An Object Oriented Approach for Ontology Modeling and Reasoning

Ouassila Labbani Narsis[a] and Christophe Nicolle[b]
CIAD UMR 7533 Laboratory, Univ. Bourgogne Franche-Comté, 21000 Dijon, France

Keywords: Ontology Engineering, Reasoning, UML Modeling, OCL Constraints Verification.

Abstract: In Industry, IT personnel commonly manipulate the object paradigm for software engineering. Industry 4.0 with its profusion of data requires the implementation of a knowledge engineering approach, breaking with data processing habits. The way from an object practice to an ontological vision is a gap for most of the staff, and transition from a business expert model to software implementation is the source of many errors and delays. Moreover, in ontology engineering, namely that domain experts have significant difficulties learning ontology languages and correctly using them. To overcome this problem, this paper presents the first results of a modeling and reasoning approach based on the object paradigm by combining UML modeling and reasoning by constraint resolution with OCL. The ontology produced by domain experts can then be checked and improved by a UML/OCL approach which is easily understandable and often more familiar to engineers.

1 INTRODUCTION AND MOTIVATION

Ontology Engineering is a subfield of artificial intelligence. It provides a set of activities that concerns methodologies, tools, and languages for building knowledge models or ontologies in the form of concepts and their relationships (Corcho et al., 2004; Mizoguchi, 2004). Deviating from its original philosophical meaning, ontology in computer science denotes a formal explicit description of concepts and their relationships in a specific domain. The role of ontologies is to capture domain knowledge generically and provides a shared and commonly agreed-upon understanding of a domain (Maedche et al., 2000).

Various ontology research activities and development methodologies are available. They deal with the process and methodological aspects of ontology engineering that enable the construction of ontologies from scratch at the knowledge level (Fernández-López et al., 1997). Despite all these advancements, ontology engineering is still a difficult process, and many difficulties remain to be solved. Potential ontology developers are facing several challenges, in particular a steep learning curve and the difficulty of modeling, as well as choosing the right ontology tools and dealing with technological limitations (Tudorache, 2020).

Ontology engineering is then a labor-intensive effort and the cost of entry for a developer into the ontology engineering area is high (Stadnicki et al., 2020). As of today, ontologies are mainly created by highly specialized ontology engineers, and there is no tool support for the methodology. They are mainly based on approaches, criteria, and development guides related to the skills of experts in a particular field, and are usually influenced by the targeted application. Developing, understanding, and using an ontology becomes a complex task and needs additional skills especially for IT people coming from software engineering and object-oriented programming. This places strong requirements on the competence and experience of the ontology engineering team which in turn represents a major hurdle to the industrial adoption of semantic technologies (Lupp et al., 2020). It is then important to propose new approaches inspired by the main activities performed in the software engineering development process to help users to create formalized knowledge, qualify their coherence, and be able to adapt and enrich them according to their needs and constraints. Our experiences in developing ontology-based modeling and reasoning solutions in an industrial environment are hampered by a technical ignorance of this knowledge engineering approach where the vast majority of the company's IT personnel have been trained in object

[a] https://orcid.org/0000-0001-5521-0126
[b] https://orcid.org/0000-0002-8118-5005

Narsis, O. and Nicolle, C.
An Object Oriented Approach for Ontology Modeling and Reasoning.
DOI: 10.5220/0010642800003064
In *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2021) - Volume 2: KEOD*, pages 128-135
ISBN: 978-989-758-533-3; ISSN: 2184-3228

programming and UML modeling for software engineering.

In software engineering, *Object-Oriented Modeling* (OOM) has several interesting characteristics in the representation and conceptualization of knowledge (Engels and Groenewegen, 2000). It allows the description of a collection of objects to describe similar individuals, and relationships or associations to group similar links between these objects. In this field, the Unified Modeling Language (UML[1]) diagrams are created and used to visualize and conceptualize the design of a system. Moreover, UML modeling, which a user can more readily understand, is widely used in the industry on large projects to communicate and design processes.

