules for verification axioms are not yet normalized for the purpose of clarity for readers but the tool also automatically saves the verification axioms directly as normalized.

In practical use of UML to OWL transformation, the initial phase involving modeller's attention is required. The modeller has to assure that all class attributes and association end names in one UML class are uniquely named. Otherwise, the transformation rules may generate repeating OWL axioms which may lead to inconsistencies or may be semantically incorrect.

The remainder of this article is organized as follows. Section 2 summarizes related works. Section 3 outlines which elements of UML class diagrams are commonly used in business and conceptual modelling. Section 4 describes the process and the results of the conducted systematic literature review which was focused on identifying the state-of-the-art transformation rules for translating UML class diagrams into their OWL representation. Section 5 presents the revised and extended transformation rules and proposes the verification rules. Section 6 summarises some important differences between OWL 2 and UML languages and their impact on transformation. Section 7 illustrates application of transformation and verification rules to example UML class diagrams. Section 8 is dedicated to the tool that implements the transformations. Finally, Section 9 concludes the paper.

2. Class diagrams in business and conceptual modelling

The UML specification [2] does not strictly specify which elements of UML class diagrams should or should not be included in the specific diagrams and this decision is always left to modellers. However, not all model elements are equally useful in the practice of business and conceptual modelling with UML class diagrams.

In [6], it is suggested that a full variety of UML constructs is not needed until the implementation phase and it is practiced that a subset of diagram elements useful for conceptual modelling in the business context is selected. The following static elements of UML class diagrams are suggested in literature as the most important in business and conceptual modelling [7, 8]:

- named classes,
- attributes of classes with types (either primitive or structured datatypes),
- associations between the classes (including aggregation) with the specified multiplicity of the association ends,
- generalization relationships.

Modelling a complex business requires using several views, each of which focuses on a particular aspect of business. Following [7], there are four commonly used Business Views: Business Vision View, Business Process View, Business Structure View and Business Behaviour View. The UML class diagrams are identified as useful [7] in Business Vision View and Business Structure View.

The UML class diagrams in a Business Vision View [7] are used to create conceptual models which establish a common vocabulary and demonstrate relationships among different concepts used in business. The important elements of UML class diagrams in the conceptual modelling are named classes and associations between the classes as they define concepts. The classes can have attributes as well as a textual explanation which together constitute a catalogue of terms. The textual descriptions may not be necessarily visible on the UML diagram but should be retrievable with the help of modelling tools. In the conceptual modelling with UML, attributes and operations of classes are not so much important [7] (can be defined only if needed) but relationships among the classes should be already correctly captured in models.

The UML class diagrams in a Business Structure View [7] are focused on presenting a structure of resources, products, services and information regarding the business including the organization of the company. The class diagrams in this view often include classes containing attributes with types and operations, as well as generalizations and associations with the specified multiplicity.

In [8], modelling business processes with UML class, activity and state machine diagrams is suggested. UML class diagrams with a number of predefined classes are used to describe process entity representatives (activities, agents, resources and artefacts). The examples in [8] present a business process at the level of the UML class diagram as consisting of classes with attributes, class generalizations, associations between the classes (including aggregation) with a specified multiplicity of the association ends. The class attributes are typed with either primitive or structured datatypes.

We have not found further recommendations for using additional static UML class diagram elements in the context of business or conceptual modelling in other reviewed literature positions. If the selected UML class diagram is compliant with the domain, it is reasonable to examine the diagram further. For example, the question outside the scope of this research is about the role of OCL[2] in business and conceptual modelling with UML class diagrams. Some other works investigate this aspect, e.g. in [9] an approach to translate OCL invariants into OWL 2 DL axioms can be found.

3. Review process

Kitchenham and Charters in [10] provide guidelines for performing systematic literature review (SLR) in software engineering. Following [10], a systematic literature review is a means of evaluating and interpreting all available research relevant to a particular research question, and aims at presenting a fair evaluation of a research topic by using a rigorous methodology. This section describes the carried out review aimed at identifying studies describing mappings of UML class diagrams to their OWL representations.

3.1. Research question

The research question is:
RQ: "What transformation rules between elements of UML class diagrams and OWL constructs have already been proposed?"

3.2. Data sources and search queries

In order to make the process repeatable, the details of our search strategy are documented below. The search was conducted in the following online databases: IEEE Xplore Digital Library, Springer Link, ACM Digital Library and Science Direct. These electronic databases were chosen because they are commonly used for searching literature in the field of Software Engineering. Additional searches with the same queries were conducted through ResearchGate and Google scholar in order to discover more relevant publications. These publication channels were searched to find papers published in all the available years until May 2018. The earliest primary study actually included was published in 2006.

For conducting the search, the following keywords were selected: "transformation", "transforming", "mapping", "translation", "OWL", "UML" and "class diagram". The keywords are alternate words and synonyms for the terms used in the research question, which aimed to minimize the effect of differences in terminologies. Pilot searches showed that the above keywords were too general and the results were too broad. Therefore, in order to obtain more relevant results, the search queries were based on the Boolean AND to join terms:

- "transformation" AND "OWL" AND "UML",
- "transforming" AND "OWL" AND "UML",
- "mapping" AND "OWL" AND "UML",
- "translation" AND "OWL" AND "UML",
- "transformation" AND "OWL" AND "class diagram",
- "transforming" AND "OWL" AND "class diagram",

[2]Object Constraint Language (OCL): http://www.omg.org/spec/OCL/

- "mapping" AND "OWL" AND "class diagram",
- "translation" AND "OWL" AND "class diagram".

3.3. Inclusion and exclusion criteria

The main inclusion criterion was that a paper provides some transformation rules between UML class diagrams and OWL constructs. Additionally, the study had to be written in English and be fully accessible through the selected online libraries. Additionally, there was a criterion for excluding a paper from the review results if the study described transformation rules between other types of UML diagrams to OWL representation or described transformation rules to other ontological languages.

3.4. Study quality assessment

The final acceptance of the literature was done by applying the quality criteria. The criteria were assigned a binary "yes"/"no" answer. In order for a work to be selected, it needed to provide "yes" answer to both questions from the checklist:

1. Are the transformation rules explicitly defined? For example, a paper could be excluded if it only reported on a possibility of specifying transformation rules for the selected UML elements, but such transformations were not provided.
2. Do the proposed transformation rules preserve the semantics of the UML elements? For example, a paper (or some selected transformation rules within the paper) could be excluded if the proposed rules in the transformation to OWL 2 did not preserve the semantics of the UML elements.

3.5. Study selection

During the search, the candidate papers for full text reading were identified by evaluating their titles and abstracts. The literature was included or excluded based on the selection criteria. The goal was to obtain the literature that answered the research question. The candidate papers, after eliminating duplicates, were fully read. After positive assessment of the quality of the literature items, they were added to the results of the systematic literature review.

Next, if the paper was included, its reference list was additionally scanned in order to identify potential further relevant papers (backward search). Later, the paper selection has additionally been extended by forward search related to works citing the included papers. In both backward search and forward search the papers for full text reading were identified based on reading title and abstract.

3.6. Threats to validity

We have identified threats to the validity of the conducted SLR, grouped in accordance with the categories presented in [11]. Wherever applicable, we included the applied mitigating factors corresponding to the identified threats.

Construct Validity: The specified search queries may not be able to completely cover all adequate search terms related to the research topic. As a mitigating factor, we used alternate words and synonyms for the terms used in the research question.

Internal Validity: The identified treats to internal validity relate to search strategy and further steps of conducting the SLR, such as selection strategy and quality assessment:

1. A threat to validity was caused by lack of assurance that all papers relevant to answering the research question were actually found. A mitigating factor to this threat was conducting a search with several search queries and analyzing the references of the primary studies with the aim of identifying further relevant studies.
2. Another threat was posed by the selected research databases. The threat was reduced by conducting the search with the use of six different electronic databases.
3. A threat was caused by the fact that one researcher conducted SLR. A mitigating factor to the search process and the study selection process was that the whole search process was twice reconducted in April 2018

and May 2018. The additional procedures did not change the identified studies.

External Validity: External validity concentrates on the generalization of findings derived from the primary studies. The carried search was aimed at identifying transformation rules of elements of UML class diagram to their OWL 2 representation. Some transformation rules could be formulated analogically in some other ontological languages, e.g. DAML+OIL, etc. Similarly, some transformation rules could be formulated analogically in some modelling languages or notations different then UML class diagrams, e.g. in Entity Relationship Diagram (ERD), EXPRESS-G graphical notation for information models, etc. A generalization of findings is out of scope of this research.

Conclusion Validity: The search process was twice reconducted and the obtained results have not changed. However, non-determinism of some database search engines is a threat to the reliability of this and any other systematic review because the literature collected through non-deterministic search engines might not be repeatable by other researchers with exactly the same results. In particular it applies to the results obtained with the use of Google scholar and ResearchGate.

4. Related work

4.1. Search results

In total, the systematic literature review identified 18 studies. 14 literature positions were found during the search: [12–26]. Additional 30 studies were excluded based on the quality assessment exclusion criterion.

Additional 3 studies were obtained through the analysis of the references of the identified studies (the backward search): [27–29].

The forward search has not resulted in any paper included. The majority of papers had already been examined during the main search and had already been either previously included or excluded. In the forward search, three papers describing transformation rules have been excluded because they were not related to UML. Most other papers have been excluded because they have not described transformation rules. Two papers have been excluded because the transformation rules were only mentioned but not defined. A relatively large number (approximately 20%) of articles has been excluded based on the language criterion – they had not been written in English (the examples of the observed repetitive languages: Russian, French, Turkish, Chinese, and Spanish).

The results of the search with respect to data sources are as follows (data source – number of selected studies): ResearchGate – 6; Springer Link – 3; IEEE Xplore Digital Library – 2; Google Scholar – 2; ACM Digital Library – 1; Science Direct – 1. In order to eliminate duplicates that were found in more than one electronic database, the place where a paper was first found was recorded.

Table 1. Search results versus years of publication

Year of publication	Resulting papers
2006	[23]
2008	[14, 16, 21, 27]
2009	[13]
2010	[26]
2012	[12, 17, 20, 22, 25]
2013	[19, 24, 28]
2014	[29]
2015	[18]
2016	[15]

To summarize, the identified studies include: 3 book chapters, 8 papers published in journals, 5 papers published in the proceedings of conferences, 1 paper published in the proceedings of a workshop and 1 technical report. The identified primary studies were published in the years between 2006–2016 (see Table 1). What can be observed is that the topic has been gaining greater attention since 2008. It should not be a surprise because OWL became a formal W3C recommendation in 2004.

4.2. Summary of identified literature

Most of the identified studies described just a few commonly used diagram elements (i.e. UML class, binary association and generalization between the classes or associations) while some other diagram elements obtained less attention in the literature (i.e. multiplicity of attributes, n-ary association or generalization sets). For some class diagram elements the literature offers incomplete transformations. Some of the transformation rules defined in the selected papers are excluded from the findings based on the quality criteria defined in Section 3.4. The state-of-the-art transformation rules were revised and extended. Section 5 contains detailed references to the literature related to relevant transformations. The following is a short description of the included studies:

The work presented in [18] transforms into OWL some selected elements of UML models containing multiple UML class, object and statechart diagrams in order to analyze consistency of the models. A similar approach is presented in [19], which is focused on detecting inconsistency in models containing UML class and statechart diagrams.

The papers [15, 17, 29] investigate the differences and similarities between UML and OWL in order to present transformations of selected (and identified as useful) elements of UML class diagram. In [29], the need for UML–OWL transformation is additionally motivated by not repeating the modelling independently in both languages.

In [14], a possible translation of few selected elements of several UML diagrams to OWL is presented. The paper takes into account a set of UML diagrams: use case, package, class, object, timing, sequence, interaction overview and component. The behavioural elements in UML diagrams in [14] are proposed to be translated to OWL with annotations.

The work of [26] focuses on representing UML and MOF-like metamodels with the use of OWL 2 language. The approach includes proposition of transforming Classes and Properties.

The paper [27] compares OWL abstract syntax elements to the equivalent UML features and appropriate OCL statements. The analysis is conducted in the direction from OWL to UML. For every OWL construct its UML interpretation is proposed.

The article [20] describes transformation rules for UML data types and class stereotypes selected from UML profile defined in ISO 19103. A full transformation for three stereotypes is proposed. The article describes also some additional OWL–UML mappings. The focus of [28] is narrowed to transformation of data types only.

Some works are focused on UML–OWL transformations against the single application domain. The paper [21] depicts the applicability of OWL and UML in the modelling of disaster management processes. In [16], transportation data models are outlined and the translation of UML model into its OWL representation is conducted for the purpose of reasoning.

The works presented in [12, 13, 23] are focused on extracting ontological knowledge from UML class diagrams and describe some UML–OWL mappings with the aim to reuse the existing UML models and stream the building of OWL domain ontologies. The paper [12] from 2012 extends and enhances the conference paper [13] from 2009. Both papers were analysed during the process of collecting the data in case of detection of any significant differences in the description of transformation rules.

In [22], UML classes are translated into OWL. Finally, [24, 25] present a few transformations of class diagram elements to OWL.

5. UML class diagram and its OWL 2 representation

This section presents **transformation rules (TR)** which seek to transform the elements of UML class diagrams to their equivalent representations expressed in OWL 2. Some of the transformation rules come from the literature identified in the review (e.g. **TR1** in Table 2). Another group of rules have their archetypes in the state-of-the-art transformation rules but we have refined them in order to clarify their contexts of use (e.g. $\mathbf{TR_A}$, $\mathbf{TR_C}$ in Section 6.2), or extend their application to a broader scope (e.g. **TR1** in Table 5). The remaining transformation rules are our new propositions (e.g. **TR5** in Table 7).

In contrast to the approaches available in the literature, together with the transformation rules we define the **verification rules (VR)** for all elements of a UML class diagram wherever applicable. The need for specifying verification rules results from the fact that we would like to check the compliance of the OWL representation of UML class diagram with the OWL domain ontology. The role of verification rules is to detect if the semantics of a diagram is not in conflict with the knowledge included in the domain ontology.

