Translating OWL 2 ontologies and SHACL shapes into UML/OCL models

Mauro Dalle Lucca Tosi[1], Fitash Ul Haq[1] and Jordi Cabot[1,2]

[1]Luxembourg Institute of Science and Technology (LIST), Esch-sur-Alzette, Luxembourg
[2]University of Luxembourg, Esch-sur-Alzette, Luxembourg

Abstract

Ontologies are increasingly used to represent domain knowledge, while UML remains a standard in Model-Driven Engineering (MDE) for conceptual modeling in software and information-system design. However, when information systems are developed from existing ontologies, these artifacts and their associated validation constraints are rarely reused directly as engineering assets. Existing transformation approaches mostly focus on structural mappings and do not preserve the constraint semantics expressed in W3C standards such as OWL 2 and SHACL in a form directly usable in MDE workflows. In this paper, we propose a framework that transforms OWL 2 ontologies and SHACL shapes into UML class diagrams enriched with OCL constraints. The framework defines systematic transformation rules for both structural and constraint-related constructs, enabling the derivation of UML/OCL models from ontology artifacts together with their relevant validation and logical constraints. As a proof of concept, we applied the approach to the YAGO schema and generated UML/OCL models automatically. The results show the feasibility of reusing ontology artifacts and their constraints as inputs to model-based system design.

Keywords

Ontologies, UML, OWL 2, SHACL, OCL

1. Introduction

Conceptual modeling is central to the design, representation, and exploitation of information systems. It is applied at two distinct but complementary levels: (1) Semantic Web level and (2) Model-Driven Engineering (MDE) level. The Semantic Web level is centered on ontologies, knowledge graphs (KG), RDF/OWL, reasoning, and knowledge integration; while the Model-Driven Engineering (MDE) level is centered on UML, code generation, and design automation. Although both levels model domains and structure knowledge, there are still only partial links to bridge the gap between the two.

Ontologies are particularly effective for representing domain knowledge in a semantically rich manner that enables interoperability, integration, traceability, and inference. Currently, OWL 2 has become the de facto standard language for ontology representation, while SHACL is used to define and validate constraints on such ontologies. OWL 2 and SHACL are defined by the World Wide Web Consortium (W3C). In contrast, MDE artifacts such as UML are well suited for structuring software-relevant views of a domain, supporting communication, model validation, and downstream automation such as code generation. In practice, both perspectives are needed as organizations often maintain rich domain knowledge through ontologies that serve as the backbone of their KGs, while still relying on MDE artifacts for the design, implementation, or evolution of their information systems.

However, when an information system is built on top of an existing ontology-driven KG, the ontology is often not fully exploited as an engineering asset. Instead, system models are frequently recreated from scratch, leading to duplicated modeling effort, an increased risk of inconsistencies, and a weakened alignment between the knowledge layer and the system-design layer.

Transforming ontologies into UML class diagrams is not a new idea, and existing research has explored mappings between ontology constructs and UML structures [1]. However, the majority of these

Knowledge Graphs and Model-driven Systems Engineering (KGMDSE), in ISWC 2026 Workshops Joint Proceedings, October 25–26, 2026, Bari, Italy

mauro.dalle-lucca-tosi@list.lu (M. Dalle Lucca Tosi); fitash.ulhaq@list.lu (F. Ul Haq); jordi.cabot@list.lu (J. Cabot)
0000-0002-0218-2413 (M. Dalle Lucca Tosi); 0000-0003-2253-9085 (F. Ul Haq); 0000-0003-2418-2489 (J. Cabot)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-05

CEUR-WS.org/Vol-4313/kgmdse-paper1.pdf

approaches focus on structural correspondences, such as classes, properties, and hierarchies [2, 3]. Only a few studies have considered the transformations from Semantic Web artifacts rules and constraints into OCL; among them, they either address it partially [4] or based on a specific ontology [5]. To the best of our knowledge, no prior work has proposed a comprehensive mapping between OWL 2 and SHACL shapes to UML and OCL constraints that preserves constraint semantics in a form directly usable within MDE workflows. As a result, with available approaches, the generated UML models may reflect the structure of an ontology, but fail to incorporate its validation-relevant or rule-oriented semantics.

To address this gap, we propose a framework that transforms OWL 2 ontologies and SHACL shapes into UML models containing OCL constraints. Beyond structural mapping, the framework translates OWL 2 axioms that cannot be captured by plain UML and SHACL shapes into OCL constraints for the generated UML model. This enables a semantically faithful transition from ontology-centered conceptualization to MDE-oriented design artifacts. The resulting UML/OCL models can serve as a bridge between enterprise knowledge assets, system design and implementation workflows, thereby supporting activities such as model validation, code generation, and system development aligned with ontological semantics.

This contribution is particularly relevant in a context where model-based development is gaining renewed importance. Low-code platforms already consume class diagrams and their constraints [6], so an application can be derived from the generated model without an LLM in the loop. And where generative AI is involved, as in requirements-driven methodologies supported by AI agents [7], the generated model is closer to the information system to be developed than the ontology it comes from: it already states what the system has to implement and enforce, leaving the agent a shorter and better-constrained step to code. More broadly, this work contributes to the convergence of ontology and model-driven engineering. In summary, our main contributions are: (i) a transformation framework from ontologies to UML models; (ii) a mapping of OWL 2 and SHACL constraints to OCL; and (iii) a step toward integrating W3C assets into model-based development pipelines.