Several research works adapt existing object-oriented development methodologies, in particular UML modeling, for the task of ontology development (Mkhinini et al., 2020). Their objectives are mainly to propose graphical modeling of ontologies using UML diagrams, to unify knowledge representation models, or to validate UML diagrams. However, no study addresses the perspective of using object-oriented modeling for reasoning purposes to infer new knowledge and help users to better understand and adapt the ontology reasoning process to their needs and constraints. We believe that this lack is mainly related to the fact that the ontology is theoretically found on Description Logic (DL) allowing automated reasoning, which is not the case for UML modeling. To deal with this limitation, we propose to add a logic capability to object-oriented modeling by using Object Constraint Language (OCL[2]). In this paper, we discuss how OCL constraints verification on the UML model can be used to, on the one hand, verify the consistency of the ontology, and on the other hand, infer new knowledge according to the verification results of OCL constraints related to the semantics of ontology's axioms. The results can help engineers better understand their models, easily manage the reasoning process, detect possible inconsistencies, and provide solutions.

2 COMBINING OBJECT ORIENTED MODELING AND ONTOLOGY ENGINEERING

Ontology development has many parallels with software development for knowledge representation. Object-oriented modeling, and in particular UML models, are often accepted as a practical ontology specification, mostly because of their ability to model knowledge in a natural way, and their widespread use in industry. The similarity between these two languages has been studied in different research work combining UML and ontologies (Mkhinini et al., 2020). In this work, UML is regarded as a suitable candidate for ontology modeling. In particular, UML class diagrams provide a rich notation for defining classes, their attributes, and the relationships between them. Compared to existing research work combining UML modeling and ontologies, few studies propose the integration of OCL language and even less for a reasoning objective. In (Cranefield and Purvis, 1999; Cranefield et al., 2001), Cranefield and Purvis discuss the potential for reasoning about ontologies expressed using UML with logical expressions in OCL, and present it as an important subject for future research. In (Cranefield et al., 2001), authors compared Description Logic (DL) and UML in terms of different categories of reasoning. They affirm that is possible to equip UML design tools with similar capabilities, and further research is needed to clarify what types of inference it would be desirable and possible to support for ontologies represented in UML. In many cases, we believe that OCL constraints can be regarded as extra detail specifying how systems that implement the ontology should behave, and new knowledge can be derived from UML models by reasoning about their contents. To our knowledge, no effort or research work has been done to prove or implement this perspective, and few research work exists on the use of OCL language in ontology development. In (Fu et al., 2017) for example, we can find an inverse approach in which authors propose to translate OCL invariant into OWL2 DL axioms for checking inconsistency of UML models. An automatic approach to analyze the consistency and satisfiability of UML statechart diagrams with OCL state invariants using logic ontology reasoners is presented in (Khan and Porres, 2015). Some research work is interested in proposing a solution to interchanging rules and queries between OCL language and semantic web language (Parreiras and Staab, 2010; Hafeez et al., 2018; Milanović et al., 2006; Timm and Gannod, 2007), and others mention OCL language to represent some attribute constraints to facilitate the mapping from knowledge model to software model (Wang and Chan, 2001; Baclawski et al., 2001).

[1]https://www.omg.org/spec/UML/About-UML

[2]https://www.omg.org/spec/OCL/About-OCL

3 ONTOLOGY REASONING USING OCL CONSTRAINTS VERIFICATION

Ontology reasoner or inference engine is mostly used to derive new facts from the existing knowledge and can be used to verify the logical consistency of the ontology model. There are several types of reasoner tools proposed by many researchers (Abburu, 2012). Some of the most popular reasoners are Pellet (Parsia and Sirin, 2004), HermiT (Shearer et al., 2008), and FACT++ (Horrocks, 1998). The inference rules are commonly specified by means of a Description Logic (DL) to perform reasoning tasks about individuals, classes, and properties.

In this section, we describe our approach using the object-oriented paradigm, in particular, UML/OCL modeling, to verify the consistency of an ontology and infer new knowledge. The idea is to automatically generate the UML model from ontology description with OCL constraints related to the semantics of ontology axioms[3]. Our approach allows, on the one hand, to ensure the consistency of the ontology, and on the other hand, to provide a reasoning system able to generate and integrate new knowledge according to OCL constraint verification results on the associated UML model. The obtained results will be used to help ontology designers by alerting them to possible inconsistencies or knowledge lacks. This will increase the level of expressiveness and decidability of the ontology, and allow it to reach the closed world assumption (CWA) necessary for the reasoning process in the case of critical systems. The purpose of this work is to provide a configurable tool allowing to assist the design, the enrichment, and the consistency verification of an ontology using UML modeling and OCL constraint verification. The main steps of our approach are described in figure 1. We have mainly five tasks in our reasoning process:

3.1 Ontology Transformation

The objective of this task is to automatically generate the UML/OCL model related to the input ontology. This transformation process allows the definition of the class diagram and OCL constraints related to the ontology terminological components and axioms semantics (TBox level), and object diagram representing individuals defined in the ontology and their relationships (ABox level). An example of transformation rules is described in section 4.