All the transformation and verification rules are presented in Tables 2–21. We took into consideration all the static elements of UML class diagrams, which are important from the point of view of pragmatics (see Section 2). To summarize the results, most of the UML elements which are recommended [7, 8] in business or conceptual modelling with UML class diagrams are fully transformable to OWL 2 constructs:

– *Class* (Table 2),
– attributes of the *Class* (Table 4),
– multiplicity of the attributes (Table 5),
– binary *Association* – both between two different *Classes* (Table 6) as well as from a *Class* to itself (Table 7),
– multiplicity of the *Association* ends (Table 9),
– *Generalization* between *Classes* (Table 12),
– *Integer*, *Boolean* and *UnlimitedNatural* primitive types (Table 18),
– structured *DataType* (Table 19),
– *Enumeration* (Table 20),
– *Comments* to the *Class* (Table 21),

We additionally fully translated into OWL 2 the following UML elements which have not been identified among recommended for business or conceptual modelling but can be used in further stages of software development:

– *Generalization* between *Associations* (Table 13),
– *GeneralizationSet* with constraints (Tables 14–17),
– *AssociationClass* (Table 10 and Table 11),

The UML and OWL languages have different expressing power. We consider also the partial transformation, which is possible for:

– *String* and *Real* primitive types because they have only similar but not equivalent to OWL 2 types (Table 18),
– aggregation and composition can be transformed only as simple associations (Tables 6–7)
– n-ary *Association* – OWL 2 offers only binary relations, a pattern to mitigate the problem of transforming n-ary *Association* is presented (Table 8),
– *AbstractClass* – OWL 2 does not offer any axiom for specifying that a class must not contain any individuals. Although, it is impossible to confirm that the UML abstract class is correctly defined with respect to the OWL 2 domain ontology, it can be detected if it is not (Table 3).

The tables below present for each UML element its short description, a graphical symbol, transformation rules, verification rules, explanations or comments, limitations of the transformations (if any) and the works related for the transformation rules (if any). Additionally, some tables include references to Section 7, where examples of UML–OWL transformations are presented.

The convention for transformation and verification rules presentation is semi-formal, similar to the convention used in other publication presenting transformation rules e.g. [17, 20]. It seems to be more readable than a strict formal presentation. However, a formal presentation is implicitly defined in the programming tool which transforms any UML class diagram into a set of OWL axioms.

All OWL 2 constructs are written with the use of a functional-style syntax [30]. Additionally, the following convention is used:

– C – indicates an OWL class;
– CE (possibly with an index) – indicates a class expression;
– OPE (possibly with an index) – indicates an object property expression;
– DPE (possibly with an index) – indicates a data property expression;
– $\alpha = \beta$ – means textual identity of α and β OWL 2 constructs;

- $\alpha \neq \beta$ – means textual difference of α and β OWL 2 constructs;
- The elements of UML meta-model, UML model, and OWL entities or literals named in the UML model are written with the use of *italic* font;
- The OWL 2 constructs (axioms, expressions and datatypes) and SPARQL queries are written in **bold**.

All presented SPARQL queries use the following prefixes:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://*...selected ontology*>

5.1. Transformation of UML classes with attributes

Table 2. Classes and the defined rules

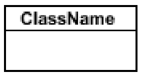
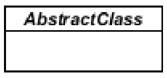
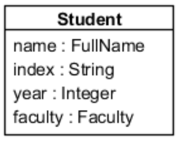
UML element	*Class*
Description of UML element	In UML, a *Class* [2] is purposed to specify a classification of objects.
Symbol of UML element	
Transformation rules	**TR1**: Specify declaration axiom for UML *Class* as OWL **Class**: **Declaration**(**Class**(:*ClassName*))
Verification rules	**VR1**: Check if :*ClassName* class has the **HasKey** axiom defined in the domain ontology. **HasKey**(:*ClassName*(OPE$_1$.. OPE$_m$) (DPE$_1$.. DPE$_n$))
Comments to the rules	1. Regarding **VR1**: The OWL **HasKey** axiom assures [30,31] that if two named instances of a class expression contain the same values of all object and data property expressions, then these two instances are the same. This axiom is in contradiction with the semantics of UML class because UML specification allows for creating different objects with exactly the same properties.
Related works	In [12–23,25–27], UML class is transformed to OWL with the use of **TR1** axiom.
Example	Section 7 example 1, 2 and 3

Table 3. Abstract classes and the defined rules

UML element	Abstract *Class*
Description of UML element	In UML, an abstract *Class* [2] cannot have any instances and only its subclasses can be instantiated.
Symbol of UML element	
Transformation rules	**Not possible.** The UML abstract classes cannot be translated into OWL because OWL does not offer any axiom for specifying that a class must not contain any individuals.
Verification rules	**VR1**: Check if the domain ontology contains any individual specified for the :*AbstractClass*. **SELECT** (**COUNT** (**DISTINCT** ?**ind**) **as** ?**count**) **WHERE** {?**ind rdf:type** :*AbstractClass*}

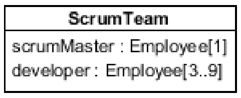
	If the :*AbstractClass* does not contain any individual specified in the domain ontology, the SPARQL query returns zero: "0"^^<http://www.w3.org/2001/XMLSchema#**integer**>
Comments to the rule	OWL follows the Open World Assumption [30], therefore, even if the ontology does not contain any instances for a specific class, it is unknown whether the class has any instances. We cannot confirm that the UML abstract class is correctly defined with respect to the OWL domain ontology, but we can detect if it is not (**VR1** checks if the class specified as abstract in the UML class diagram is indeed abstract in the domain ontology).
Related works	In [17, 20, 29], UML abstract class is stated as not transformable into OWL. In [17, 20], it is proposed that **DisjointUnion** is used as an axiom which covers some semantics of UML abstract class. However, UML specification does not require an abstract class to be a union of disjoint classes, and the **DisjointUnion** axiom does not prohibit creating members of the abstract superclass, therefore, it is insufficient.

Table 4. Attributes and the defined rules

UML element	Attributes
Description of UML element	The UML attributes [2] are *Properties* that are owned by a *Classifier*, e.g. *Class*.
Symbol of UML element	 Student name : FullName index : String year : Integer faculty : Faculty
Transformation rules	**TR1**: Specify declaration axiom(s) for attribute(s) as OWL data or object properties respectively **Declaration**(**ObjectProperty**(:*name*)) **Declaration**(**DataProperty**(:*index*)) **Declaration**(**DataProperty**(:*year*)) **Declaration**(**ObjectProperty**(:*faculty*)) **TR2**: Specify data (or object) property domains for attribute(s) **ObjectPropertyDomain**(:*name* :*Student*) **DataPropertyDomain**(:*index* :*Student*) **DataPropertyDomain**(:*year* :*Student*) **ObjectPropertyDomain**(:*faculty* :*Student*) **TR3**: Specify data (or object) property ranges for attribute(s) (for transformation of UML *PrimitiveTypes* refer to Table 18, for transformation of UML structure*DataTypes* to Table 19) **ObjectPropertyRange**(:*name* :*FullName*) **DataPropertyRange**(:*index* **xsd:string**) **DataPropertyRange**(:*year* **xsd:integer**) **ObjectPropertyRange**(:*faculty* :*Faculty*)
Verification rules	**VR1**: Check if the domain ontology contains **ObjectPropertyDomain** (or **DataPropertyDomain)** axiom specified for OPE (or DPE) where CE is specified for a different than given UML *Class* (here :*Student*) **ObjectPropertyDomain**(:*name* CE), where CE $\neq$:*Student* **DataPropertyDomain**(:*index* CE), where CE $\neq$:*Student* **DataPropertyDomain**(:*year* CE), where CE $\neq$:*Student* **ObjectPropertyDomain**(:*faculty* CE), where CE $\neq$:*Student* **VR2**: Check if the domain ontology contains **ObjectPropertyRange** (or **DataPropertyRange)** axiom specified for OPE (or DPE) where CE is

	specified for a different than given UML structure*DataType* (or DR is specified for a different than given UML *PrimitiveType*) **ObjectPropertyRange**(:*faculty* CE), where CE $\neq$:*Faculty* **DataPropertyRange**(:*index* DR), where DR $\neq$ **xsd:string** **DataPropertyRange**(:*year* DR), where DR $\neq$ **xsd:integer** **ObjectPropertyRange**(:*name* CE), where CE $\neq$:*FullName*
Comments to the rules	1. Both UML attributes and associations are represented by one meta-model element – *Property.* OWL also allows one to define properties. A transformation of UML attribute to OWL data property or OWL object property bases on its type. If the type of the attribute is *PrimitiveType* it should be transformed into OWL **DataProperty**. However, if the type of the attribute is a structured *DataType*,it should be transformed into an OWL **ObjectProperty**. 2. **VR1** checks whether or not the object properties (or respectively data properties) indicate that the UML attributes are specified for given UML *Class.* 3. **VR2** checks whether or not the object properties (or respectively data properties) indicate that the UML attributes of the specified UML *Class* have specified given types, either *PrimitiveTypes* or structured *DataTypes.*
Related works	**TR1**–**TR3** are proposed in [15–17, 20]. In [12–14, 18, 19, 21–24], all UML attributes are translated into data properties only.
Example	Section 7 example 2 and 3

Table 5. Multiplicity of attributes and the defined rules

UML element	Multiplicity of attributes
Description of UML element	In [2], multiplicity bounds of*MultiplicityElement* are specified in the form of <*lower-bound*> ".." <*upper-bound*>. The *lower-bound* is of a non-negative *Integer* type and the *upper-bound* is of an *UnlimitedNatural* type. The strictly compliant specification of UML in version 2.5 defines only a single value range for *MultiplicityElement.* However, in practical examples it is sometimes useful not limit oneself to a single interval. Therefore, the below UML to OWL mapping covers a wider case – a possibility of specifying more value ranges for a multiplicity element. Nevertheless, if the reader would like to strictly follow the current UML specification, the particular single *lower..upper* bound interval is therein also comprised. In comparison to UML, the OWL specification [30] defines three class expressions **ObjectMinCardinality**, **ObjectMaxCardinality** and **ObjectExactCardinality** for specifying the individuals that are connected by an object property to at least, at most or exactly to a given number (non-negative integer) of instances of the specified class expression. Analogically, **DataMinCardinality**, **DataMaxCardinality** and **DataExactCardinality** class expressions are used for data properties.
Symbol of UML element	 ScrumTeam scrumMaster : Employee[1] developer : Employee[3..9]
Transformation rules	**TR1**: If UML attribute is specified with the use of OWL **ObjectProperty**, its multiplicity should be specified analogously to **TR1** from Table 9 (multiplicity of association ends). If UML attribute is specified with the use of OWL **DataProperty**, its multiplicity should be specified with the use of axiom: **SubClassOf**(:*ClassName multiplicityExpression*) We define *multiplicityExpression* as one of class expressions: **A**, **B**, **C** or **D**: **A**. a **DataExactCardinality** class expression if UML *MultiplicityElement* has *lower-bound* equal to its *upper-bound*, e.g. "1..1", which is semantically equivalent to "1".

B. a **DataMinCardinality** class expression if UML *MultiplicityElement* has *lower-bound* of *Integer* type and *upper-bound* of unlimited *upper-bound*, e.g. “2..*”.
C. an **ObjectIntersectionOf** class expression consisting of **DataMinCardinality** and **DataMaxCardinality** class expressions if UML *MultiplicityElement* has *lower-bound* of *Integer* type and *upper-bound* of *Integer* type, e.g. “4..6”.
D. an **ObjectUnionOf** class expression consisting of a combination of **ObjectIntersectionOf** class expressions (if needed) or **DataExactCardinality** class expressions (if needed) or one **DataMinCardinality** class expression (if the last range has unlimited *upper-bound*), if UML *MultiplicityElement* has more value ranges specified, e.g. “2, 4..6, 8..9, 15..*”.
The following is the result of application of **TR1** to the above diagram:

SubClassOf(:*ScrumTeam*
  ObjectExactCardinality(1 :*scrumMaster* :*Employee*))
SubClassOf(:*ScrumTeam* **ObjectIntersectionOf**(
  ObjectMinCardinality(3 :*developer* :*Employee*)
  ObjectMaxCardinality(9 :*developer* :*Employee*)))

Verification rules

VR1: Regardless of whether or not the UML attribute is specified with the use of OWL **DataProperty** or **ObjectProperty**, the verification rule is defined with the use of the SPARQL query (only applicable for multiplicities with maximal *upper-bound* not equal “*”).

SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE {?**vioInd** :*leaf* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > *maxUpperBoundValue*)

where *maxUpperBoundValue* is a maximal *upper-bound* value of the multiplicity range. If the query returns a number greater than 0, it means that UML multiplicity is in contradiction with the domain ontology (**?vioInd** lists individuals that cause the violation).
The following is the result of definition of **VR1** to the above diagram:
maxUpperBoundValue for *scrumMaster*: 1
SPARQL query for *scrumMaster*:

SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** : *scrumMaster* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > 1)

maxUpperBoundValue for *developer*: 9
SPARQL query for *developer*:

SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** : *developer* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > 9)

VR2: Check if the domain ontology contains **SubClassOf** axiom, which specifies CE with different multiplicity of attributes than it is derived from the UML class diagram.

SubClassOf(:*ScrumTeam* CE)

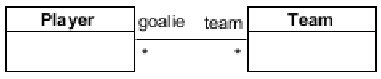
Comments to the rules

1.It should be noted that *upper-bound* of UML *MultiplicityElement* can be specified as unlimited: “*”. In OWL, cardinality expressions serve to restrict the number of individuals that are connected by an object property expression to a given number of instances of a specified class expression [30]. Therefore, UML unlimited *upper-bound* does not add any information to OWL ontology, hence it is not transformed.
2. Regarding **TR1**: the rule relies on the **SubClassOf(** CE_1 CE_2) axiom, which restricts CE_1 to necessarily inherit all the characteristics of CE_2, but not the other way around. The difference of using **EquivalentClasses(** CE_1 CE_2)

	axiom is that the relationship is implied to go in both directions (and the reasoner would infer in both directions). 3. Regarding **VR1**: As motivated in [17], reasoners that base on Open World Assumption can detect a violation of an upper limit of the cardinality restrictions only. This is caused by the fact that in Open World Assumption it is assumed that there might be other individuals beyond those that are already presented in the ontology. The verification rules for the cardinality expressions are defined with the use of SPARQL queries, which are aimed to verify whether or not the domain ontology does have any individuals that are contradictory to **TR1** axiom. Therefore, the **VR1** verifies the existence of individuals that are connected to the selected object property a number of times that is greater than the specified UML multiplicity. 4. The rule **VR2** verifies if the ontology contains axioms which describe multiplicity of *Attributes* different than the multiplicity specified in the UML class diagram.
Related works	The related works present only partial solutions for multiplicity of attributes. In [29], a solution for a single value interval is proposed. In [17], multiplicity associated with class attributes is transformed to a single expression of exact, maximum or minimum cardinality. In [24], multiplicity is transformed only into maximum or minimum cardinality.
Example	Section 7 example 2

5.2. Transformation of UML associations

Table 6. Binary Associations between two different Classes and the defined rules

UML element	Binary *Association* (between two different *Classes*)
Description of UML element	Following [2], a binary *Association* specifies a semantic relationship between two *memberEnds* represented by *Properties*. Please note that in accordance with specification [2], the association end names are not obligatory. In the method of validation and the prototype tool we followed the same convention which is adopted for all metamodel diagrams throughout the specification ([2, page 61]): *If an association end is unlabeled, the default name for that end is the name of the class to which the end is attached, modified such that the first letter is a lowercase letter.* Due to the fact that our method of transformation requires additionally unique names, either the modeller has to rename the names, or the tool in such cases automatically adds subsequent numbers to the names. For transformation of UML multiplicity of the association ends, refer to Table 9.
Symbol of UML element	
Transformation rules	**TR1**: Specify declaration axiom(s) for object properties **Declaration**(**ObjectProperty**(:*team*)) **Declaration**(**ObjectProperty**(:*goalie*)) **TR2**: Specify object property domains for association ends (note: if the association contains an *AssociationClass*, the domains should be transformed in accordance with **TR1** from Table 10) **ObjectPropertyDomain**(:*team* :*Player*) **ObjectPropertyDomain**(:*goalie* :*Team*) **TR3**: Specify object property ranges for association ends **ObjectPropertyRange**(:*team* :*Team*) **ObjectPropertyRange**(:*goalie* :*Player*)