2. State of the art

Model transformations between UML and OWL 2 have been widely studied, but most works focus on the UML-to-OWL direction. In contrast, OWL-to-UML remains comparatively underexplored [8]. In their review of transformation approaches, Suchánek and Pergl [8] identify the work of Zedlitz and Luttenberger [9] as explicitly addressing OWL-to-UML transformation. Suchánek and Pergl further observe that real-world OWL-to-UML transformation cases do not seem to have been published. Broader reviews on the combination of UML and ontologies confirm this imbalance [1].

It is important to note that not all ontology-to-UML works address the same ontology language. For instance, Hnatkowska [5] specifically targets the SUMO ontology, while Waldemarin and de Farias [10] converts OBO, an ontology serialization format less expressive than OWL, into UML.

This distinction is important because OWL 2, introduced at the end of 2009, extends OWL with constructs that significantly increase the expressiveness of conceptual models, such as qualified cardinality restrictions, property chains, keys, and other advanced axioms. These features are particularly relevant for faithful conceptual modeling, but they also make transformation to UML more difficult. As a result, works published before 2010, such as [11, 12, 3, 4], necessarily target OWL rather than OWL 2, and their transformation results do not cover the full expressiveness of contemporary OWL 2 ontologies.

Accordingly, only a few works address transformation from OWL 2 to UML [13, 14, 9, 2]. Among them, Liepinš et al. [13] transforms OWL 2 into UML mainly for visualization and conceptual understanding. In contrast, Zedlitz and Luttenberger [14, 9, 2] propose a bidirectional mapping, from OWL 2 to UML and vice versa. Their OWL2-to-UML mapping is closer to our objective of transforming OWL 2 ontologies into UML artifacts that support the development of information systems. However, because they rely exclusively on UML class diagrams, they are not able to represent OWL 2 axioms that would require OCL in UML, such as *ReflexiveObjectProperty*, *TransitiveObjectProperty*, and *NegativeDataPropertyAssertion*,

among others.

Some studies are available in the literature that translates ontologies into UML diagrams enriched with OCL constraints [5, 4]. Hnatkowska [5] uses OCL constraints to translate the SUMO ontology into UML, while Milanović et al. [4] provide a metamodel mapping framework between OWL/SWRL and UML/OCL. However, only a subset of the transformation rules from OWL is presented explicitly. Overall, the literature shows that OWL2-to-UML transformation remains underexplored when the target is a UML model enriched with OCL in order to fully capture the semantics of OWL 2. Table 1 summarizes the comparison among the methods discussed so far. Its last column reports how much of the non-structural semantics of the source language each approach preserves, that is, to what extent constructs without a counterpart in a UML class diagram are still translated: *limited* when they are dropped, *partial* when only a subset of them is translated, and *broad* when they are translated systematically.

Method	Source language	Target model	Semantic Coverage
[10]	OBO	UML	Limited
[5]	SUMO	UML/OCL	Partial
[11, 12, 3]	OWL	UML	Limited
[4]	OWL	UML/OCL	Partial
[14, 9, 13, 2]	OWL 2	UML	Limited
This work	**OWL 2 / SHACL**	**UML/OCL**	**Broad**

Table 1
Comparison of ontology-to-UML transformation approaches.

In addition to OWL 2, the W3C provides several standards to support the Semantic Web community for ontology-related tasks. In particular, the Shapes Constraint Language (SHACL) was developed to validate graphs against a set of predefined conditions. While OWL 2 conceptually models domain knowledge, including its restrictions, SHACL is used to validate predefined conditions, raising errors when the ontology violates them.

Since both SHACL and OCL are languages designed to validate constraints in their respective domains, it is natural to explore this transformation in the context of OWL 2 to UML/OCL. OCL-to-SHACL transformations have already been investigated [15, 16]. Larhrib et al. [15] provide the overall correspondence and intuition behind the OCL-to-SHACL transformation, but they do not present a direct mapping table or handle non-trivial cases involving nested expressions, collections, and similar constructs. By contrast, the technical report [16] by the Open Geospatial Consortium provides a direct and complete mapping table from OCL to SHACL. Although this table offers insights into the opposite transformation, from SHACL to OCL, several SHACL shapes are not described in it. For instance, shapes that are simple to transform, such as *sh:minCount*, as well as shapes that do not have a direct transformation, such as *sh:nodeKind*, are not discussed in the report. This illustrates the current gap in the literature on SHACL-to-OCL transformation.

In summary, the literature reveals two main gaps. First, OWL2-to-UML transformation remains underexplored when the target is a UML model enriched with formal constraints. Second, SHACL-to-OCL transformation has also not been explored yet. In this paper, we address both gaps by proposing a framework that transforms OWL 2/SHACL into UML/OCL.

3. From OWL 2 and SHACL to UML and OCL

In this section, we present a framework that transforms semantic web artifacts into UML models for information-system development. Given an OWL 2 DL ontology under the Direct Semantics and SHACL shapes, it produces a UML class diagram enriched with OCL constraints. Unlike approaches limited to ontology visualization or structural translation, the proposed framework addresses both schema structure and integrity constraints.