In UML modeling, OCL constraints verification must be performed on instances of the model described by a UML object diagram. To ensure the verification of all generated OCL constraints and not be limited by individuals defined in the ontology, and also to take into account unpopulated ontologies, we propose to automatically generate an object diagram with an instance of each class of the ontology.

3.2 OCL Constraints Verification

After generating the UML/OCL model, constraints verification will be applied to the object diagram to find unverified constraints. There are several UML/OCL tools allowing efficient and automatic constraints verification and analysis (González and Cabot, 2014; Richters and Gogolla, 2002). In our approach, we propose to use the UML Specification Environment USE[4] which supports UML modeling and OCL constraints description and validation (Gogolla et al., 2007). The purpose of this task is to provide the verification result of each OCL constraint, as well as the list of individuals concerned by this verification.

3.3 Analysis of Constraints Verification Results

In this task, we proceed to the extraction and analysis of unverified constraints with concerned individuals. In the object-oriented paradigm, if an OCL constraint is not verified, this may be related to either an error in the design of the model or a knowledge lack in the ontology description. This is the closed world assumption (CWA) which asserts that a statement is true only if it is known to be true and that knowledge of a system is known to be complete. The result of this analysis task will be used to detect inconsistency or missing knowledge that can be generated and integrated into the ontology.

3.4 Results Generation

Depending on the unverified constraint type and the semantics of ontology axioms, this task allows detecting either the presence of an inconsistency in the ontology and then alert the designer, or a lack of knowledge that can be automatically generated and integrated into the ontology using our transformation process and after expert validation.

[3]In this paper, we only present the first results on atomic classes and object properties.

[4]https://sourceforge.net/projects/useocl

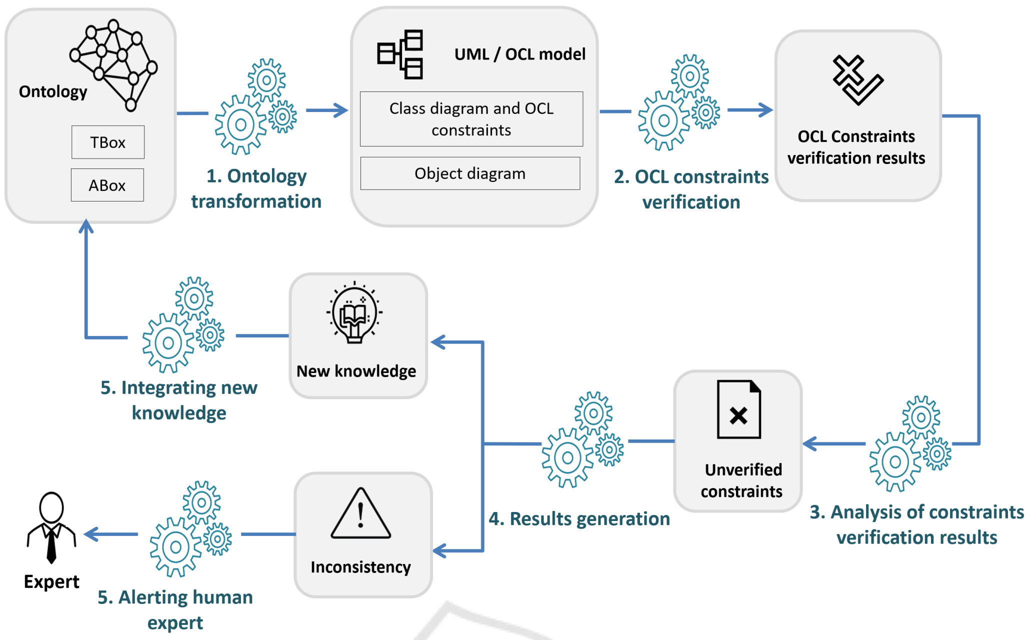

Ontology
TBox
ABox
1. Ontology transformation
UML / OCL model
Class diagram and OCL constraints
Object diagram
2. OCL constraints verification
OCL Constraints verification results
3. Analysis of constraints verification results
Unverified constraints
4. Results generation
New knowledge
5. Integrating new knowledge
Inconsistency
5. Alerting human expert
Expert

Figure 1: Main steps of the UML/OCL reasoning process.