	TR4: Specify **InverseObjectProperties** axiom for the association **InverseObjectProperties**(:*team* :*goalie*)
Verification rules	**VR1**: Check if **AsymmetricObjectProperty** axiom is specified for any of UML association ends. **AsymmetricObjectProperty**(:*goalie*) **AsymmetricObjectProperty**(:*team*) **VR2**: Check if the domain ontology contains **ObjectPropertyDomain** specified for the same OPE but different CE than it is derived from the UML class diagram. **ObjectPropertyDomain**(:*team* CE), where CE$\neq$:*Player* **ObjectPropertyDomain**(:*goalie* CE), where CE$\neq$:*Team* **VR3**: Check if the domain ontology contains **ObjectPropertyRange** axiom specified for the given OPE but different CE than it is derived from the UML class diagram. **ObjectPropertyRange**(:*team* CE), where CE$\neq$:*Team* **ObjectPropertyRange**(:*goalie* CE), where CE$\neq$:*Player*
Comments to the rules	1. **TR4** is specified to state that both resulting object properties are part of one UML *Association.* 2. Regarding **VR1**: A binary *Association* between two different *Classes* may not be asymmetric. Please refer to Table 7 for explanation of asymmetric binary *Association* from a *Class* to itself. 3. Regarding **VR2**: If the domain ontology contains **ObjectPropertyDomain** specified for the same OPE but different CE than it is derived from the UML class diagram, the Association is defined in the ontology but between different Classes. 4. Regarding **VR3**: If the domain ontology contains **ObjectPropertyRange** axiom specified for the given OPE but different CE than it is derived from the UML class diagram, the Association is defined in the ontology but between different Classes.
Limitations of the mapping	1. UML *Association* has two important aspects. The first is related to its existence and it can be transformed to OWL. It should be noted that UML introduces an additional notation related to communication between objects. The second one concerns navigability of the association ends which is untranslatable because OWL does not offer any equivalent concept. 2. Both UML aggregation and composition can be only transformed to OWL as regular *Associations.* This approach loses the specific semantics related to the composition or aggregation, which is untranslatable to OWL.
Related works	In [14–22, 25, 27], **TR1**–**TR3** rules for the transformation of UML binary association to object property domain and range are proposed. In [15, 20, 26], **TR4** rule is additionally proposed. In [17, 20], a unidirectional association is transformed into one object property and a bi-directional association into two object properties (one for each direction). This interpretation does not seem to be sufficient because if an association end is not navigable in UML 2.5, access from the other end may be possible, but it might not be efficient ([2, page 198]).
Example	Section 7 example 1 and 3

Table 7. Binary Association from the Class to itself and the defined rules

UML element	Binary *Association* from a *Class* to itself
Description of UML element	A binary *Association* [2] contains two *memberEnds* represented by *Properties.* For transformation of multiplicity of the association ends, refer to Table 9.

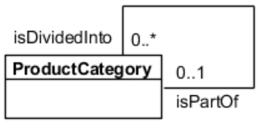
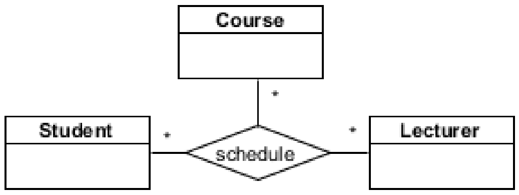
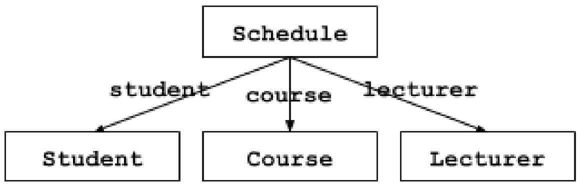
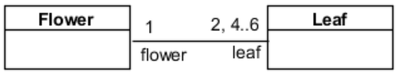
Symbol of UML element	 isDividedInto 0..* ProductCategory 0..1 isPartOf
Transformation rules	**TR1–TR4**: The same as **TR1–TR4** from Table 6. **TR5**: Specify **AsymmetricObjectProperty** axiom for each UML association end **AsymmetricObjectProperty**(:*isPartOf*) **AsymmetricObjectProperty**(:*isDividedInto*)
Verification rules	**VR1** is the same as **VR2** from Table 6. **VR2** is the same as **VR3** from 6.
Comments to the rules	1. In **TR2** domain and range of binary association is the same UML class. **VR4** checks if the domain ontology does not specify a different domain or range for the *Association*. 2. In **TR5** object property OPE is defined as asymmetric. In OWL, if an individual x is connected by OPE to an individually, then y cannot be connected by OPE to x.
Limitations of the mapping	The same as presented in Table 6.
Related works	For **TR1–TR4** related works are analogous as in Table 6, while **TR5** is our new proposition. In [15], the UML binary association from the *Class* to itself is converted to OWL with the use of two **ReflexiveObjectProperty** axioms. We do not share this approach because a specific association may be reflexive but in the general case it is not true. The **ReflexiveObjectProperty** axiom states that each individual is connected by OPE to itself. In consequence, it would mean that every object of the class should be connected to itself. The UML binary *Association* has a different meaning where the association ends have different roles.
Example	Section 7 example 2

Table 8. *N*-ary associations and the defined rules

UML element	*N*-ary *Association*
Description of UML element	UML *n*-ary *Association* [2] specifies the relationship between three or more *memberEnds* represented by *Properties*. For transformation of UML multiplicity of the association ends refer to Table 9.
Symbol of UML element	
Transformation rules	**Not possible** to directly represent UML *n*-ary associations in OWL 2. The following is a partial transformation based on the pattern presented in [32]. The pattern requires creating a new class and N new properties to represent the *n*-ary association. The figure below shows the corresponding classes and properties.  Schedule student course lecturer Student Course Lecturer

	TR1: Specify declaration axiom for the new class which represent the n-ary association (declaration axioms for other classes are added following Table 2) **Declaration**(**Class**(:*Schedule*)) **TR2**: Specify declaration axiom(s) for object properties **Declaration**(**ObjectProperty**(:*student*)) **Declaration**(**ObjectProperty**(:*course*)) **Declaration**(**ObjectProperty**(:*lecturer*)) **TR3**: Specify object property domains for association ends **ObjectPropertyDomain**(:*student* :*Student*) **ObjectPropertyDomain**(:*course* :*Course*) **ObjectPropertyDomain**(:*lecturer* :*Lecturer*) **TR4**: Specify object property ranges for association ends **ObjectPropertyRange**(:*student* :*Schedule*) **ObjectPropertyRange**(:*course* :*Schedule*) **ObjectPropertyRange(** :*lecturer* :*Schedule*) **TR5**: Specify **SubClassOf(** CE_1**ObjectSomeValuesFrom(** OPE CE_2)) axioms, where CE_1 is a newly added class, OPE are properties representing the UML *Association* and CE_2 are corresponding UML *Classes* **SubClassOf**(:*Schedule* ObjectSomeValuesFrom(:*student* :*Student*)) **SubClassOf**(:*Schedule* ObjectSomeValuesFrom(:*course* :*Course*)) **SubClassOf**(:*Schedule* ObjectSomeValuesFrom(:*lecturer* :*Lecturer*))
Verification rules	None
Limitations of the mapping	Properties in OWL 2 are only binary relations. Three solutions to represent an n-ary relation in OWL are presented in W3C Working Group Note [32] in a form of ontology patterns. Among the proposed solutions for n-ary association, we selected one the most appropriate to UML and we supplemented it by adding unlimited "*" multiplicity at every association end of the UML n-ary association.
Related works	The transformation rules (**TR1, TR2, TR5**) of a n-ary association base on the pattern proposed in [32]. **TR3, TR4** complement the rules, analogically as it is in binary associations. In [15], a partial transformation for n-ary association is proposed, but one rule should be modified because an object property expression is used in the place of a class expression.

Table 9. Multiplicity of association ends and the defined rules

UML element	Multiplicity of *Association* ends
Description of UML element	Description of multiplicity is presented in Table 5 (multiplicity of attributes). If no multiplicity of association end is defined, the UML specification implies a multiplicity of 1.
Symbol of UML element	
Transformation rules	**TR1**: For each association end with the multiplicity different than "*" specify axiom: **SubClassOf**(:*ClassName multiplicityExpression*) We define *multiplicityExpression* as one of class expressions: **A**, **B**, **C** or **D**: **A.** an **ObjectExactCardinality** if UML *MultiplicityElement* has *lower-bound* equal to its *upper-bound*, e.g. "1..1", which is semantically equivalent to "1". **B.** an **ObjectMinCardinality** class expression if UML *MultiplicityElement* has *lower-bound* of *Integer* type and *upper-bound* of unlimited *upper-bound*, e.g. "2..*". **C.** an **ObjectIntersectionOf** consisting of **ObjectMinCardinality** and **ObjectMaxCardinality** class expressions if UML *MultiplicityElement* has *lower-bound* of *Integer* type and *upper-bound* of *Integer* type, e.g. "4..6".

D. an **ObjectUnionOf** consisting of a combination of **ObjectIntersectionOf** class expressions (if needed) **OR ObjectExactCardinality** class expressions (if needed) **OR** one **ObjectMinCardinality** class expression (if the last range has an unlimited *upper-bound*), if UML *MultiplicityElement* has more value ranges specified, e.g. "2, 4..6, 8..9, 15..*".

The following is a result of application of **TR1** to the above diagram:

SubClassOf(:*Leaf* **ObjectExactCardinality**(1 :*flower* :*Flower*))
SubClassOf(:*Flower* **ObjectUnionOf**(
    ObjectExactCardinality(2 :*leaf* :*Leaf*)
    ObjectIntersectionOf(**ObjectMinCardinality**(4 :*leaf* :*Leaf*)
      ObjectMaxCardinality(6 :*leaf* :*Leaf*))))

TR2: Specify **FunctionalObjectProperty** axiom if a multiplicity of the association end equals 1.

FunctionalObjectProperty(:*flower*)

Verification rules

VR1: The rule is defined with the use of the SPARQL query (only applicable for multiplicities with maximal *upper-bound* not equal "*").

SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** :*leaf* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > *maxUpperBoundValue*)

where *maxUpperBoundValue* is a maximal *upper-bound* value of the multiplicity range. If the query returns a number greater than 0, it means that UML multiplicity is in contradiction with the domain ontology (**?vioInd** lists individuals that cause the violation).

The following is a result of application of **VR1** to the above diagram:
maxUpperBoundValue for *flower*: 1
SPARQL query for *flower*:

SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** :*flower* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > 1)

maxUpperBoundValue for *leaf*: 6
SPARQL query for *leaf*:

SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** :*leaf* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > 6)

VR2: Check if the domain ontology contains **SubClassOf** axiom, which specifies CE with different multiplicity of association ends than is derived from the UML class diagram.

SubClassOf(:*Leaf* CE)
SubClassOf(:*Flower* CE)

Comments to the rules

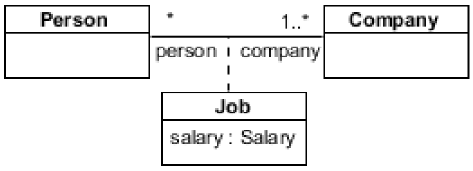
1. The **TR1**, **TR2** and **VR1** rules are explained in Table 5.
2. Regarding **TR2**: The **FunctionalObjectProperty** axiom states that each individual can have a maximum of one outgoing connection of the specified object property expression.
3. The rule **VR2** verifies whether or not the ontology contains axioms, which describe multiplicity of association ends different than multiplicity specified in the UML class diagram.
4. We have considered one additional validation rule for checking if the domain ontology contains **FunctionalObjectProperty** axiom specified for the association end which multiplicity is different from 1:

FunctionalObjectProperty(:*leaf*)

However, after analyzing of this rule, it would never be triggered. This is caused by the fact that the violation of cardinality is checked by **TR1** rule. And specifying **FunctionalObjectProperty** axiom in the ontology along with the transformation axiom describing cardinality different than 1, makes the ontology inconsistent.

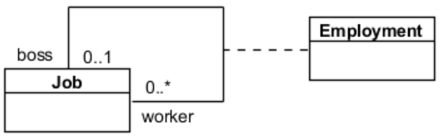
Related works	The related works present partial solutions for multiplicity of association ends. In [14, 18, 19, 26], the multiplicity of an association end is mapped to **SubClassOf** axiom containing a single **ObjectMinCardinality** or **ObjectMaxCardinality** class expression. In [17], **ObjectExactCardinality** expression is also considered and **TR2** rule is additionally proposed. In [12, 13, 15, 21, 22, 24], multiplicity is only suggested to be transformed into OWL cardinality restrictions.
Example	Section 7 example 1, 2 and 3

Table 10. Association class (the association is between two different classes) and the defined rules

UML element	*AssociationClass* (the *Association* is between two different *Classes*)
Description of UML element	*AssociationClass* [2] is both an *Association* and a *Class*, and preserves the semantics of both. Table 11 presents *AssociationClass* in the case when association is from a UML *Class* to itself.
Symbol of UML element	
Transformation rules	The binary association between *Person* and *Company* UML classes should be transformed to OWL in accordance with the transformations **TR1**, **TR3**–**TR4** from Table 6. The object property ranges should be specified in accordance with **TR2** from Table 6. The transformation of object property domains between *Person* and *Company* UML classes should be transformed with **TR1** rule below. Transformation of multiplicity of the association ends are specified in Table 9. The attributes of the UML association class :*Job* should be specified in accordance with the transformation rules presented in Table 4. If multiplicity of attributes is specified, it should be transformed in accordance with the guidelines from Table 5. **TR1**: Specify object property domains for *Association* ends **ObjectPropertyDomain**(:*person* **ObjectUnionOf**(:*Company* :*Job*)) **ObjectPropertyDomain**(:*company* **ObjectUnionOf**(:*Person* :*Job*)) **TR2**: Specify declaration axiom for UML association class as OWL **Class**: **Declaration**(**Class**(:*Job*)) **TR3**: Specify declaration axiom for object property of UML *AssociationClass* **Declaration**(**ObjectProperty**(:*job*)) **TR4**: Specify object property domain for UML *AssociationClass* **ObjectPropertyDomain**(:*job* **ObjectUnionOf**(:*Person* :*Company*)) **TR5**: Specify object property range for UML association class **ObjectPropertyRange**(:*job* :*Job*)
Verification rules	**VR1**: Check if :*Job* class has the **HasKey** axiom defined in the domain ontology. **HasKey**(:*Job* ($\text{OPE}_1 \ldots \text{OPE}_m$) ($\text{DPE}_1 \ldots \text{DPE}_n$)) **VR2**: Check if the domain ontology contains **ObjectPropertyDomain** axiom specified for a given OPE (from *Association* ends and *AssociationClass*) but different CE than is derived from the UML class diagram. **ObjectPropertyDomain**(:*person*CE), where CE $\neq$ **ObjectUnionOf**(:*Company* :*Job*), **ObjectPropertyDomain**(:*company* CE), where CE $\neq$ **ObjectUnionOf**(:*Person* :*Job*) **ObjectPropertyDomain**(:*job* CE), where CE $\neq$ **ObjectUnionOf**(:*Person* :*Company*)