The framework has two phases. The first translates OWL 2 constructs into UML elements and, when needed, OCL constraints. The second translates SHACL shapes into OCL constraints attached to the

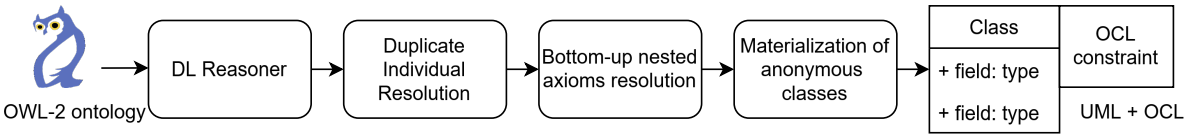


Figure 1: OWL 2 to UML/OCL Transformation Overview

generated UML model. These constraints are expressed as OCL invariants. Therefore, they hold not only for the content produced by the transformation, but also for any instance created later.

3.1. From OWL 2 to UML

The first phase transforms an OWL 2 ontology into a UML model enriched, when necessary, with OCL constraints. Beyond schema-level elements, the proposed framework also translates instance-level content: OWL 2 classes and properties become UML classes, associations, and attributes, while individuals and assertions become UML objects, links, and values.

Although ontology-to-UML transformations have been studied before [11, 17], most existing work targets ontology engineering or visualization. In contrast, the proposed framework produces UML artifacts intended to support information-system modeling and implementation.

At a high level, OWL 2 and UML class diagrams share several concepts: classes map to UML classes, object properties to associations, data properties to attributes, subclass relations to generalizations, and individuals to UML objects. However, the translation is not fully direct because OWL 2 follows the Open World Assumption (OWA), whereas UML and OCL are generally interpreted under a Closed World Assumption (CWA). This mismatch affects the treatment of incomplete information, equivalent individuals, and anonymous class expressions.

To mitigate these issues, the OWL 2 to UML/OCL transformation follows the process shown in Figure 1. First, the OWL 2 ontology is enriched with inferable knowledge obtained from an OWL 2 DL reasoner; the step depends on the entailed axioms rather than on a particular implementation, so any sound and complete reasoner can be used. Second, equivalent individuals are merged into a canonical entity; in inconsistent cases, the framework retains the property values of the most connected version. Third, nested OWL 2 axioms are processed bottom-up so that inner expressions are translated before enclosing ones. Finally, anonymous or implicit classes required for the mapping are materialized explicitly. For *ObjectPropertyDomain* and *ObjectPropertyRange* with class unions, associations are linked directly to each member class; if both a superclass and one of its subclasses appear in the same union, only the superclass is retained.

Because the OWA-to-CWA shift may still create inconsistencies, we provide three strategies: materialize missing objects, discard the problematic constraints, or resolve inconsistencies manually before conversion. The goal is therefore not perfect semantic equivalence, but a principled and practical transformation that preserves the information most relevant for information-system design.

Based on these choices, we define a systematic mapping from OWL 2 axioms to UML and OCL constructs. Some axioms map directly to UML; others require complementary OCL constraints; others are omitted when they add complexity without affecting the produced UML structure. Table 2 lists the OWL 2 axioms translated directly into UML, and Table 3 those requiring OCL.

Table 2: OWL 2 → UML direct-transformation rules.

#	OWL 2	UML
D01	Ontology	Package
D02	Ontology IRI / Version IRI	Package name / version tag
D03	Import	PackageImport / PackageMerge
D04	Ontology annotations	Comment on the package

continued on next page

Table 2 (continued)