3.5 Integrating New Knowledge or Alerting Human Expert

According to the result of the previous task, this step consists of displaying a warning message to alert the designer in the case of inconsistency detection or displaying the list of new knowledge that can be integrated into the ontology. This process is iterative and will continue until all OCL constraints are verified and no new knowledge to generate.

4 TRANSFORMATION EXAMPLE OF ONTOLOGY IN UML/OCL MODEL

This section presents a simple example of transformation rules of OWL2 axioms to the UML/OCL model for reasoning purpose. In our approach, we use UML class diagram and OCL constraints to model the domain of interest as a set of terminological axioms (TBox statements); and UML object diagram to model individuals as a set of assertional axioms (ABox statements), and on which OCL constraints will be verified to prove the consistency of the model and infer new knowledge.

All transformation rules are based on the semantics of OWL2 axioms according to the W3C structural specification and fonctional-style syntax[5]. In our study, we use the Protégé tool[6] for ontology description, and the UML Specification Environment USE[7] to support UML modeling and OCL constraints verification.

Our transformation process is mainly based on two steps:

1. Transforming each element of the ontology to its equivalent in the UML model with the associated OCL constraints.

2. Adding OCL constraints describing the semantics of each axiom and its relation to other axioms in the model.

For the subclass axiom `SubClassOf` $(C_i\ C_j)$[8] for example, which means that C_i is a subclass of C_j, we, first, translate the axiom on its equivalent in the UML model. In this case, we propose to use the UML generalization to express subclass relations and hierarchy. The subclass axiom is then represented in the UML model by a generalization link between C_i and C_j, and the following OCL constraint to ensure this property:

context C_i inv:
self.oclIsKindOf(C_j)

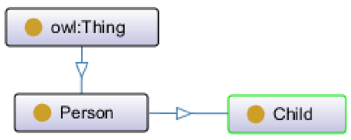
[5]https://www.w3.org/TR/owl2-syntax/
[6]https://protege.stanford.edu/
[7]https://sourceforge.net/projects/useocl
[8]https://www.w3.org/TR/owl2-syntax/#Subclass_Axioms

Table 1 gives a simple example of the application of this first transformation step.

Table 1: First transformation step example.

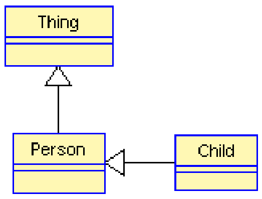
Input OWL2 model:

```
SubClassOf(Child Person)
```



Output UML/OCL model:

```
context Child inv:
self.oclIsKindOf(Person)
```



Second, we generate for this subclass axiom all OCL constraints allowing us to ensure its semantics in relation to the other ontology's axioms. For example, in relation to the equivalent classes axiom EquivalentClasses(C_1 ... C_n), which states that all of the classes C_i, $1 \leq i \leq n$, are semantically equivalent to each other, if C_i is a subclass of C_j, and if an equivalent classes axiom is defined between C_j and C_k, then C_i is also a subclass of C_k, and a subclass of each class defined as equivalent to C_k. To ensure this property, we add the following OCL constraints:
context C_i inv:
self.oclIsKindOf(C_k),
and:
context C_i inv:
self.oclIsKindOf(C_m), for each C_m equivalent to C_k.

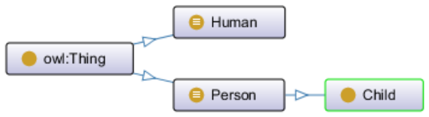
Table 2 gives a simple example of the application of this second transformation step. In this example, we show only the UML model and OCL constraints related to the subclass axiom. Those related to the equivalent classes axiom are not described in the example.

The verification of generated OCL constraints allows the detection of errors or missing information in the ontology if C_i is not defined as a subclass of C_k (or a subclass of C_m in the case of equivalent classes). Based on this verification result, we can automatically generate the associated new knowledge that can be added to the ontology. This new knowledge is

Table 2: Second transformation step example.