	VR3: Check if the domain ontology contains **ObjectPropertyRange** axiom specified for the same object property of UML association class but different CE than it is derived from the UML class diagram. **ObjectPropertyRange**(:*job* CE), where CE $\neq$:*Job*
Comments to the rules	1. The proposed transformation of UML association class covers both the semantics of the UML class (**TR1**–**TR2**, plus the transformation of attributes possibly with multiplicity), as well as UML *Association* (**TR3**–**TR5**, plus the transformation of multiplicity of *Association* ends). 2. Regarding **TR1** and **TR3**: The domain of the specified property is restricted to those individuals that belong to the union of two classes. 3. Explanation of **VR1** is analogous to **VR1** from Table 2. 4. **VR2** checks if the UML *Association* and *AssociationClass* is specified correctly with respect to the domain ontology. **VR3** checks if the domain ontology does not specify a different range for the *AssociationClass*.
Related works	**TR1, TR3**–**TR5** transformation rules of the UML association class to OWL are original propositions and the proposed transformations to OWL cover full semantics of the UML *AssociationClass*. The literature [14, 15, 25] present only partial solutions for transforming UML association classes. In [14], it is only suggested that UML *AssociationClass* be transformed with the use of the named class (here: *Job*) and two functional properties that demonstrate associations (here: *Job–Person* and *Job–Company*). In [15, 25] some rules are with an unclear notation, more precisely *AssociationClass* is transformed to OWL with the use of **TR2** rule and a set of mappings which base on a specific naming convention.
Example	Section 7 example 3

Table 11. Association class (the Association is from a UML Class to itself) and the defined rules

UML element	*AssociationClass* (the *Association* is from a UML *Class* to itself)
Description of UML element	*AssociationClass* [2] is both an *Association* and a *Class*, and preserves the semantics of both. Table 10 presents *AssociationClass* in the case when association is between two different classes.
Symbol of UML element	
Transformation rules	All comments presented in Table 10 in TR section are applicable also for *AssociationClass* where association is from a UML *Class* to itself. Additionally, **TR5** from Table 7 has to be specified. Transformation rules **TR1**, **TR2**, **TR3** and **TR5** are the same as **TR1**, **TR2**, **TR3** and **TR5** from Table 10. Except for **TR4**, which has form: **TR4**: Specify object property domain for UML *AssociationClass* **ObjectPropertyDomain**(:*employment* :*Job*)
Verification rules	**VR1** and **VR3**: The same as **VR1** and **VR3** from Table 10. **VR2**: Check if the domain ontology contains **ObjectPropertyDomain** axiom specified for a given OPE (from *Association* ends and *AssociationClass*) but different CE than is derived from the UML class diagram. **ObjectPropertyDomain**(:*boss* CE), where CE $\neq$ **ObjectUnionOf**(:*Job* :*Employment*), **ObjectPropertyDomain**(:*worker* CE), where CE $\neq$ **ObjectUnionOf**(:*Job* :*Employment*) **ObjectPropertyDomain**(:*employment* CE), where CE $\neq$:*Job*

Comments to the rules	The same as presented in Table 10.
Related works	The same as presented in Table 10.

5.3. Transformation of UML generalization relationship

Table 12. Generalization between classes and the defined rules

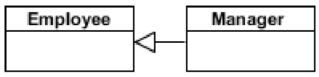
UML element	*Generalization* between *Classes*
Description of UML element	*Generalization* [2] defines specialization relationship between *Classifiers*. In case of UML classes it relates a more specific *Class* to a more general *Class*.
Symbol of UML element	
Transformation rules	**TR1**: Specify **SubClassOf(** CE$_1$ CE$_2$) axiom for the generalization between UML classes, where CE$_1$ represents a more specific and CE$_2$ a more general UML *Class*. **SubClassOf**(:*Manager* :*Employee*)
Verification rules	**VR1**: Check if the domain ontology contains **SubClassOf(** CE$_2$ CE$_1$) axiom specified for classes, which take part in the generalization relationship, where CE$_1$ represents a more specific and CE$_2$ a more general UML *Class*. **SubClassOf**(:*Employee* :*Manager*)
Related works	In [15,17–19,21–23,25–27,29] **TR1** is specified. In [12,13], generalizations are only suggested to be transformed to OWL with the use of **SubClassOf** axiom.
Example	Section 7 example 1 and 2.

Table 13. Generalization between associations and the defined rules

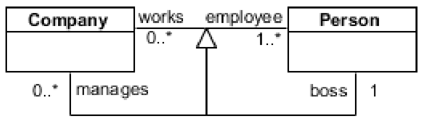
UML element	*Generalization* between *Associations*
Description of UML element	*Generalization* [2] defines specialization relationship between *Classifiers*. In case of the UML associations it relates a more specific *Association* to more general *Association*.
Symbol of UML element	 Company works employee Person 0..* 1..* 0..* manages boss 1
Transformation rules	**TR1**: Specify two **SubObjectPropertyOf(** OPE$_1$ OPE$_2$) axioms for the generalization between UML *Association*, where OPE$_1$ represents a more specific and OPE$_2$ a more general association end connected to the same UML *Class*. **SubObjectPropertyOf**(:*manages* :*works*) **SubObjectPropertyOf**(:*boss* :*employee*)
Verification rules	**VR1**: Check if the domain ontology contains **SubObjectPropertyOf(** OPE$_2$ OPE$_1$) axiom specified for associations, which take part in the generalization relationship, where OPE$_1$ represents a more specific and OPE$_2$ a more general UML association end connected to the same UML *Class*. **SubObjectPropertyOf**(:*works* :*manages*) **SubObjectPropertyOf**(:*employee* :*boss*)
Related works	In [15,17,18,26,27,29], **TR1** rule is proposed additionally with two **InverseObjectProperties** axioms (one for each association). This table does not add a transformation rule for **InverseObjectPropertie** axioms because the axioms were already added while transforming binary associations (see Tables 6, 7.
Example	Section 7 example 1

Table 14. GeneralizationSet with {incomplete, disjoint} constraints and the defined rules

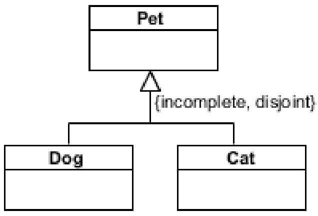
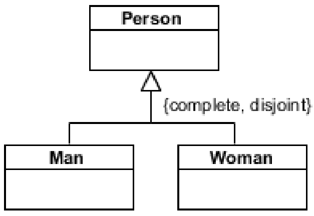
UML element	*GeneralizationSet* with {*incomplete, disjoint*} constraints
Description of UML element	UML *GeneralizationSet* [2] groups generalizations; *incomplete* and *disjoint* constraints indicate that the set is not complete and its specific *Classes* have no common instances.
Symbol of UML element	
Transformation rules	**TR1**: Specify **DisjointClasses** axiom for every pair of more specific *Classes* in the *Generalization.* **DisjointClasses**(:*Dog* :*Cat*)
Verification rules	**VR1**: Check if the domain ontology contains any of **SubClassOf(** CE_1 CE_2) or **SubClassOf(** CE_2 CE_1) axioms specified for any pair of more specific *Classes* in the *Generalization.* **SubClassOf**(:*Dog* :*Cat*) **SubClassOf**(:*Cat* :*Dog*)
Comments to the rules	1. **TR** and **VR** for *Generalization* between UML *Classes* are specified in Table 12. 2. Regarding **TR1**: the **DisjointClasses(** CE_1 CE_2) axiom states that no individual can be at the same time an instance of both CE_1 and CE_2 for $CE_1 \neq CE_2$.
Related works	In [15, 17, 29], **TR1** rule is proposed.

Table 15. GeneralizationSet with {complete, disjoint} constraints and the defined rules

UML element	*GeneralizationSet* with {*complete, disjoint*} constraints
Description of UML element	UML *GeneralizationSet* [2] is used to group generalizations; *complete* and *disjoint* constraints indicate that the generalization set is complete and its specific *Classes* have no common instances.
Symbol of UML element	
Transformation rules	**TR1**: Specify **DisjointUnion** axiom for UML *Classes* within the *GeneralizationSet.* **DisjointUnion**(:*Person* :*Man* :*Woman*)
Verification rules	**VR1**: Check if the domain ontology contains **SubClassOf(** CE_1 CE_2) or **SubClassOf(** CE_2 CE_1) axioms specified for any pair of more specific *Classes* in the *Generalization.* **SubClassOf**(:*Man* :*Woman*) **SubClassOf**(:*Woman* :*Man*) **VR2**: Check if the domain ontology contains **DisjointUnion(** C CE_1.. CE_N) axiom specified for the given more general UML *Class* (here :*Person*) and at least one more specific UML *Class* different than those specified on the UML class diagram.

	DisjointUnion(:*Person* CE_1.. CE_N)
Comments to the rules	1.**TR** and **VR** for *Generalization* between UML *Classes* are specified in Table 12. 2. **VR2** checks if the *GeneralizationSet* with {*complete, disjoint*} constraints is defined correctly with respect to domain ontology.
Related works	In [15, 17, 29], **TR1** is proposed.
Example	Section 7 example 2

Table 16. GeneralizationSet with {incomplete, overlapping} constraints and the defined rules

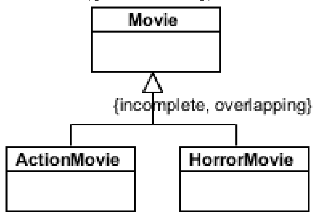
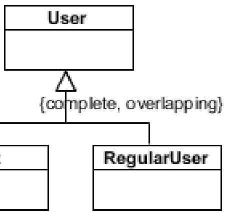
UML element	*GeneralizationSet* with {*incomplete, overlapping*} constraints
Description of UML element	UML *GeneralizationSet* [2] is used to group generalizations; *incomplete* and *overlapping* constraints indicate that the generalization set is not complete and its specific *Classes* do share common instances. If no constraints of *GeneralizationSet* are specified, {*incomplete, overlapping*} are assigned as default values ([2, p. 119]).
Symbol of UML element	 Movie {incomplete, overlapping} ActionMovie HorrorMovie
Transformation rules	None
Verification rules	**VR1**: Check if the domain ontology contains **DisjointClasses(** CE_1 CE_2) axiom specified for any pair of more specific *Classes* in the *Generalization.* **DisjointClasses**(:*ActionMovie* :*HorrorMovie*)
Comments to the rules	1. **TR** and **VR** for *Generalization* between UML *Classes* are specified in Table 12. 2. OWL follows Open World Assumption and by default incomplete knowledge is assumed, hence the UML *incomplete* and *overlapping* constraints of *GeneralizationSet* do not add any new knowledge to the ontology, so no **TR** are specified. 3. UML *overlapping* constraint states that specific UML *Classes* in the *Generalization* do share common instances. Therefore, the **DisjointClasses** axiom is a verification rule **VR1** for the constraint (the axiom assures that no individual can be at the same time an instance of both classes).
Related works	None

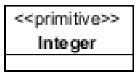
Table 17. GeneralizationSet with {complete, overlapping} constraints and the defined rules

UML element	*GeneralizationSet* with {*complete, overlapping*} constraints
Description of UML element	UML *GeneralizationSet* [2] is used to group generalizations; *complete* and *overlapping* constraints indicate that the generalization set is complete and its specific *Classes* do share common instances.

Symbol of UML element	
Transformation rules	**TR1**: Specify **EquivalentClasses** axiom for UML *Classes* within the *GeneralizationSet.* **EquivalentClasses**(:*User* **ObjectUnionOf**(:*Root* :*RegularUser*))
Verification rules	**VR1**: Check if the domain ontology contains **DisjointClasses(** CE_1 CE_2) axiom specified for any pair of more specific *Classes* in the *Generalization.* **DisjointClasses**(:*Root* :*RegularUser*) **VR2**: Check if the domain ontology contains **EquivalentClasses** axiom specified for the given more general UML *Class* (here :*User*) and **ObjectUnionOf** containing at least one UML *Class* different than specified on the UML class diagram for the more specific classes. **EquivalentClasses**(:*User* **ObjectUnionOf**($\mathrm{CE}_1..\mathrm{CE}_N$)), where **ObjectUnionOf**($\mathrm{CE}_1..\mathrm{CE}_N$) $\neq$ **ObjectUnionOf**(:*Root* :*RegularUser*)
Comments to the rules	1. **TR** and **VR** for *Generalization* between UML *Classes* are specified in Table 12. 2. Explanation for **VR1** is presented in Table 16. 3. **VR2** checks if the *GeneralizationSet* with {*complete, overlapping*} constraint is compliant with the domain ontology.
Related works	In [15], **TR1** rule is defined with additional **DisjointClasses(** :*Dog* :*Cat*) axiom. However, the **DisjointClasses** axiom should not be specified for the UML *Classes* which may share common instances.