#	OWL 2	UML
D05	Datatypes within XML-Schema	UML primitive
D06	IRI	URI datatype OR create URI datatype with URI as string attribute
D07	XML Literals	string
D08	Class	Class
D09	Datatype	DataType
D10	ObjectProperty	Association
D11	DataProperty	attribute of the Domain class
D12	AnnotationProperty	Comment / tagged value
D13	NamedIndividual	Instance
D14	AnonymousIndividual	Materialize individual
D15	Literal	Value
D16	ObjectInverseOf	opposite association end
D17	DataOneOf (literals)	Enumeration
D18	ObjectIntersectionOf($C_1 \dots C_n$)	Auxiliary class _C_1_…_C_n_Intersection with one Generalization _Aux $\triangleright C_i$ per operand
D19	ObjectUnionOf($C_1 \dots C_n$)	Auxiliary abstract superclass _C_1_…_C_n_Union with one Generalization $C_i \triangleright$ _Aux per operand.
D20	ObjectOneOf($a_1 \dots a_n$) — symbolic individuals	Enumeration
D21	ObjectOneOf($a_1 \dots a_n$) — relational individuals	Auxiliary utility class _a_1_…_a_n_OneOf with one association/attribute per individual
D22	ObjectSomeValuesFrom($p\ D$)	Auxiliary class _some_p_D. Association [_some_p_D] — $p \rightarrow$ [D] [1..*]
D23	ObjectMinCardinality($n\ p\ D$)	Auxiliary class _min_n_p_D. Association [_min_n_p_D] — $p \rightarrow$ [D] [n..*]
D24	ObjectMaxCardinality($n\ p\ D$)	Auxiliary class _max_n_p_D. Association [_max_n_p_D] — $p \rightarrow$ [D] [0..n]
D25	ObjectExactCardinality($n\ p\ D$)	Auxiliary class _exact_n_p_D. Association [_exact_n_p_D] — $p \rightarrow$ [D] [n..n]
D26	SubClassOf($CE_1\ CE_2$)	Class CE_1 and Class CE_2. $CE_1 \triangleright CE_2$
D27	EquivalentClasses($CE_1 \dots CE_n$)	Normalize to one canonical class.
D28	EquivalentObjectProperties($OPE_1 \dots OPE_n$)	Normalize to one canonical association
D29	InverseObjectProperties($OPE_1\ OPE_2$)	One bidirectional association with opposite ends named OPE_1 and OPE_2
D30	ObjectPropertyDomain($OPE\ CE$)	Source end of association/property OPE typed by the UML class CE
D31	ObjectPropertyRange($OPE\ CE$)	Target end of association/property OPE typed by the UML class CE
D32	FunctionalObjectProperty(OPE)	Association end multiplicity at target set to [0..1]
D33	EquivalentDataProperties($DP_1 \dots DP_n$)	Normalize to one canonical data property
D34	DataPropertyDomain($DP\ C$)	Attribute/property DP owned by UML class C
D35	DataPropertyRange($DP\ DR$)	Attribute/property DP of type DR
D36	FunctionalDataProperty(DP)	Single-valued UML attribute/property DP. If is a complex type, create association to DP with multiplicity [0..1]
D37	DatatypeDefinition($DT\ DR$)	UML DataType DT materializing the UML translation of DR
D38	HasKey(CE ($OPE_1 \dots OPE_m$) ($DPE_1 \dots DPE_n$))	UML key/unique identifier constraint on class CE using the participating properties
D39	ClassAssertion($CE\ a$)	UML InstanceSpecification a typed by the class CE
D40	ObjectPropertyAssertion($OPE\ a_1\ a_2$)	UML object link $a_1 - OPE \rightarrow a_2$
D41	DataPropertyAssertion($DPE\ a\ lt$)	Slot/value assignment $a.DPE = lt$

Table 3: OWL 2 → UML/OCL transformation rules. †: the auxiliary class carries the OCL invariant, which every named class subsumed by the expression inherits by becoming its subclass (step left implicit).

#	OWL 2	UML	OCL
O01	DataIntersectionOf($DT_1 \dots DT_n$)	Constrained UML DataType DT	context DT inv: satisfies_DT_1(self.value) and ... and satisfies_DT_n(self.value)
O02	DataUnionOf($DT_1 \dots DT_n$)	Constrained UML DataType DT	context DT inv: satisfies_DT_1(self.value) or ... or satisfies_DT_n(self.value)
O03	DataComplementOf(DT)	Constrained UML DataType DT	context DT inv: not satisfies_DT(self.value)
O04	DataOneOf (typed)	class with OCL invariant constraint	separate class containing one attribute, which value and type are checked with OCL invariant.
O05	DatatypeRestriction	constrained PrimitiveType	OCL facet invariant (≥ min, ≤ max, matches pattern, length, ...)
O06	ObjectComplementOf(C)	Abstract auxiliary class _not_C†	context _not_C inv: not self.oclIsKindOf(C)
O07	ObjectAllValuesFrom(p D)	Auxiliary class _all_p_D; association [_all_p_D] − p → [D] [0..*]	context _all_p_D inv: self.p → forAll(o | o.oclIsKindOf(D))
O08	ObjectHasSelf(p)	Auxiliary class _hasSelf_p; self-loop [_hasSelf_p] − p → [_hasSelf_p]	context _hasSelf_p inv: self.p → includes(self)
O09	ObjectHasValue(p a)	Auxiliary class _hasValue_p_a; association [_hasValue_p_a] − p → [T_a] [1..*], where T_a is the UML type of individual a	context _hasValue_p_a inv: self.p→includes(a)
O10	DataSomeValuesFrom(dp dr)	Auxiliary class _some_dp_dr†	context _some_dp_dr inv: self.dp → asSet() → exists(v | satisfies_dr(v))
O11	DataAllValuesFrom(dp dr)	Auxiliary class _all_dp_dr†	context _all_dp_dr inv: self.dp → asSet() → forAll(v | satisfies_dr(v))
O12	DataHasValue(dp v)	Auxiliary class _hasValue_dp_v†	context _hasValue_dp_v inv: self.dp → asSet() → includes(v)
O13	DataMinCardinality(n dp dr)	Auxiliary class _min_n_dp_dr†	context _min_n_dp_dr inv: self.dp → asSet() → select(v | satisfies_dr(v)) → size() ≥ n
O14	DataMaxCardinality(n dp dr)	Auxiliary class _max_n_dp_dr†	context _max_n_dp_dr inv: self.dp → asSet() → select(v | satisfies_dr(v)) → size() ≤ n
O15	DataExactCardinality(n dp dr)	Auxiliary class _exact_n_dp_dr†	context _exact_n_dp_dr inv: self.dp → asSet() → select(v | satisfies_dr(v)) → size() = n
O16	DisjointClasses($CE_1 \dots CE_n$)	Disjoint GeneralizationSet if the CE_i share a common superclass; otherwise pairwise OCL (last column)	context CE_i inv: not self.oclIsKindOf(CE_j)
O17	DisjointUnion(C $CE_1 \dots CE_n$)	Superclass C with GeneralizationSet {disjoint, complete} over all CE_i (named via bottom-up materialization)	context C inv complete: self.oclIsKindOf(CE_1) or ... or self.oclIsKindOf(CE_n); context C inv disjoint: Sequence{$CE_1, \dots, CE_n$} → select(cei | self.oclIsKindOf(cei)) → size() ≤ 1
O18	SubObjectPropertyOf(OPE_1 OPE_2)	OPE_1 and OPE_2 as independent named associations	context C inv: self.OPE_1 → forAll(o | self.OPE_2 → includes(o))
O19	SubObjectPropertyOf(ObjectPropertyChain($OPE_1 \dots OPE_n$) OPE) — OPE exclusively derived	All OPE_i as associations; OPE as {derived} association	context C def: /OPE : Set(T) = self.OPE_1.OPE_2.....OPE_n
O20	SubObjectPropertyOf(ObjectPropertyChain($OPE_1 \dots OPE_n$) OPE) — OPE has independent assertions	All OPE_i and OPE as independent named associations	context C inv: self.OPE_1.OPE_2.....OPE_n → forAll(o | self.OPE → includes(o))
O21	DisjointObjectProperties($OPE_1 \dots OPE_n$)	All OPE_i as independent named associations	context C inv: self.OPE_i → forAll(o | self.OPE_j → excludes(o)) for every pair (i,j) where $i \neq j$