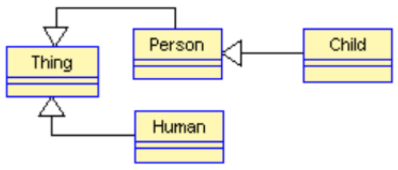
Input OWL2 model:

```
SubClassOf(Child Person)
EquivalentClasses (Human Person)
```



Output UML/OCL model:

```
context Child inv:
self.oclIsKindOf(Person)
context Child inv:
self.oclIsKindOf(Human)
```



related to the following axioms: SubClassOf(C_i C_k), and SubClassOf(C_i C_m), for each C_m equivalent to C_k. In our example (Table 2), the second OCL constraint verifying that any individual of type Child must also be of type Human will not be verified since in the automatically generated UML model, the Child class does not inherit from the Human class. This verification result will be used to automatically generate the new knowledge SubClassOf(Child Human) that can be integrated into the ontology.

We note that in our transformation process, and to resolve the semantic difference between UML and OWL2 languages, we proposed in some specific cases adapted transformation rules. For example, the equivalent classes axiom between C_i and C_j means that every instance of C_i class is also an instance of C_j class, which is equivalent to the following two axioms: SubClassOf(C_i C_j), SubClassOf(C_j C_i). However, in UML modeling, it is not possible to define cycle in generalization hierarchy. We cannot express in the same model that C_i is a subclass of C_j, and C_j is a subclass of C_i. To address this limitation, we proposed in the transformation process to add a new generic class C_iC_j that inherits from C_i and C_j. In this case, each individual in the ontology of type C_i or C_j will be implicitly transformed and considered in the UML model as an instance of the new generic class C_iC_j. Another specific case is related to the transformation of object properties. We proposed to represent each object property by an association class instead of a simple association link. This allows to easily associate OCL constraints to the object property

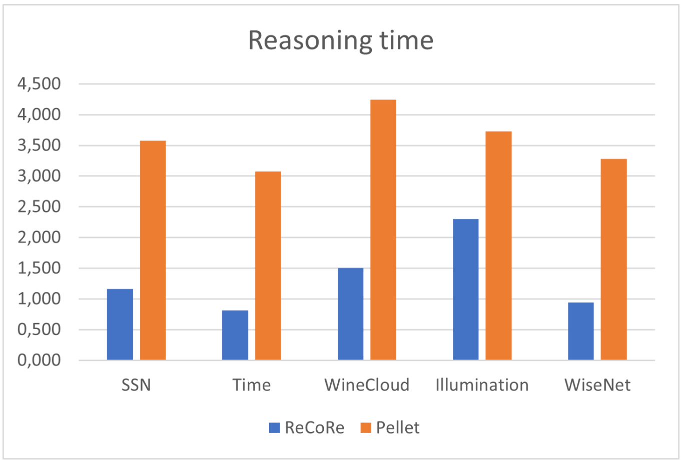


Figure 2: Reasoning time comparison.

that are necessary for the generation of new knowledge related to its axioms. We note that the obtained UML model can be simplified by removing specific elements like generic classes and association classes to be more intuitive and easy to understand for users. Due to the page limit, we can't detail all transformation rules and specific adaptations we have done to deal with the modeling difference between semantic paradigm and object-oriented paradigm.

5 IMPLEMENTATION AND EXPERIMENTATION

To test our approach, we have implemented ReCoRe (Reasoning by Constraint Resolution) application to automate all steps described in figure 1, and we tested its application for several ontologies such as: SSN[9], Time[10], Winecloud[11], Illumination[12], and WiseNet[13].

The number of inferred triples generated by our application is summarized in table 3 and figure 3. We have compared all inferred triples generated by our application with that generated by Pellet reasoner[14], and it matches for studied axioms related to atomic classes and object properties. We have also compared time reasoning and the obtained results are interesting as shown in table 3 and figures 2. We are rather careful in interpreting the results given by the comparison tests with Pellet. We see in table 3 that the Pellet reasoner is slower. However, we have only dealt with a sub-part of the constraints that Pellet can manage. Our goal is not to do faster than Pellet but to build a bridge between knowledge engineering and software engineering. This result shows the feasibility of our approach using UML modeling and OCL constraint verification to infer new knowledge. In our application, each inferred knowledge is explained, and the user can select which ones he wants to be integrated into the ontology according to its needs.