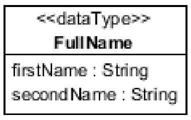
5.4. Transformation of UML data types

Table 18. Primitive types and the defined rules

UML element	*PrimitiveType*
Description of UML element	The UML *PrimitiveType* [2] defines a predefined *DataType* without any substructure. The UML specification [2] predefines five primitive types: *String*, *Integer*, *Boolean*, *UnlimitedNatural* and *Real.*
Symbol of UML element	
Transformation rules	It is impossible to define unambiguously the transformation of UML String and UML Real type, therefore, the decision on the final transformation is left to the modeller. The proposed transformations for the two types base on their similarity in UML 2.5 and OWL 2 languages. The transformation between UML predefined primitive types and OWL 2 datatypes: **TR1**: UML *String* has only a similar OWL 2 type: **xsd:string** String types in the sense of UML and OWL are countable sets. It is possible to define an infinite number of equivalence functions, which is left to the user, wherein, the UML is imprecise as to what the accepted characters are. **TR2**: UML *Integer* has an equivalent OWL 2 type: **xsd:integer** **TR3**: UML *Boolean* has an equivalent OWL 2 type: **xsd:boolean** **TR4**: UML *Real* has two similar OWL 2 types: **xsd:float** and **xsd:double** Both UML and OWL 2 languages describe types that are subsets of the set of

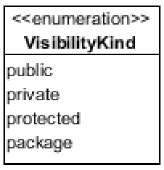
	real numbers. The subsets are countable. If one accepts a 32 or 64-bit precision of UML *Real* type, they will obtain an appropriate compatibility with OWL 2 **xsd:float** or **xsd:double** types.
	TR5: UML *UnlimitedNatural* can be explicitly defined in OWL 2 as:
	DatatypeDefinition(:*UnlimitedNatural*
	DataUnionOf(**xsd**:nonNegativeInteger
	DataOneOf("*"^^**xsd**:**string**)))
Verification rules	None
Comments to the rules	The UML specification [2] on page 717 defines the semantics of five predefined *PrimitiveTypes*. The specification of OWL 2 [30] also offers predefined datatypes (many more than UML).
	TR1: An instance of UML *String* [2] defines a sequence of characters. Character sets may include non-Roman alphabets. On the other hand, OWL 2 supports **xsd:string** defined in XML Schema [33]. The value space of **xsd:string** [33] is a set of finite-length sequences of zero or more characters that match the Char production from XML, where Char is any Unicode character, excluding the surrogate blocks, FFFE, and FFFF. The cardinality of **xsd:string** is defined as countably infinite. Due to the fact that the ranges of characters differ, UML *String* and OWL 2 **xsd:string** are only similar datatypes.
	TR2: An instance of UML *Integer* [2] is a value in the infinite set of integers $(\ldots, -2, -1, 0, 1, 2, \ldots)$. OWL 2 supports **xsd:integer** defined in XML Schema [33]. The value space of **xsd:integer** is an infinite set $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$. The cardinality is defined as countably infinite. The UML *Integer* and OWL 2 **xsd:integer** types can be seen as equivalent.
	TR3: An instance of UML *Boolean* [2] is one of the predefined values: true and false. OWL 2 supports **xsd:boolean** defined in XML Schema [33], which represents the values of two-valued logic :{true, false}. The lexical space of **xsd:boolean** is a set of four literals: 'true', 'false', '1' and '0' but the lexical mapping for **xsd:boolean** returns true for 'true' or '1', and false for 'false' or '0'. Therefore the UML *Boolean* and **xsd:boolean** types can be seen as equivalent.
	TR4: An instance of UML *Real* [2] is a value in the infinite set of real numbers. Typically [2] an implementation will internally represent *Real* numbers using a floating point standard such as ISO/IEC/IEEE 60559:2011, whose content is identical [2] to the predecessor IEEE 754 standard. On the other hand, OWL 2 supports **xsd:float** and **xsd:double**, which are defined in XML Schema [33]. The **xsd:float** [33] is patterned after the IEEE single-precision 32-bit floating point datatype IEEE 754-2008 and the **xsd:double** [33] after the IEEE double-precision 64-bit floating point datatype IEEE 754-2008. The value space contains the non-zero numbers $m \times 2^e$, where m is an integer whose absolute value is less than 2^{53} for **xsd:double** (or less than 2^{24} for **xsd:float**), and e is an integer between -1074 and 971 for **xsd:double** (or between -149 and 104 for **xsd:float**), inclusive. Due to the fact that the value spaces differ, UML *Real* and OWL 2 **xsd:double** (or **xsd:float**) are only similar datatypes.
	TR5: An instance of UML *UnlimitedNatural* [2] is a value in the infinite set of natural numbers (0, 1, 2…) plus unlimited. The value of unlimited is shown using an asterisk ('*'). UnlimitedNatural values are typically used [2] to denote the *upper-bound* of a range, such as a multiplicity; unlimited is used whenever the range is specified as having no *upper-bound*. The UML *UnlimitedNatural* can be defined in OWL and added to the ontology as a new datatype (**TR5**).
Related works	The related works are not precise with respect to the transformation of primitive types. In [17, 27–29], some mappings of UML and OWL types are only mentioned.
Example	Section 7 example 2

Table 19. Structured data types and the defined rules

UML element	Structured *DataType*
Description of UML element	The UML structured *DataType* [2] has attributes and is used to define complex data types.
Symbol of UML element	 <<dataType>> FullName firstName : String secondName : String
Transformation rules	**TR1**: Specify declaration axiom for UML data type as OWL class: **Declaration**(**Class**(:*FullName*)) **TR2**: Specify declaration axiom(s) for attributes – as OWL data or object properties respectively (see Table 4 for more information regarding attributes) **Declaration**(**DataProperty**(:*firstName*)) **Declaration**(**DataProperty**(:*secondName*)) **TR3**: Specify data (or object) property domains for attributes **DataPropertyDomain**(:*firstName* :*FullName*) **DataPropertyDomain**(:*secondName* :*FullName*) **TR4**: Specify data (or object) property ranges for attributes (OWL 2 datatypes for UML primitive types are defined in Table 18) **DataPropertyRange**(:*firstName* **xsd:string**) **DataPropertyRange**(:*secondName* **xsd:string**) **TR5**: Specify **HasKey** axiom for the UML data type expressed in OWL with the use of a class uniquely identified by the data and/or object properties. **HasKey**(:*FullName* ( ) (:*firstName* :*secondName*))
Verification rules	**VR1**: Check if the domain ontology contains **DataPropertyDomain** axiom specified for DPE where CE is different than given UML structured *DataType* **DataPropertyDomain**(:*firstName* CE), where CE $\neq$:*FullName* **DataPropertyDomain**(:*secondName* CE), where CE $\neq$:*FullName* **VR2**: Check if the domain ontology contains **DataPropertyRange** axiom specified for DPE where CE is different than given UML *PrimitiveType* **DataPropertyRange**(:*firstName* DR), where DR $\neq$ **xsd:string** **DataPropertyRange**(:*secondName* DR), where DR $\neq$ **xsd:string**
Comments to the rules	1. UML *DataType* [2] is a kind of *Classifier*, whose instances are identified only by their values. All instances of a UML *DataType* with the same value are considered to be equal [2]. A similar meaning can be assured in OWL with the use of **HasKey** axiom. The **HasKey** axiom [30] assures that each instance of the class expression is uniquely identified by the object and/or data property expressions. 2. **VR1** checks whether the data properties indicate that the UML attributes are correct for the specified UML structured *DataType*. 3. **VR2** checks whether the data properties indicate that the UML attributes of the specified UML structured *DataType* have correctly specified *PrimitiveTypes*.
Limitations of the mapping	Due to the fact that we define the UML structure *DataType* as an OWL **Class** and not as an OWL **Datatype** (see Section 6.3 for further explanation), the presented transformation results in some consequences. A limitation is posed by the fact that the instances of the UML *DataType* are identified only by their value [2], while the **TR1** rule opens a possibility of explicitly defining the named instances for the **Entity** in OWL.
Related works	In [28, 29] **TR1**–**TR5** rules and in [15] **TR2**–**TR5** rules are proposed for the transformation of UML structured *DataType*. In [17], it is only noted that UML *DataTypes* can be defined in OWL with the use of **DatatypeDefinition** axiom

	but no example is provided. The related works specify exclusively the data properties as attributes of the structured data types in **TR2**. We extend the state-of-the-art **TR2** transformation rule by the possibility of defining also object properties, wherever needed (see Table 4).
Example	Section 7 example 2

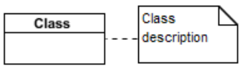
Table 20. Enumeration and the defined rules

UML element	*Enumeration*
Description of UML element	UML *Enumerations* [2] are kinds of *DataTypes*, whose values correspond to one of user-defined literals.
Symbol of UML element	 <<enumeration>> VisibilityKind public private protected package
Transformation rules	**TR1**: Specify declaration axiom for UML *Enumeration* as OWL **Datatype**: **Declaration**(**Datatype**(:*VisibilityKind*)) **TR2**: Specify **DatatypeDefinition** axiom including the named **Datatype** (here :*VisibilityKind*) with a data range in a form of a predefined enumeration of literals (**DataOneOf**). **DatatypeDefinition**(:*VisibilityKind* **DataOneOf**(*"public" "private" "protected" "package"*))
Verification rule	**VR1**: Check if the list of user-defined literals in the *Enumeration* on the class diagram is correct and complete with respect to the OWL datatype definition for *:VisibilityKind* included in the domain ontology. The SPARQL query: **SELECT** ?**literal** { :*VisibilityKind* **owl**:**equivalentClass** ?**dt**. ?**dt a rdfs**:**Datatype** ; **owl**:**oneOf**/**rdf**:**rest**∗/**rdf**:**first** ?**literal** } returns a list of literals of the enumeration from the domain ontology. The list of literals should be compared with the list of user-defined literals on the class diagram if the UML *Enumeration* includes a correct and complete list of literals.
Limitations of the mapping	*Enumerations* [2] in UML are specializations of a *Classifier* and therefore can participate in generalization relationships. OWL has no construct allowing for generalization of datatypes. See Section 6.3 for further explanation.
Related works	In [17, 20, 28, 29], UML *Enumeration* is transformed to OWL with the use of **TR1**–**TR2** rules.

5.5. Transformation of UML comments

Table 21. Comment and the defined rules

UML element	*Comment* to the *Class*
Description of UML element	In accordance with [2], every kind of UML *Element* may own *Comments* which add no semantics but may represent information useful to the reader. In OWL it is possible to define the annotation axiom for OWL **Class**, **Datatype**, **ObjectProperty**, **DataProperty**, **AnnotationProperty** and **NamedIndividual**. The textual explanation added to UML *Class* is identified as useful for conceptual modelling [7], therefore the *Comments* that are connected to UML *Classes* are taken into consideration in the transformation.

Symbol of UML element	
Transformation rules	**TR1**: Specify annotation axiom for UML *Comment* **AnnotationAssertion**(**rdfs**:**comment** :*Class* "*Class description*"^^**xsd**:**string**)
Verification rule	Not applicable
Comments to the rule	As UML *Comments* add no semantics, they are not used in any method of semantic validation [1]. In OWL the **AnnotationAssertion** [30] axiom does not add any semantics either, and it only improves readability.
Related works	The transformation of UML *Comments* in the context of mapping to OWL has not been found in literature.

6. Influence of UML–OWL differences on transformations

Obviously, OWL 2 and UML 2.5 languages differ from each other.

In general, notice that OWL ontologies are based on the Open World Assumption while UML class diagrams are based on Closed World Assumption. We can compare a UML class diagram to a given OWL ontology assuming that this ontology is in a given state. Examining that the UML class diagram conforms to the OWL ontology we transform the diagram into equivalent OWL representation and check if this representation forms a subset of the ontology. So, the notion of semantic equivalence relates only to the UML class diagram and its OWL representation.

The further part of the section focuses exclusively on two selected differences which influence the form of transformations.

6.1. Instances

OWL 2 defines several kinds of axioms: declarations, axioms about classes, axioms about objects and data properties, datatype definitions, keys, assertions (used to state that individuals are instances of e.g. class expressions) and axioms about annotations. What can be observed is that the information about classes and their instances (in OWL called individuals) coexists within a single ontology.

On the other hand, in UML two different kinds of diagrams are used in order to present the classes and objects. UML defines object diagrams which represent instances of class diagrams at a certain moment in time. The object diagrams focus on presenting attributes of objects and relationships between objects.

Despite the fact that information about the objects is not present in UML class diagrams, verification rules in the form of SPARQL queries take advantage of the knowledge about individuals in the domain ontology. The rules are useful in validation of class diagrams against the selected domain ontologies as they can check, for example, if an abstract class is indeed abstract (does not have any direct instances in ontology) or if multiplicity restrictions are specified correctly.

6.2. Disjointness in OWL 2 and UML

In OWL 2 an individual can be an instance of several classes [34]. It is also possible to state that no individual can be an instance of selected classes, which is called class disjointness. The information that some specific classes are disjoint is part of domain knowledge which serves a purpose of reasoning.

OWL specification emphasises [34]: *In practice, disjointness statements are often forgotten or neglected. The arguable reason for this could be that intuitively, classes are considered disjoint unless there is other evidence. By omitting disjointness statements, many potentially useful consequences can get lost.*

What can be observed in typical existing OWL ontologies, axioms of disjointness (**DisjointClasses**, **DisjointObjectProperties** and **DisjointDataProperties**) are stated for classes, object properties or data properties only for the most evident situations. If disjointness is not specified, the semantics of OWL states that the ontology does not contain enough information that disjointness takes place. For example, it is possible that the information is actually true but it has not been included in the ontology.

On the other hand, in a UML class diagram every pair of UML classes (which are not within one generalization set with an *overlapping* constraint) is disjoint, where disjointness is understood in the way that the classes have no common instances. This aspect of UML semantics could be mapped to OWL with the use of an extensive set of additional transformations. The transformations would not be intuitive from the perspective of OWL and should add a lot of unnecessary information which might never be useful due to the fact that e.g. one should consider every pair of classes on the diagram and add additional axioms for it.

For the purpose of completeness of our revision, below we present transformation rules also for disjointness:

– *Transformation rule for disjointness of UML classes* ($\mathbf{TR_A}$): Specify **DisjointClasses** axiom for every pair of UML Classes: CE_1, CE_2 where $\mathrm{CE}_1 \neq \mathrm{CE}_2$ and the pair is not in the generalization relation or within one generalization set with an overlapping constraint. *Comment*: The $\mathbf{TR_A}$ rule for classes within a generalization relationship was originally proposed in [17, 18, 20]. We have refined the rule in order to cover only the pairs of classes which are not only in a direct generalization relation but also not within one *GeneralizationSet* with an overlapping constraint. This is caused by the fact that the *GeneralizationSet* with the *overlapping* constraint (see Tables 16–17) defines specific *Classes*, which do share common instances. Please note that UML *GeneralizationSet* with *disjoint* constraint adds **DisjointClasses** axioms – either directly or indirectly through **DisjointUnion** axiom (see Tables 14–15).

– *Transformation rule for disjointness of UML attributes* ($\mathbf{TR_B}$): Specify **DisjointObjectProperties** axiom for every pair OPE_1, OPE_2 where $\mathrm{OPE}_1 \neq \mathrm{OPE}_2$ of object properties within the same UML Class (domain of both OPE_1 and OPE_2 is the same OWL **Class**) and specify **DisjointDataProperties** axiom for every pair DPE_1, DPE_2 where $\mathrm{DPE}_1 \neq \mathrm{DPE}_2$ of object properties within the same UML Class (domain of both DPE_1 and DPE_2 is the same OWL **Class**)
Comment: The $\mathbf{TR_B}$ rule is original proposition.