continued on next page

Table 3 (continued)

#	OWL 2	UML	OCL
O22	InverseFunctionalObjectProperty(*OPE*)	Source-end multiplicity [0..1]: [*C*] [0..1] − *OPE* → [*D*]	context *D* inv: *D*.allInstances() → forAll(*d* | *C*.allInstances() → select(*c* | *c*.*OPE* → includes(*d*)) → size() ≤ 1)
O23	ReflexiveObjectProperty(*OPE*)	*OPE* as a self-loop association on its domain class *C*: [*C*] − *OPE* → [*C*]	context *C* inv: self.*OPE* → includes(self)
O24	IrreflexiveObjectProperty(*OPE*)	*OPE* as named association	context *C* inv: self.*OPE* → excludes(self)
O25	AsymmetricObjectProperty(*OPE*)	Association *OPE*	context *C* inv: self.*OPE* → forAll(x | not x.*OPE* → includes(self))
O26	TransitiveObjectProperty(*OPE*)	Association *OPE*	context *C* inv: self.*OPE* → forAll(*x* | *x*.*OPE* → forAll(*y* | self.*OPE* → includes(*y*)))
O27	SubDataPropertyOf(DP_1 DP_2)	Data properties DP_1 and DP_2; if several share DP_2, derive DP_2 as their union	context *C* inv: self.DP_2 = self.DP_1 → union(self.DP_3 ...) (when multiple subproperties exist)
O28	DisjointDataProperties(DP_1 ... DP_n)	All DP_i as data properties	context *C* inv: self.DP_1 → intersection(self.DP_2) → isEmpty()
O29	NegativeObjectPropertyAssertion(*OPE* a_1 a_2)	May be ignored (under CWA a missing link is already false) or enforced via OCL (last column)	context a_1 inv: not self.*OPE* → includes(a_2)
O30	NegativeDataPropertyAssertion(*DPE* *a* *lt*)	Forbid literal *lt* on property *DPE* of instance *a* (via OCL, last column)	context *a* inv: not self.*DPE* → asSet() → includes(*lt*)

Other than the axioms listed in Tables 2 and 3, we do not translate annotation-related axioms, nor the following constructs: *Prefix*, *Declaration*, *DifferentIndividuals*, and *SameIndividual*. Annotation-related axioms are omitted because, in UML, they serve only documentation purposes. *DifferentIndividuals* and *SameIndividual* are handled during pre-processing and canonicalization, since UML objects are expected to be uniquely identified.

3.2. SHACL shapes to OCL

Once the OWL 2 ontology has been transformed, the structure of the target UML model is already defined. In particular, the OWL 2-to-UML phase produces the classes, attributes, associations, and, when relevant, instance-level elements that constitute the structural backbone of the target model. The role of the SHACL transformation phase is therefore not to modify this structure, but to complement it with validation constraints expressed in OCL.

This design reflects the different purposes of OWL 2 and SHACL in the proposed framework. OWL 2 is used primarily to derive the conceptual structure of the domain, whereas SHACL is used to express data-quality and validation rules over that structure. Consequently, the SHACL-to-OCL transformation relies on the UML elements already generated from OWL 2: SHACL node shapes are interpreted with respect to the corresponding UML classes, and SHACL property shapes are translated into OCL constraints over the UML attributes and associations created in the previous phase.

Compared with OWL 2, SHACL is semantically closer to OCL because both are intended for validation. For this reason, many SHACL constraints can be translated directly or systematically into OCL invariants. Table 4 presents the SHACL constraints currently supported by the transformation and their corresponding OCL expressions. Typical examples include cardinality constraints such as *sh:minCount* and *sh:maxCount*, datatype constraints such as *sh:datatype*, value restrictions such as *sh:hasValue*, and logical combinations such as *sh:and*, *sh:or*, and *sh:not*. Table 4 summarizes the supported mappings.

Table 4: SHACL Constraints and their OCL Equivalents.