[9]https://www.w3.org/2005/Incubator/ssn/ssnx/ssn
[10]https://www.w3.org/TR/owl-time/
[11]https://ontology.winecloud.checksem.fr
[12]https://hal.archives-ouvertes.fr/hal-02329636
[13]https://hal.archives-ouvertes.fr/hal-01342886
[14]https://github.com/stardog-union/pellet

6 CONCLUSIONS AND FUTURE WORK

Industry 4.0 is a new challenge for IT departments. Used to the development of data storage, exchange, and visualization solutions, they must now manage the meaning of data with regard to its context of creation and operation. Unfortunately, knowledge engineering methods and tools are still poorly understood and little practiced in these environments. To facilitate the adoption of these approaches by IT personnel, we have developed an approach based on UML and OCL to model knowledge and allow reasoning.

This paper presents the first results of our approach using UML modeling and OCL constraints verification to infer new knowledge from an ontology. Our approach consists of transforming ontology in a UML model with OCL constraints according to the semantics of ontology axioms. We have shown that OCL constraint verification results can be used to automatically infer new knowledge from an ontology. The objective of this approach is to facilitate the use of ontologies in the industrial field. Obtained results clearly show the feasibility of our approach, and that UML and OCL modeling has good promises in mod-

Table 3: Experimentation results.

Ontology	**Number of triples**	**Number of classes**	**Number of inferred triples**	**ReCoRe reasoning time (s)**	**Pellet reasoning time (s)**
SSN	156	51	60	1,160	3,576
Time	146	20	155	0,815	3,079
WineCloud	244	42	231	1,507	4,241
Illumination	289	49	778	2,302	3,726
WiseNet	177	27	212	0,943	3,280

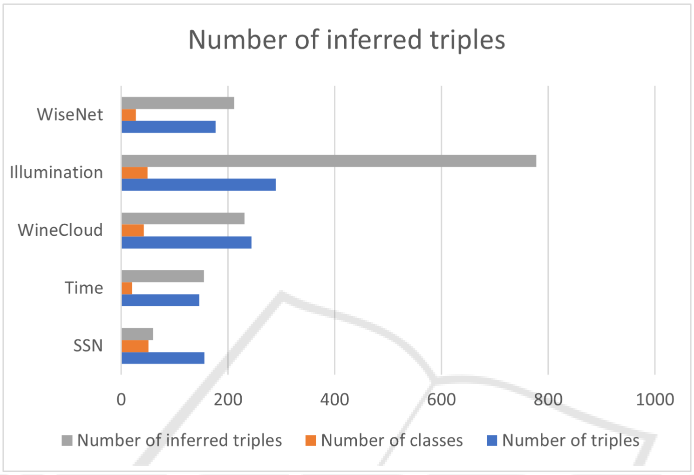


Figure 3: Number of inferred triples.

eling and reasoning on ontologies.

This study is part of ongoing work. Presented results mainly concern transformation rules related to atomic classes and object properties. Less trivial transformations of other parts of the ontology are in progress and under evaluation. Future works will finalize transformation rules of all ontology axioms, do a comparative study with several reasoners such as HermiT[15] and FACT++[16]. A comparative study of our approach with other validation models such as ShEx (Shape Expression Schema) (Prud'hommeaux et al., 2014; Staworko et al., 2015) and SHACL (Shapes Constraint Language)[17] will be conducted, and a template or a common structure will be proposed to help users in writing OCL constraints to control and specialize their ontology. Other perspectives are related to the use of UML modeling to implement the ontology as an object-oriented program and manage its dynamic part.

[15]http://www.hermit-reasoner.com/

[16]http://owl.cs.manchester.ac.uk/tools/fact/

[17]https://www.w3.org/TR/shacl/

ACKNOWLEDGEMENTS

The author would like to thank Nicolas Gros, the engineer who implemented transformation rules and the ReCoRe application interface.

REFERENCES

Abburu, S. (2012). A survey on ontology reasoners and comparison. *International Journal of Computer Applications*, 57(17).

Baclawski, K., Kokar, M. K., Kogut, P. A., Hart, L., Smith, J., Holmes, W. S., Letkowski, J., and Aronson, M. L. (2001). Extending uml to support ontology engineering for the semantic web. In *International Conference on the Unified Modeling Language*, pages 342–360. Springer.

Corcho, O., Gómez-Pérez, A., and Fernández-López, M. (2004). Ontological engineering. *With examples from the areas of Knowledge Management, e-Commerce and the Semantic Web (Advanced Information and Knowledge Processing)*.