– *Transformation rule for disjointness of UML associations* ($\mathbf{TR_C}$): Specify **DisjointObjectProperties** axiom for every pair of association ends OPE_1 and OPE_2 where $\mathrm{OPE}_1 \neq \mathrm{OPE}_2$ and OPE_1 is not generalized by OPE_2 and OPE_2 is not generalized by OPE_1 and domain and range of OPE_1 and OPE_2 are the same classes.
Comment: In [17, 20], it is suggested that **DisjointObjectProperties** and **DisjointDataProperties** axioms for all properties that are not in a generalization relationship should be specified. In a general case this suggestion is not clear, but we have modified the rule to be applicable for UML associations which are not in generalization relationship. Even though the $\mathbf{TR_A}$, $\mathbf{TR_B}$ and $\mathbf{TR_C}$ rules are reasonable from the point of view of covering semantics of a class diagram to OWL, they have not been implemented in a tool for validation of UML class diagram [4] due to their questionable usefulness from the perspective of pragmatics. This is caused by the fact that including these rules would lead to a large increase in the number of axioms in the ontology, which would increase the computational complexity.

6.3. Concepts of class and datatype in UML and OWL

OWL 2 allows specifying declaration axioms for datatypes:

Declaration(**Datatype**(:*DatatypeName*))

However, the current specification of OWL 2 [30] does not offer any constructs neither to specify the internal structure of the datatypes, nor the possibility to define generalization relationships between the datatypes. Both are available in UML 2.5.

Please note that the OWL **HasKey**, **DataPropertyDomain** and **ObjectPropertyDomain** axioms can only be defined for the class expressions (not for the data ranges). Therefore the **TR2–TR5** rules in Table 19 can only be specified if the UML structured *DataType* is declared as an OWL **Class**. This transformation has its consequences, which are presented in Table 19.

If future extensions of the OWL language allow one to precisely define the internal structure of datatypes, by analogy, as it is possible in UML, the proposed transformation of UML structured *DataType* presented in Table 19 should then be modified. Additionally, if future extensions of the OWL language allow one to define generalization relationships between datatypes, the currently valid limitation of the transformation of UML *Enumeration* presented in Table 20 will no longer be applicable.

7. Examples of UML–OWL transformations

This section presents some examples of transformations of UML class diagrams to their equivalent OWL representations. The UML class diagram examples are relatively small but cover a number of different UML elements. For clarity of reading, the examples include references to tables from Section 5.

The order of transformations is arbitrary (the resulting set of axioms will always be the same despite the order) but we suggest to conduct the transformations starting from Table 2 to Table 21. In this way, all the classes with attributes will be mapped to OWL first, then the associations and generalization relationships and finally data types and comments.

Each example includes two tables containing transformational and verificational part of UML class diagram (e.g. in Example 1 there are two tables: 22 and 23). Each verificational part should be considered in the context of the selected domain ontology. The Table 23 which presents verificational part of the diagram from Example 1 has been supplemented with additional comments of how each verificational axiom or verificational query should be interpreted. The comments and the ontological background presented in Table 23 is also applicable to other examples.

Example 1

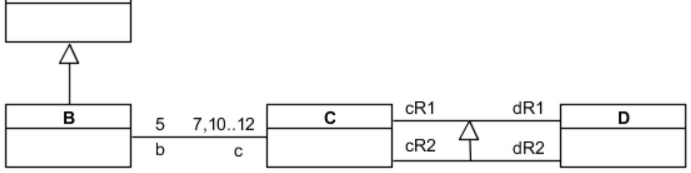


Figure 1. Example 1 of UML class diagram (see Tables 22, 23)

Table 22. Transformational part of UML class diagram from Example 1

Set of transformation axioms	Transformation rules
Transformation of UML Classes	
Declaration(**Class**(:A)) **Declaration**(**Class**(:B)) **Declaration**(**Class**(:C)) **Declaration**(**Class**(:D))	Table 2 **TR1**
Transformation of UML binary Associations between two different Classes	
Declaration(**ObjectProperty**(:b)) **Declaration**(**ObjectProperty**(:c)) **Declaration**(**ObjectProperty**(:$cR1$)) **Declaration**(**ObjectProperty**(:$dR1$)) **Declaration**(**ObjectProperty**(:$cR2$)) **Declaration**(**ObjectProperty**(:$dR2$))	Table 6 **TR1**
ObjectPropertyDomain(:b :C) **ObjectPropertyDomain**(:c :B) **ObjectPropertyDomain**(:$cR1$:D) **ObjectPropertyDomain**(:$dR1$:C) **ObjectPropertyDomain**(:$cR2$:D) **ObjectPropertyDomain**(:$dR2$:C)	Table 6 **TR2**
ObjectPropertyRange(:b :B) **ObjectPropertyRange**(:c :C) **ObjectPropertyRange**(:$cR1$:C) **ObjectPropertyRange**(:$dR1$:D) **ObjectPropertyRange**(:$cR2$:C) **ObjectPropertyRange**(:$dR2$:D)	Table 6 **TR3**
InverseObjectProperties(:b :c) **InverseObjectProperties**(:$cR1$:$dR1$) **InverseObjectProperties**(:$cR2$:$dR2$)	Table 6 **TR4**
Transformation of UML multiplicity of Association ends	
SubClassOf(:C **ObjectExactCardinality**(5 :b :B)) **SubClassOf**(:B **ObjectUnionOf**(**ObjectExactCardinality**(7 :c :C) **ObjectIntersectionOf**(**ObjectMinCardinality**(10 :c :C) **ObjectMaxCardinality**(12 :c :C)))) **SubClassOf**(:C **ObjectExactCardinality**(1 :$dR1$:D)) **SubClassOf**(:D **ObjectExactCardinality**(1 :$cR1$:C)) **SubClassOf**(:C **ObjectExactCardinality**(1 :$dR2$:D)) **SubClassOf**(:D **ObjectExactCardinality**(1 :$cR2$:C))	Table 9 **TR1**
FunctionalObjectProperty(:$dR1$) **FunctionalObjectProperty**(:$cR1$) **FunctionalObjectProperty**(:$dR2$) **FunctionalObjectProperty**(:$cR2$)	Table 9 **TR2**
Transformation of UML Generalization between Classes	
SubClassOf(:B :A)	Table 12 **TR1**
Transformation of UML Generalization between Associations	
SubObjectPropertyOf(:$cR2$:$cR1$) **SubObjectPropertyOf**(:$dR2$:$dR1$)	Table 13 **TR1**

Table 23. Verificational part of UML class diagram from Example 1

Verificational part of UML class diagram	Verification rules
Transformation of UML Classes	
If the domain ontology contains any **HasKey** axiom with any internal structure (OPE$_1$,..., DPE$_1$...) defined for :A, :B, :C or :D UML *Class*, the element should be UML structured *DataType* not UML *Class*. **HasKey**(:A (OPE$_1$...OPE$_{mA}$) (DPE$_1$...DPE$_{nA}$)) **HasKey**(:B (OPE$_1$...OPE$_{mB}$) (DPE$_1$...DPE$_{nB}$)) **HasKey**(:C (OPE$_1$...OPE$_{mC}$) (DPE$_1$...DPE$_{nC}$)) **HasKey**(:D (OPE$_1$...OPE$_{mD}$) (DPE$_1$...DPE$_{nD}$))	Table 2 **VR1**
Transformation of UML binary Associations between two different Classes	
If the domain ontology contains any of below defined **AsymmetricObjectProperty** axioms, the defined UML Association is incorrect. **AsymmetricObjectProperty**(:*b*) **AsymmetricObjectProperty**(:*c*) **AsymmetricObjectProperty**(:*cR1*) **AsymmetricObjectProperty**(:*dR1*) **AsymmetricObjectProperty**(:*cR2*) **AsymmetricObjectProperty**(:*dR2*)	Table 6 **VR1**
If the domain ontology contains any of the below-defined **ObjectPropertyDomain** axioms where class expression is different than the given UML *Class*, the *Association* is defined in the ontology but between different *Classes*, than it is specified on the diagram. **ObjectPropertyDomain**(:*b* CE), where CE$\neq$:C **ObjectPropertyDomain**(:*c* CE), where CE$\neq$:B **ObjectPropertyDomain**(:*cR1* CE), where CE$\neq$:D **ObjectPropertyDomain**(:*dR1* CE), where CE$\neq$:C **ObjectPropertyDomain**(:*cR2* CE), where CE$\neq$:D **ObjectPropertyDomain**(:*dR2* CE), where CE$\neq$:C	Table 6 **VR2**
If the domain ontology contains any of below-defined **ObjectPropertyRange** axioms where the class expression is different than the given UML *Class*, the *Association* is defined in the ontology but between different *Classes*. **ObjectPropertyRange**(:*b* CE), where CE$\neq$:B **ObjectPropertyRange**(:*c* CE), where CE$\neq$:C **ObjectPropertyRange**(:*cR1* CE), where CE$\neq$:C **ObjectPropertyRange**(:*dR1* CE), where CE$\neq$:D **ObjectPropertyRange**(:*cR2* CE), where CE$\neq$:C **ObjectPropertyRange**(:*dR2* CE), where CE$\neq$:D	Table 6 **VR3**
Transformation of UML multiplicity of Association ends	

If the verification query returns a number greater than 0, it means that UML multiplicity is in contradiction with the domain ontology (?vioInd lists individuals that cause the violation).
SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** :*b* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > 5)
SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** :*c* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > 12)
SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** :*dR1* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > 1)
	Table 9 **VR1**

SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** :*cR1* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > 1)
SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** :*dR2* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > 1)
SELECT ?**vioInd** (**count** (?**range**) **as** ?n)
WHERE { ?**vioInd** :*cR2* ?**range** } **GROUP BY** ?**vioInd**
HAVING (?n > 1)	

If the domain ontology contains **FunctionalObjectProperty** axiom specified for the association end which multiplicity is different from 1, the multiplicity is incorrect. **FunctionalObjectProperty**(:*b*) **FunctionalObjectProperty**(:*c*)	Table 9 **VR2**
If the domain ontology contains **SubClassOf** axiom, which specifies class expression with different multiplicity of the association ends than is derived from the UML class diagram, the multiplicity is incorrect. **SubClassOf**(:*C* CE), where CE ≠ **ObjectExactCardinality**(5 :*b* :*B*) **SubClassOf**(:*B* CE), where CE ≠ **ObjectUnionOf**(**ObjectExactCardinality**(7 :*c* :*C*) **ObjectIntersectionOf**(**ObjectMinCardinality**(10 :*c* :*C*) **ObjectMaxCardinality**(12 :*c* :*C*))) **SubClassOf**(:*C* CE), where CE ≠ **ObjectExactCardinality**(1 :*dR1* :*D*) **SubClassOf**(:*D* CE), where CE ≠ **ObjectExactCardinality**(1 :*cR1* :*C*) **SubClassOf**(:*C* CE), where CE ≠ **ObjectExactCardinality**(1 :*dR2* :*D*) **SubClassOf**(:*D* CE), where CE ≠ **ObjectExactCardinality**(1 :*cR2* :*C*)	Table 9 **VR3**
Transformation of UML Generalization between Classes	
If the domain ontology contains the defined **SubClassOf** axiom specified for *Classes*, which take part in the generalization relationship, the generalization relationship should be inverted on the diagram. **SubClassOf**(:*A* :*B*)	Table 12 **VR1**
Transformation of UML Generalization between Associations	
If the domain ontology contains the defined **SubObjectPropertyOf** axioms specified for *Association*, which take part in the generalization relationship, the generalization relationship should be inverted on the diagram. **SubObjectPropertyOf**(:*cR1* :*cR2*) **SubObjectPropertyOf**(:*dR1* :*dR2*)	Table 13 **VR1**

Example 2

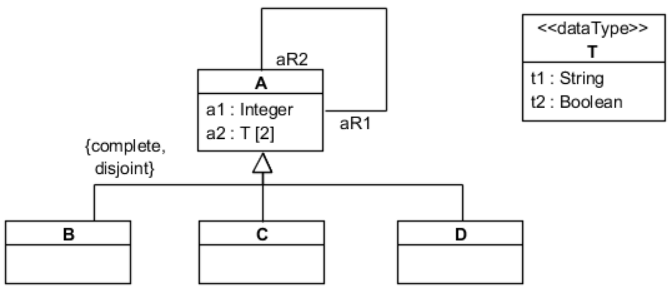


Figure 2. Example 2 of UML class diagram (see Tables 24–25)

Table 24. Transformational part of UML class diagram from Example 2

Set of transformation axioms	Transformation rules
Transformation of UML Classes	
Declaration(**Class**(:*A*))	Table 2 **TR1**
Declaration(**Class**(:*B*))	
Declaration(**Class**(:*C*))	
Declaration(**Class**(:*D*))	
Transformation of UML attributes	
Declaration(**DataProperty**(:*a1*))	Table 4 **TR1**
Declaration(**ObjectProperty**(:*a2*))	
DataPropertyDomain(:*a1* :*A*)	Table 4 **TR2**
ObjectPropertyDomain(:*a2* :*A*)	
DataPropertyRange(:*a1* **xsd**:**integer**)	Table 4 **TR3**
ObjectPropertyRange(:*a2* :*T*)	Table 18 **TR2**
Transformation of UML multiplicity of attributes	
SubClassOf(:*A* **ObjectExactCardinality**(2 :*a2* :*T*))	Table 5 **TR1**
Transformation of UML binary Association from the Class to itself	
Declaration(**ObjectProperty**(:*aR1*))	
Declaration(**ObjectProperty**(:*aR2*))	Table 7 **TR1**
ObjectPropertyDomain(:*aR1* :*A*)	
ObjectPropertyDomain(:*aR2* :*A*)	Table 7 **TR2**
ObjectPropertyRange(:*aR1* :*A*)	
ObjectPropertyRange(:*aR2* :*A*)	Table 7 **TR3**
InverseObjectProperties(:*aR1* :*aR2*)	Table 7 **TR4**
AsymmetricObjectProperty(:*aR1*)	Table 7 **TR5**
AsymmetricObjectProperty(:*aR2*)	
Transformation of UML multiplicity of Association ends	
SubClassOf(:*A* **ObjectExactCardinality**(1 :*aR1* :*A*))	Table 9 **TR1**
SubClassOf(:*A* **ObjectExactCardinality**(1 :*aR2* :*A*))	
FunctionalObjectProperty(:*aR1*)	Table 9 **TR2**
FunctionalObjectProperty(:*aR2*)	
Transformation of UML Generalization between Classes	
SubClassOf(:*B* :*A*)	Table 12 **TR1**
SubClassOf(:*C* :*A*)	
SubClassOf(:*D* :*A*)	
Transformation of UML GeneralizationSet with {complete, disjoint} constraints	
DisjointUnion(:*A* :*B* :*C* :*D*)	Table 15 **TR1**
Transformation of UML structured DataType	
Declaration(**Class**(:*T*))	Table 19 **TR1**
Declaration(**DataProperty**(:*t1*))	Table 19 **TR2**
Declaration(**DataProperty**(:*t2*))	
DataPropertyDomain(:*t1* :*T*)	Table 19 **TR3**
DataPropertyDomain(:*t2* :*T*)	
DataPropertyRange(:*t1* **xsd**:**string**)	Table 19 **TR4**
DataPropertyRange(:*t2* **xsd**:boolean)	Table 18 **TR1**
	Table 18 **TR3**
HasKey(:*T* ( ) (:*t1* :*t2*))	Table 19 **TR5**