#	SHACL Constraint	OCL Equivalent
S01	sh:datatype	self.property.oclIsTypeOf(*"datatype"*)
S02	sh:xone	Set{c_1, c_2, … , c_n} → select(x | x = true) →size() = 1
S03	sh:hasValue	self.property → includes(value)
S04	sh:pattern	matches(regex)
S05	sh:uniqueLang	self.property → collect(p | p.language)→ isUnique() (if ∃p.language)
S06	sh:equals	self.p →forAll(v | self.q→ includes(v)) and self.q →forAll(v | self.p → includes(v))
S07	sh:qualifiedValueShape	filtered collection count. self.member→select(m | m.oclIsKindOf(Person))→size() ≥ 2
S08	sh:nodeKind	No equivalent
S09	sh:closed	No equivalent
S10	sh:flags	Tool-specific
S11	sh:languageIn	No equivalent
S12	sh:severity	No equivalent
S13	sh:message	No equivalent
S14	sh:class	oclIsKindOf(Class)
S15	sh:and	and
S16	sh:or	or
S17	sh:not	not
S18	sh:in	Set→includes(x)
S19	sh:minCount	property→size() ≥ n
S20	sh:maxCount	property→size() ≤ n
S21	sh:minLength	string.size() ≥ n
S22	sh:maxLength	string.size() ≤ n
S23	sh:minInclusive	≥
S24	sh:maxInclusive	≤
S25	sh:minExclusive	>
S26	sh:maxExclusive	<
S27	sh:disjoint	intersection()→isEmpty()
S28	sh:lessThan	<
S29	sh:lessThanOrEquals	≤
S30	sh:path	self.property
S31	sh:node	self.property→forAll(v | v.oclIsKindOf(ShapeType))

However, the translation is not always one-to-one. Some SHACL constraints depend on validation features that have no direct counterpart in standard OCL, while others require assumptions about how the target information system will represent data. In particular, constraints involving language tags require a configuration choice: depending on whether the target system is intended to be multilingual, the generated UML model and the corresponding OCL constraints may need to be extended differently. Likewise, SHACL constructs intended only for human-readable validation feedback, such as *sh:message* or *sh:severity*, are not translated into OCL because they do not define model constraints.

This phase therefore builds on the UML model produced earlier and adds a further layer of OCL constraints derived from SHACL shapes. Unlike the previous phase, which may generate both UML elements and OCL expressions from OWL 2 axioms, the present phase assumes that the structural elements already exist and uses them as the basis for expressing SHACL constraints in OCL.

4. Case study and proof of concept

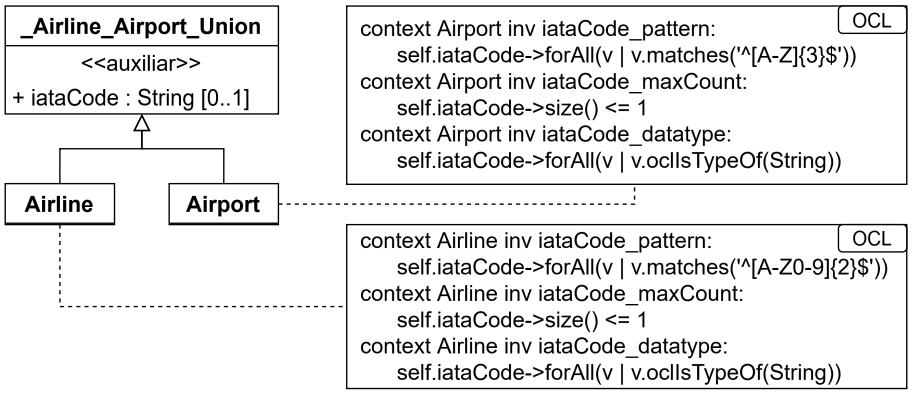
To assess the feasibility of the transformation, we use the YAGO knowledge base [18] as a case study. More precisely, we rely on the *YAGO-schema* OWL 2 DL ontology and its corresponding SHACL shapes

rather than on the full YAGO dataset, which is too large for this proof of concept. The schema is nevertheless suitable for evaluation because it contains a diverse set of constructs and constraints. The transformation code and the resulting artifacts are available online[1].

We converted the input `.nt` files into `.ttl` in the preprocessing step. Consirering that the YAGO schema is already explicit enough for the mapping, no reasoner enrichment was required and the reported results follow the transformation rules directly. We then analyzed the source artifacts and identified 407 OWL 2 and 266 SHACL constructs, which were transformed into a UML class model enriched with OCL constraints. Figure 2 shows an excerpt of the result.

```
<http://schema.org/iataCode> a owl:DatatypeProperty, owl:FunctionalProperty ;
    rdfs:domain [ a owl:Class ;
            owl:unionOf ( <http://schema.org/Airline> <http://schema.org/Airport> ) ] ;
    rdfs:range xsd:string .
<http://yago-knowledge.org/value/shape-prop-schema-Airline-schema-iataCode>
    a sh:PropertyShape ;
    sh:datatype xsd:string ;
    sh:maxCount 1 ;
    sh:path <http://schema.org/iataCode> ;
    sh:pattern "^[A-Z0-9]{2}$" .
<http://yago-knowledge.org/value/shape-prop-schema-Airport-schema-iataCode>
    a sh:PropertyShape ;
    sh:datatype xsd:string ;
    sh:maxCount 1 ;
    sh:path <http://schema.org/iataCode> ;
    sh:pattern "^[A-Z]{3}$" .
```

(a) Excerpt of YAGO schema OWL 2 ontology and corresponding SHACL property shapes.