Cranefield, S., Haustein, S., and Purvis, M. (2001). Uml-based ontology modelling for software agents.

Cranefield, S. and Purvis, M. K. (1999). Uml as an ontology modelling language. In *Intelligent Information Integration*.

Engels, G. and Groenewegen, L. (2000). Object-oriented modeling: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 103–116.

Fernández-López, M., Gómez-Pérez, A., and Juristo, N. (1997). Methontology: from ontological art towards ontological engineering.

Fu, C., Yang, D., Zhang, X., and Hu, H. (2017). An approach to translating ocl invariants into owl 2 dl axioms for checking inconsistency. *Automated Software Engineering*, 24(2):295–339.

Gogolla, M., Büttner, F., and Richters, M. (2007). Use: A uml-based specification environment for validating uml and ocl. *Science of Computer Programming*, 69(1-3):27–34.

González, C. A. and Cabot, J. (2014). Formal verification of static software models in mde: A systematic review. *Information and Software Technology*, 56(8):821–838.

Hafeez, A., Musavi, S. H. A., and ur Rehman, A. (2018). Ontology-based verification of uml class/ocl model. *Mehran University Research Journal of Engineering & Technology*, 37(4):521–534.

Horrocks, I. (1998). The fact system. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 307–312. Springer.

Khan, A. H. and Porres, I. (2015). Consistency of uml class, object and statechart diagrams using ontology reasoners. *Journal of Visual Languages & Computing*, 26:42–65.

Lupp, D. P., Hodkiewicz, M., and Skjæveland, M. G. (2020). Template libraries for industrial asset maintenance: A methodology for scalable and maintainable ontologies. In *CEUR Workshop Proceedings*, volume 2757, pages 49–64. Rheinisch-Westfaelische Technische Hochschule Aachen* Lehrstuhl Informatik V.

Maedche, A., Schnurr, H.-P., Staab, S., and Studer, R. (2000). Representation language-neutral modeling of ontologies. In *Proceedings of the German Workshop" Modellierung-2000". Koblenz, Germany*, pages 129–142.

Milanović, M., Gašević, D., Giurca, A., Wagner, G., and Devedžić, V. (2006). On interchanging between owl/swrl and uml/ocl. In *Proceedings of 6th Workshop on OCL for (Meta-) Models in Multiple Application Domains (OCLApps) at the 9th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS), Genoa, Italy*, pages 81–95.

Mizoguchi, R. (2004). Tutorial on ontological engineering part 2: Ontology development, tools and languages. *New Generation Computing*, 22(1):61–96.

Mkhinini, M. M., Labbani-Narsis, O., and Nicolle, C. (2020). Combining uml and ontology: An exploratory survey. *Computer Science Review*, 35:100–223.

Parreiras, F. S. and Staab, S. (2010). Using ontologies with uml class-based modeling: The twouse approach. *Data & Knowledge Engineering*, 69(11):1194–1207.

Parsia, B. and Sirin, E. (2004). Pellet: An owl dl reasoner. In *Third international semantic web conference-poster*, volume 18, page 13. Citeseer.

Prud'hommeaux, E., Labra Gayo, J. E., and Solbrig, H. (2014). Shape expressions: an rdf validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems*, pages 32–40.

Richters, M. and Gogolla, M. (2002). Ocl: Syntax, semantics, and tools. In *Object Modeling with the OCL*, pages 42–68. Springer.

Shearer, R., Motik, B., and Horrocks, I. (2008). Hermit: A highly-efficient owl reasoner. In *Owled*, volume 432, page 91.

Stadnicki, A., Pietroń, F. F., and Burek, P. (2020). Towards a modern ontology development environment. *Procedia Computer Science*, 176:753–762.

Staworko, S., Boneva, I., Gayo, J. E. L., Hym, S., Prud'Hommeaux, E. G., and Solbrig, H. (2015). Complexity and expressiveness of shex for rdf. In *18th International Conference on Database Theory (ICDT 2015)*.

Timm, J. T. and Gannod, G. C. (2007). Specifying semantic web service compositions using uml and ocl. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 521–528. IEEE.

Tudorache, T. (2020). Ontology engineering: Current state, challenges, and future directions. *Semantic Web*, 11(1):125–138.

Wang, X. and Chan, C. W. (2001). Ontology modeling using uml. In *OOIS 2001*, pages 59–68. Springer.