Table 25. Verificational part of UML class diagram from Example 2

Verificational part of UML class diagram	Verification rules
Transformation of UML Classes	
HasKey(:A (OPE$_1$ …OPE$_{mA}$) (DPE$_1$ …DPE$_{nA}$))	Table 2 **VR1**
HasKey(:B (OPE$_1$ …OPE$_{mB}$) (DPE$_1$ …DPE$_{nB}$))	
HasKey(:C (OPE$_1$ …OPE$_{mC}$) (DPE$_1$ …DPE$_{nC}$))	
HasKey(:D (OPE$_1$ …OPE$_{mD}$) (DPE$_1$ …DPE$_{nD}$))	
Transformation of UML attributes	
DataPropertyDomain(:*a1* CE), where CE $\neq$ A	Table 4 **VR1**
ObjectPropertyDomain(:*a2* CE), where CE $\neq$ A	
DataPropertyRange(:*a1* DR), where	Table 4 **VR2**
DR $\neq$ **xsd:integer** **ObjectPropertyRange**(:*a2* CE), where	Table 18 **TR2**
CE $\neq$:T	
Transformation of UML multiplicity of attributes	
SELECT ?**vioInd** (**count** (?**range**) **as** ?n)	Table 5 **VR1**
WHERE { ?**vioInd** :*a2* ?**range** } **GROUP BY** ?**vioInd**	
HAVING (?n > 2)	
SubClassOf(:A CE), where	Table 5 **VR2**
CE $\neq$ **ObjectExactCardinality**(2 :*a2* :T)	
Transformation of UML binary Association from the Class to itself	
ObjectPropertyDomain(:*aR1* CE), where CE $\neq$:A	Table 7 **VR1**
ObjectPropertyDomain(:*aR2* CE), where CE $\neq$:A	
ObjectPropertyRange(:*aR1* CE), where CE $\neq$:A	Table 7 **VR2**
ObjectPropertyRange(:*aR2* CE), where CE $\neq$:A	
Transformation of UML multiplicity of Association ends	
SELECT ?**vioInd** (**count** (?**range**) **as** ?n)	Table 9 **VR1**
WHERE { ?**vioInd** :*aR1* ?**range** } **GROUP BY** ?**vioInd**	
HAVING (?n > 1)	
SELECT ?**vioInd** (**count** (?**range**) **as** ?n)	
WHERE { ?**vioInd** :*aR2* ?**range** } **GROUP BY** ?**vioInd**	
HAVING (?n > 1)	
SubClassOf(:A CE), where CE $\neq$ **ObjectExactCardinality**(1 :*aR1* :A)	Table 9 **VR3**
SubClassOf(:A CE), where CE $\neq$ **ObjectExactCardinality**(1 :*aR2* :A)	
Transformation of UML Generalization between Classes	
SubClassOf(:A :B)	Table 12 **VR1**
SubClassOf(:A :C)	
SubClassOf(:A :D)	
Transformation of UML GeneralizationSet with {complete, disjoint} constraints	
SubClassOf(:B :C)	Table 15 **VR1**
SubClassOf(:C :B)	
SubClassOf(:C :D)	
SubClassOf(:D :C)	
SubClassOf(:B :D)	
SubClassOf(:D :B)	

Transformation of UML structured DataType	
Check if the $:T$ class is specified in the domain ontology as a subclass (**SubClassOf** axiom) of any class expression, which does not have **HasKey** axiom defined.	Table 19 **VR1**

Example 3

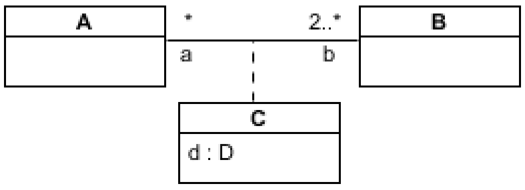


Figure 3. Example 3 of UML class diagram (see Tables 26–27)

Table 26. Transformational part of UML class diagram from Example 3

Set of transformation axioms	Transformation rules
Transformation of UML Classes	
Declaration(**Class**($:A$))	Table 2 **TR1**
Declaration(**Class**($:B$))	
Transformation of UML attributes	
Declaration(**ObjectProperty**($:d$))	Table 4 **TR1**
ObjectPropertyDomain($:d$ $:C$)	Table 4 **TR2**
ObjectPropertyRange($:d$ $:D$)	Table 4 **TR3**
Transformation of UML binary Associations between two different Classes	
Declaration(**ObjectProperty**($:a$))	Table 6 **TR1**
Declaration(**ObjectProperty**($:b$))	
ObjectPropertyDomain($:a$ **ObjectUnionOf**($:B$ $:C$))	Table 6 **TR2**
ObjectPropertyDomain($:b$ **ObjectUnionOf**($:A$ $:C$))	Table 10 **TR1**
ObjectPropertyRange($:a$ $:A$)	Table 6 **TR3**
ObjectPropertyRange($:b$ $:B$)	
InverseObjectProperties($:a$ $:b$)	Table 6 **TR4**
Transformation of UML multiplicity of Association ends	
SubClassOf($:A$ **ObjectMinCardinality**(2 $:b$ $:B$))	Table 9 **TR1**
Transformation of UML AssociationClass	
Declaration(**Class**($:C$))	Table 10 **TR2**
Declaration(**ObjectProperty**($:c$))	Table 10 **TR3**
ObjectPropertyDomain($:c$ **ObjectUnionOf**($:A$ $:B$))	Table 10 **TR4**
ObjectPropertyRange($:c$ $:C$)	Table 10 **TR5**

Table 27. Verificational part of UML class diagram from Example 3

Verificational part of UML class diagram	Verification rules
Transformation of UML Classes	
HasKey(:A (OPE$_1$ ...OPE$_m$) (DPE$_1$ ...DPE$_n$)) **HasKey**(:B (OPE$_1$ ...OPE$_m$) (DPE$_1$ ...DPE$_n$))	Table 2 **VR1**
Transformation of UML attributes	
ObjectPropertyDomain(:d CE), where CE $\neq$:C	Table 4 **VR1**
ObjectPropertyRange(:d CE), where CE $\neq$:D	Table 4 **VR2**
Transformation of UML binary Associations between two different Classes	
AsymmetricObjectProperty(:a) **AsymmetricObjectProperty**(:b)	Table 6 **VR1**
Transformation of UML multiplicity of Association ends	
FunctionalObjectProperty(:a) **FunctionalObjectProperty**(:b)	Table 9 **VR2**
SubClassOf(:A CE), where CE $\neq$ **ObjectMinCardinality**(2 :b :B) **SubClassOf**(:B CE), where CE = any explicitly specified multiplicity	Table 9 **VR3**
Transformation of UML AssociationClass	
HasKey(:C (OPE$_1$ ...OPE$_m$) (DPE$_1$ ...DPE$_n$))	Table 10 **VR1**
ObjectPropertyDomain(:a CE), where CE $\neq$ **ObjectUnionOf**(:B :C) **ObjectPropertyDomain**(:b CE), where CE $\neq$ **ObjectUnionOf**(:A :C) **ObjectPropertyDomain**(:c CE), where CE $\neq$ **ObjectUnionOf**(:A :B)	Table 10 **VR2**
ObjectPropertyRange(:c CE), where CE $\neq$:C	Table 10 **VR3**

8. Tool support for validation and automatic correction of UML class diagrams

The transformation and verification rules presented in Section 5 have been implemented in a tool. All the defined rules are proved to be fully implementable. As a result, the tool allows one to transform any UML class diagram built of different kinds of UML elements (listed in Section 5, and selected based on their importance from the perspective of pragmatics) to OWL 2 representation. In comparison to other available tools which allow transforming UML class diagrams to an OWL 2 representation, the range of the transformed constructions is wider as it benefits from the results of the conducted systematic literature review and its analysis, revision and extension.

Due to the fact that the tool is still under development, the following webpage has been created in which the tool with the installation instructions will be later accessible: https://sourceforge.net/projects/uml-class-diagrams-validation/

After the development and experiment phases are finished, the tool will be made available online.

The tool has been tested with a number of test cases aimed to determine whether the tool fully and correctly implemented the transformation and verification rules, as well as the validation method. At least one test case has been prepared for every normalization, transformation and verification rule. Additionally, a number of test cases have been prepared to cover popular assemblies of UML elements, e.g. an association from a class to itself, an association between two classes, two associations between two classes, two associations between three classes, etc. Each rule has been independently checked if it returns the expected result. In total, the number of test cases was as follows:

1. 80 test cases for ontology normalization rules,

2. 40 test cases for transformation rules,
3. 25 test cases for verification rules.

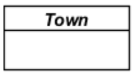
The tool passed all test cases.

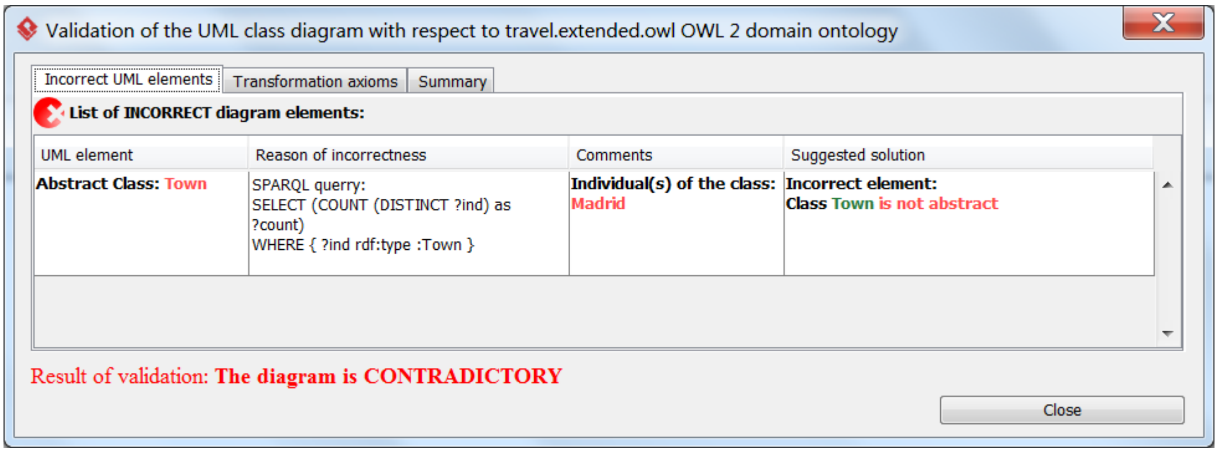
The implementation of the transformation and verification rules as well as the method of validation explained in [1], resulted in a functionality of the tool allowing for validation if a UML class diagram is compliant with the selected domain ontology. Furthermore, on the basis of the result of validation, the tool automatically generates ontology-based suggestions for diagram corrections. In [4], a few initial suggestions for diagram corrections have been presented. The initial list of suggestions has been revised and extended, and currently the tool automatically generates suggestions of two kinds:

1. What has to be corrected in the UML class diagram in order for the diagram not to be contradictory with the selected domain ontology (approximately 30 types of suggestions – one for violation of every verification rules plus one general suggestion listing incorrect UML elements if a transformation rule has caused the inconsistency in the domain ontology). This list of suggestions is reported by the tool for the modeller and strongly advised his or her attention (Example 4 and 5).
2. Based on the domain ontology of what might be additionally included in the class diagram (9 types of suggestions). Whether or not to consider this list of proposed suggestions is for the modeller to decide. Depending on the specific requirements, the suggestions may be incorporated in the diagram (Example 6).

Example 4

Table 28. Example of what has to be corrected in the diagram based on the ontology: abstract class verification

Suggestion: the class is not abstract	
Axiom(s) in the OWL domain ontology	**Declaration**(**Class**(:*Town*)) **ClassAssertion**(:*Town* :*Madrid*)
Element on the UML class diagram	
Result of validation:	



Validation of the UML class diagram with respect to travel.extended.owl OWL 2 domain ontology

List of INCORRECT diagram elements:

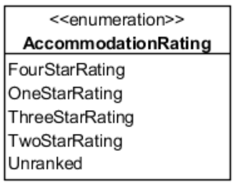
UML element	Reason of incorrectness	Comments	Suggested solution

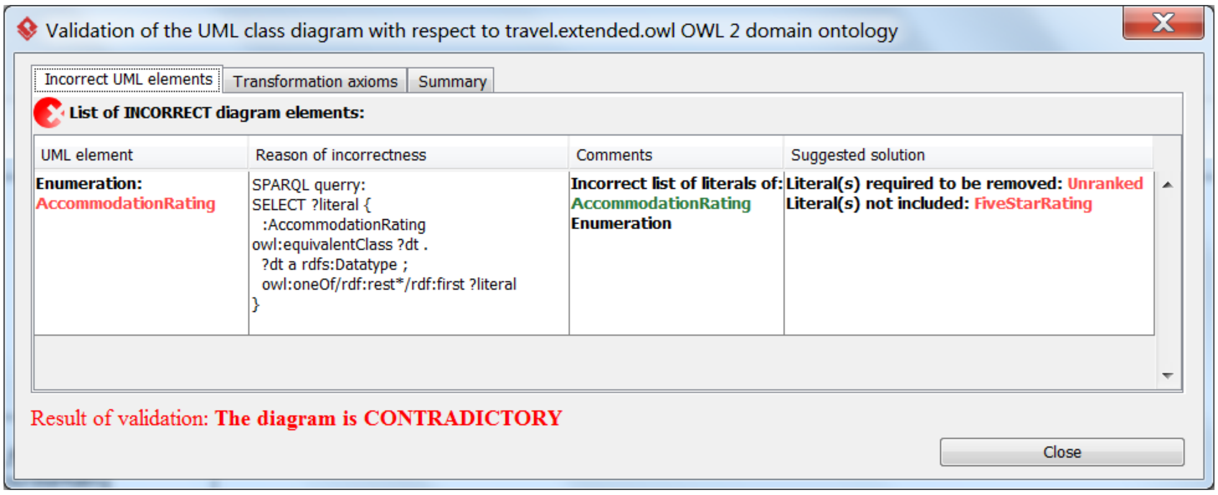
Abstract Class: Town	SPARQL querry:
	SELECT (COUNT (DISTINCT ?ind) as ?count)
	WHERE { ?ind rdf:type :Town }
		Individual(s) of the class: Madrid	**Incorrect element:**
			Class Town is not abstract

Result of validation: **The diagram is CONTRADICTORY**

Example 5

Table 29. Example of what has to be corrected in the diagram based on the ontology: enumeration verification

Suggestion: the enumeration is incorrectly defined	
Axiom(s) in the OWL domain ontology	**DatatypeDefinition**(:*AccommodationRating* **DataOneOf**("OneStarRating" "TwoStarRating" "ThreeStarRating" "FourStarRating" "FiveStarRating"))
Element on the UML class diagram	 <<enumeration>> **AccommodationRating** FourStarRating OneStarRating ThreeStarRating TwoStarRating Unranked
Result of validation:	