(b) UML/OCL representation generated from the OWL 2/SHACL excerpt shown in Figure 2a.

Figure 2: Illustration of the proposed transformation from OWL 2 and SHACL constructs to UML/OCL.

As shown in Figure 2b, the transformation rules from Section 3 can be applied to a real ontology schema. The example illustrates both the materialization of anonymous nodes in the UML model and the integration of SHACL-derived constraints into OCL.

To evaluate the transformation, we randomly sampled statistically significant subsets of the identified OWL 2 and SHACL constructs, using a 95% confidence level and a 5% margin of error. This yielded samples of 198 OWL 2 constructs and 158 SHACL constructs. One of the authors manually validated every transformation in the sampled subsets. For each case, the author compared the input OWL 2 axiom or SHACL construct with its generated UML/OCL representation and assessed whether the two were syntactically and semantically consistent.

[1] https://github.com/maurodlt/owl2uml

Table 5 summarizes the results of the complete manual validation. Out of the 198 analyzed OWL 2 constructs, 197 were transformed correctly and one was classified as partially correct. Of the 158 analyzed SHACL constructs, 156 were transformed correctly and two were classified as partially correct. No transformation was classified as incorrect.

Dataset	Total Validated	Correct	Partial	Incorrect
OWL2	198	197	1	0
SHACL	158	156	2	0

Table 5
Results of the complete manual validation. Every construct in the sampled OWL 2 and SHACL subsets was manually inspected by one author.

The three partially correct transformations had the same underlying cause. They involved *subPropertyOf* relations referring to the external property `<http://schema.org/identifier>`, which had not been imported before the transformation. The implementation includes a safeguard that creates OCL constraints only when all referenced properties and objects are present in the generated UML diagrams. Consequently, the corresponding constraints were omitted to prevent generation of constraints with broken references. These cases could be resolved by explicitly importing the required external vocabulary before performing the transformation.

Overall, the manual validation confirms that the transformation was successfully applied to the YAGO schema and that the proposed framework works as intended for the analyzed constructs. The results show that OWL 2 axioms and SHACL constraints can be systematically represented as UML class-model elements and OCL constraints, while preserving the relevant structural and validation-related information.

4.1. Discussion

The case study shows that the proposed framework can successfully transform OWL 2 and SHACL artifacts into UML/OCL models suitable for software-oriented conceptual modeling. In this sense, the proof of concept confirms that semantic web assets can be reused beyond knowledge representation and validation, and can also support downstream engineering activities through the generation of UML and OCL artifacts.

At the same time, the transformation should not be understood as a strict one-to-one correspondence between OWL 2/SHACL and UML/OCL. The main reason is the semantic difference between the OWA adopted by OWL 2 and the CWA typically underlying UML/OCL validation. Because of this mismatch, together with the design decisions discussed in Section 3, the transformation is not fully reversible: converting the resulting UML/OCL model back into OWL 2/SHACL would necessarily lead to some information loss.

The manual validation also highlighted limitations and opportunities for improvement. First, the generated models may contain redundancies, particularly when equivalent constraints are specified in both OWL 2 and SHACL. For example, an OWL 2 datatype property may already define its range as `xsd:string`, while a corresponding SHACL property shape independently requires the same property to have datatype `xsd:string`. In the generated UML/OCL model, this may result both in a UML attribute typed as `String` and in an OCL constraint verifying that its values are strings. Although this representation is correct and preserves the information contained in both source artifacts, it introduces redundant elements.

Second, lists represented through anonymous nodes in OWL 2 or SHACL lose their ordering when materialized in the UML model. This limitation did not affect the YAGO case study, but it may be relevant for source artifacts in which list order carries semantic significance.

Third, some correctly generated materializations may introduce a degree of complexity that would not normally be present in a UML model created manually by a domain expert. In particular, auxiliary

classes may be necessary to represent the source artifacts faithfully, even though they reduce the compactness and readability of the resulting diagrams. This reflects a trade-off between semantic completeness and model usability.

The quality of the generated UML model is also directly related to the quality and comprehensiveness of the input ontology and SHACL shapes. The transformation can only be as complete and precise as the source artifacts. When those are rich, well-structured, explicit, and include the external vocabularies on which they depend, the resulting UML/OCL model benefits from that quality.

As future work, we intend to investigate post-transformation simplification techniques that reduce diagram complexity while preserving relevant semantics. This includes the identification and removal of redundant associations and constraints, simplification of auxiliary materializations, and strategies for preserving ordered collections where needed. A further direction concerns the assessment of instance-level data, which is not covered by the present case study. In addition, the framework should be evaluated on further OWL 2 ontologies and SHACL shape collections from different domains and with varying modeling styles, sizes, and levels of complexity. Such evaluations would help assess the generalizability of the transformation rules and identify additional construct combinations or modeling patterns that may require refinement. Finally, future manual validations should involve multiple annotators to strengthen the reliability of the evaluation.

5. Conclusions

In this paper, we presented a framework that defines systematic transformation rules from OWL 2 ontologies and SHACL shapes into UML class diagrams enriched with OCL constraints. Unlike approaches limited to structural translation, the proposed framework also captures validation and logical constraints relevant for information-system development, thus enabling the generation of UML/OCL artifacts that can be used within MDE workflows.