Validation of the UML class diagram with respect to travel.extended.owl OWL 2 domain ontology

List of INCORRECT diagram elements:

UML element	Reason of incorrectness	Comments	Suggested solution

Enumeration: AccommodationRating	SPARQL querry:
	SELECT ?literal {
	:AccommodationRating
	owl:equivalentClass ?dt .
	?dt a rdfs:Datatype ;
	owl:oneOf/rdf:rest*/rdf:first ?literal
	}
		Incorrect list of literals of: AccommodationRating **Enumeration**	**Literal(s) required to be removed:** Unranked
			Literal(s) not included: FiveStarRating

Result of validation: **The diagram is CONTRADICTORY**

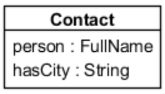
Example 6

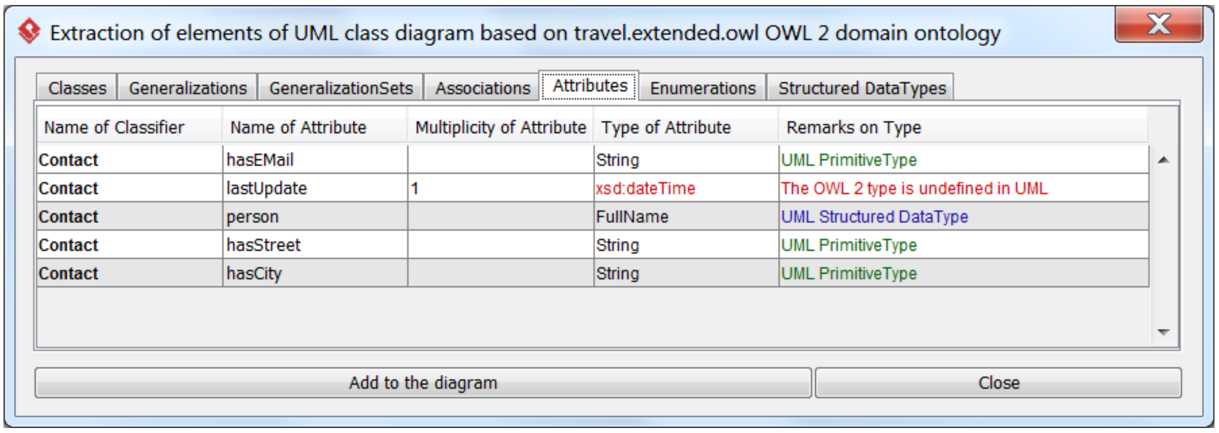
Table 30. Example of what may be incorporated in the diagram based on the ontology: attribute verification

Suggestion: Insert missing attribute, missing type of attribute or missing multiplicity of attribute	
Axiom(s) in the OWL domain ontology	**Declaration**(**Class**(:*Contact*)) **ObjectPropertyDomain**(:*person* :*Contact*) **ObjectPropertyRange**(:*person* :*FullName*) **Declaration**(**Class**(:*FullName*)) **DataPropertyDomain**(:*firstName* :*FullName*) **DataPropertyRange**(:*firstName* **xsd:string**) **DataPropertyDomain**(:*secondName* :*FullName*) **DataPropertyRange**(:*secondName* **xsd:string**) **HasKey**(:*FullName* ( ) (:*firstName* :*secondName*)) **Declaration**(**DataProperty**(:*hasEMail*)) **DataPropertyDomain**(:*hasEMail* :*Contact*) **DataPropertyRange**(:*hasEMail* **xsd:string**)

Declaration(**DataProperty**(:*hasStreet*))
DataPropertyDomain(:*hasStreet* :*Contact*)
DataPropertyRange(:*hasStreet* **xsd**:**string**)
Declaration(**DataProperty**(:*hasCity*))
DataPropertyDomain(:*hasCity* :*Contact*)
DataPropertyRange(:*hasCity* **xsd**:**string**)
Declaration(**DataProperty**(:*lastUpdate*))
DataPropertyDomain(:*lastUpdate* :*Contact*)
DataPropertyRange(:*lastUpdate* **xsd**:dateTime)
SubClassOf(:*Contact* **DataExactCardinality**(1 :*lastUpdate*))

Element on the UML class diagram





Name of Classifier	Name of Attribute	Multiplicity of Attribute	Type of Attribute	Remarks on Type
Contact	hasEMail		String	UML PrimitiveType
Contact	lastUpdate	1	xsd:dateTime	The OWL 2 type is undefined in UML
Contact	person		FullName	UML Structured DataType
Contact	hasStreet		String	UML PrimitiveType
Contact	hasCity		String	UML PrimitiveType

Legend:
white rows – suggestions of UML elements which might be included in the class diagram
grey rows – UML elements already included in the class diagram

9. Conclusions

The paper presents rules for transforming UML class diagrams to their OWL 2 representations. All the static elements of UML class diagrams commonly used in business or conceptual modelling have been considered. The vast majority of the elements can be fully transformed to OWL 2 constructs. The presented transformation rules result from an in-depth analysis and extension of the state-of-the-art transformation rules identified through a systematic literature review. In total, 41 transformation rules have been described (not counting our complementation to the rules of disjointness presented in Section 6). 25 transformation rules have been directly extracted from the literature, 8 rules originate in the literature but have been extended by us in order to reflect the semantics of UML elements in OWL more precisely, and 8 transformation rules are our new propositions.

In addition to the transformation rules, we have defined all the presented verification rules (26 in total). The verification rules are aimed at checking the compliance of the OWL representation of UML class diagram with the given OWL domain ontology. The described transformation and verification rules are crucial in the method of semantic validation of UML class diagrams [1]. The approach validates automatically if a selected class diagram is compliant with the selected OWL 2 domain ontology.

The developed method and the tool are a pragmatic attempt of bringing together the differences in the philosophy of UML and OWL 2 languages. In order to make the process automatic, the tool has been supplemented with all the transformation and verification rules, and has been tested with a wide range of test cases. The inclusions to the tool are the result of pragmatic thinking. For example, the implementation allowed observing that it is worth extending the tool so that it automatically generates ontology-based suggestions for diagram corrections.

The tool already offers a range of new possibilities for practical application of domain ontologies. However, as a consequence, the proposed approach creates a need for greater involvement of domain ontologies in modeling.

The research background of our considerations can be supported by other publications, e.g. [35–37]. The potential of reusing domain ontologies for the purpose of validation is promising and may help the modelers through automation. The choice of OWL is justified by the growing number of the already created ontologies in this language. For future work, the development of the tool is planned to be finished soon. The next step of work is preparation of experiment aimed at validation of the tool and the method in practice. The experiment is aimed to state the practicality of our proposal.

References

[1] M. Sadowska and Z. Huzar, “Semantic validation of UML class diagrams with the use of domain ontologies expressed in OWL 2,” in *Software Engineering: Challenges and Solutions*. Springer International Publishing, 2016, pp. 47–59.

[2] *Unified Modeling Language, Version 2.5*, OMG, 2015. [Online]. http://www.omg.org/spec/UML/2.5

[3] *OWL 2 Web Ontology Language Document Overview (Second Edition)*, W3C, 2012. [Online]. https://www.w3.org/TR/owl2-overview/

[4] M. Sadowska, “A prototype tool for semantic validation of UML class diagrams with the use of domain ontologies expressed in OWL 2,” in *Towards a Synergistic Combination of Research and Practice in Software Engineering*. Springer International Publishing, 2017, pp. 49–62.

[5] M. Sadowska and Z. Huzar, “The method of normalizing OWL 2 DL ontologies,” *Global Journal of Computer Science and Technology*, Vol. 18, No. 2, 2018, pp. 1–13.

[6] A. Korthaus, “Using UML for business object based systems modeling,” in *The Unified Modeling Language*. Physica-Verlag HD, 1998, pp. 220–237.

[7] H.E. Eriksson and M. Penker, *Business Modeling With UML: Business Patterns at Work*. New York, USA: John Wiley & Sons inc., 2000.

[8] E.D. Nitto, L. Lavazza, M. Schiavoni, E. Tracanella, and M. Trombetta, “Deriving executable process descriptions from UML,” in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE ’02. New York, NY, USA: ACM, 2002, pp. 155–165.

[9] C. Fu, D. Yang, X. Zhang, and H. Hu, “An approach to translating OCL invariants into OWL 2 DL axioms for checking inconsistency,” *Automated Software Engineering*, Vol. 24, No. 2, 2017, pp. 295–339.

[10] B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” Keele University & University of Durham, EBSE Technical Report EBSE 2007-01, 2007.

[11] X. Zhou, Y. Jin, H. Zhang, S. Li, and X. Huang, “A map of threats to validity of systematic literature reviews in software engineering,” in *23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2016, pp. 153–160.

[12] Z. Xu, Y. Ni, W. He, L. Lin, and Q. Yan, “Automatic extraction of OWL ontologies from UML class diagrams: a semantics-preserving approach,” *World Wide Web*, Vol. 15, No. 5, 2012, pp. 517–545.

[13] Z. Xu, Y. Ni, L. Lin, and H. Gu, “A semantics-preserving approach for extracting OWL ontologies from UML class diagrams,” in *Database Theory and Application*, ser. Communications in Computer and Information Science. Berlin, Heidelberg: Springer, 2009, pp. 122–136.

[14] M. Mehrolhassani and A. Elçi, “Developing ontology based applications of semantic web using UML to OWL conversion,” in *The Open Knowlege Society. A Computer Science and Information Systems Manifesto*, ser. Communications in Computer and Information Science. Berlin, Heidelberg: Springer, 2008, pp. 566–577.

[15] O. El Hajjamy, K. Alaoui, L. Alaoui, and M. Bahaj, “Mapping UML to OWL2 ontology,” *Journal of Theoretical and Applied Information Technology*, Vol. 90, No. 1, 2016, pp. 126–143.

[16] C. Zhang, Z.R. Peng, T. Zhao, and W. Li, “Transformation of transportation data models from Unified Modeling Language to Web Ontology Language,” *Transportation Research Record: Journal of the Transportation Research Board*, Vol. 2064, No. 1, 2008, pp. 81–89.

[17] J. Zedlitz, J. Jörke, and N. Luttenberger, “From UML to OWL 2,” in *Knowledge Technology*, ser.

Communications in Computer and Information Science. Berlin, Heidelberg: Springer, 2012, pp. 154–163.

[18] A.H. Khan and I. Porres, "Consistency of UML class, object and statechart diagrams using ontology reasoners," *Journal of Visual Languages & Computing*, Vol. 26, 2015, pp. 42–65.

[19] A.H. Khan, I. Rauf, and I. Porres, "Consistency of UML class and statechart diagrams with state invariants," in *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development*, S. Hammoudi, L.F. Pires, J. Filipe, and R.C. das Neves, Eds., Vol. 1. SciTePress Digital Library, 2013, p. 1–11.

[20] J. Zedlitz and N. Luttenberger, "Transforming between UML conceptual models and OWL 2 ontologies," in *Terra Cognita 2012 Workshop*, Vol. 6, 2012, p. 15.

[21] W. Xu, A. Dilo, S. Zlatanova, and P. van Oosterom, "Modelling emergency response processes: Comparative study on OWL and UML," in *Proceedings of the Joint ISCRAM-CHINA and GI4DM Conference*, Harbin, China, 2008, pp. 493–504.

[22] N. Gherabi and M. Bahaj, "A new method for mapping UML class into OWL ontology," *International Journal of Computer Applications Special Issue on Software Engineering, Databases and Expert Systems*, Vol. SEDEXS, No. 1, 2012, pp. 5–9. [Online]. https://research.ijcaonline.org/sedex/number1/sedex1002.pdf

[23] H.S. Na, O.H. Choi, and J.E. Lim, "A method for building domain ontologies based on the transformation of UML models," in *Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*, D.K. Baik, D. Primeaux, N. Ishii, and R. Lee, Eds. IEEE, 2006, pp. 332–338.

[24] M. Bahaj and J. Bakkas, "Automatic conversion method of class diagrams to ontologies maintaining their semantic features," *International Journal of Soft Computing and Engineering*, Vol. 2, No. 6, 2013, pp. 65–69.

[25] A. Belghiat and M. Bourahla, "Transformation of UML models towards OWL ontologies," in *2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*, 2012, pp. 840–846.

[26] S. Höglund, A.H. Khan, Y. Liu, and I. Porres, "Representing and validating metamodels using OWL 2 and SWRL," in *Proceedings of the 9th Joint Conference on Knowledge-Based Software Engineering*, 2010.

[27] K. Kiko and C. Atkinson, "A detailed comparison of UML and OWL," University of Mannheim, Fakultät für Mathematik und Informatik, Lehrstuhl für Softwaretechnik, Tech. Rep. TR-2008-004, 2008.

[28] J. Zedlitz and N. Luttenberger, "Data types in UML and OWL-2," in *The Seventh International Conference on Advances in Semantic Processing*, 2013, pp. 32–35.

[29] J. Zedlitz and N. Luttenberger, "Conceptual modelling in UML and OWL-2," *International Journal on Advances in Software*, Vol. 7, No. 1 & 2, 2014, pp. 182–196.

[30] *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*, W3C, 2012. [Online]. https://www.w3.org/TR/owl2-syntax/

[31] *OWL 2 Web Ontology Language New Features and Rationale (Second Edition)*, W3C, 2012. [Online]. https://www.w3.org/TR/owl2-new-features/

[32] N. Noy and A. Rector, *Defining N-ary Relations on the Semantic Web*, W3C, 2006. [Online]. https://www.w3.org/TR/swbp-n-aryRelations/

[33] *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*, W3C, 2012. [Online]. https://www.w3.org/TR/xmlschema11-2/

[34] *OWL 2 Web Ontology Language Primer (Second Edition)*, W3C, 2012. [Online]. https://www.w3.org/TR/owl2-primer/

[35] I. Dubielewicz, B. Hnatkowska, Z. Huzar, and L. Tuzinkiewicz, "Domain modeling in the context of ontology," *Foundations of Computing and Decision Sciences*, Vol. 40, No. 1, 2015, pp. 3–15. [Online]. https://content.sciendo.com/view/journals/fcds/40/1/article-p3.xml

[36] B. Hnatkowska, Z. Huzar, L. Tuzinkiewicz, and I. Dubielewicz, "A new ontology-based approach for construction of domain model," in *Intelligent Information and Database Systems*, N.T. Nguyen, S. Tojo, L.M. Nguyen, and B. Trawiński, Eds. Cham: Springer International Publishing, 2017, pp. 75–85.

[37] I. Dubielewicz, B. Hnatkowska, Z. Huzar, and L. Tuzinkiewicz, "Domain modeling based on requirements specification and ontology," in *Software Engineering: Challenges and Solutions*, L. Madeyski, M. Śmiałek, B. Hnatkowska, and Z. Huzar, Eds. Cham: Springer International Publishing, 2017, pp. 31–45.