The proof of concept based on the YAGO schema demonstrated the feasibility of the approach and showed that ontology artifacts and their associated constraints can be reused as engineering assets for software-oriented conceptual modeling. At the same time, the transformation is not a one-to-one mapping. Due in particular to the semantic differences between the OWA of OWL 2 and the CWA typically underlying UML/OCL, some information is inevitably lost or approximated in the target model. This is acceptable because the two artifacts are not interchangeable: the ontology remains the reference for knowledge-level semantics, while the UML/OCL model is the engineering view.

This division of roles is also what makes the generated artifacts useful in practice. Model-driven tools that take class diagrams and their constraints as input can derive an application from the generated model directly, preserving the classic MDE promise of deterministic generation with no LLM in the loop. And where generative AI is used instead, it offers a starting point that already describes the system to be built rather than the domain it draws on.

As future work, we intend to investigate post-transformation simplification techniques to reduce redundancies and improve the readability of the generated UML/OCL models while preserving their relevant semantics. We will also extend the evaluation to instance-level transformation and to a broader range of ontologies and SHACL shape collections, supported by manual validation involving multiple annotators. Overall, this work represents a step toward a closer integration of ontology engineering and model-based system development.

Acknowledgments

This project is supported by the Luxembourg National Research Fund (FNR) PEARL program, grant agreement 16544475.

Declaration on Generative AI

During the preparation of this work, the authors used GPT-5.5 in order to: check grammar and spelling, and improve writing style. The authors also used Claude Opus 4.8 to assist in the development of the experiments. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] M. M. Mkhinini, O. Labbani-Narsis, C. Nicolle, Combining uml and ontology: An exploratory survey, Computer Science Review 35 (2020) 100223.

[2] J. Zedlitz, N. Luttenberger, Transforming between uml conceptual models and owl 2 ontologies., in: Terra Cognita@ ISWC, 2012, pp. 15–26.

[3] H. El-Ghalayini, M. Odeh, R. McClatchey, Engineering conceptual data models from domain ontologies: a critical evaluation, International Journal of Information Technology and Web Engineering (IJITWE) 2 (2007) 57–70.

[4] M. Milanović, D. Gašević, A. Giurca, G. Wagner, V. Devedžić, Towards sharing rules between owl/swrl and uml/ocl, Electronic Communications of the EASST 5 (2006).

[5] B. Hnatkowska, Automatic sumo to uml translation, e-Informatica Software Engineering Journal 10 (2016).

[6] I. Alfonso, A. Conrardy, A. Sulejmani, A. Nirumand, F. Ul Haq, M. Gomez-Vazquez, J.-S. Sottet, J. Cabot, Building besser: an open-source low-code platform, in: International Conference on Business Process Modeling, Development and Support, Springer, 2024, pp. 203–212.

[7] J. Cabot, Vibe-driven model-based engineering, arXiv preprint arXiv:2604.10645 (2026).

[8] M. Suchánek, R. Pergl, Case-study-based review of approaches for transforming uml class diagrams to owl and vice versa, in: 2020 IEEE 22nd Conference on Business Informatics (CBI), volume 1, IEEE, 2020, pp. 270–279.

[9] J. Zedlitz, N. Luttenberger, Conceptual modelling in uml and owl-2, International Journal on Advances in Software 7 (2014) 182–196.

[10] R. C. Waldemarin, C. R. de Farias, Obo to uml: Support for the development of conceptual models in the biomedical domain, Journal of Biomedical Informatics 80 (2018) 14–25.

[11] S. Brockmans, R. Volz, A. Eberhart, P. Löffler, Visual modeling of owl dl ontologies using uml, in: International Semantic Web Conference, Springer, 2004, pp. 198–213.

[12] R. Grønmo, M. C. Jaeger, H. Hoff, Transformations between uml and owl-s, in: European Conference on Model Driven Architecture-Foundations and Applications, Springer, 2005, pp. 269–283.

[13] R. Liepinš, M. Grasmanis, U. Bojars, Owlgred ontology visualizer, in: Proceedings of the 2014 international conference on developers, volume 1268, CEUR-WS. org, 2014, pp. 37–42.

[14] J. Zedlitz, N. Luttenberger, Data types in uml and owl-2, SEMAPRO 2013 (2013) 40.

[15] M. Larhrib, M. Escribano, C. Cerrada, J. J. Escribano, Converting ocl and cgmes rules to shacl in smart grids, IEEE Access 8 (2020) 177255–177266.

[16] D. L. Nichols, P. Birkel, OGC Testbed-14: Application Schema-based Ontology Development Engineering Report, Technical Report OGC 18-032r2, OGC Public Engineering Report, City, Country, 2018. URL: http://www.opengis.net/doc/PER/t14-D022.

[17] F. S. Parreiras, T. Walter, G. Gröner, Visualizing ontologies with uml-like notation, in: Ontology-Driven Software Engineering, 2010, pp. 1–6.

[18] F. M. Suchanek, M. Alam, T. Bonald, L. Chen, P.-H. Paris, J. Soria, Yago 4.5: A large and clean knowledge base with a rich taxonomy, in: Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval, 2024, pp. 131–140.