University of Twente

Master's thesis

Metamodel Transformations Between UML and OWL

R.E.Y. Haasjes

Supervisors Dr. L. Ferreira Pires
Dr. A. Fehnker
MSc. J. van Genuchten
MSc. M.H.M. Stornebrink
MSc. P. Stapersma

August 27, 2019

Abstract

Both the *Unified Modeling Language* (UML) and the *Web Ontology Language* (OWL) are used for conceptual modeling. Currently, a push towards the use of OWL can be observed. However, whereas UML has been an established conceptual modeling language for decades, OWL struggles with adoption and tooling. Academic research has attempted to leverage existing UML models by automatically transforming UML models into OWL ontologies. But despite research spanning over a decade, there is no consensus on how UML should be transformed into OWL. We observed that most studies focus solely on unidirectional transformations from UML to OWL, and tools implementing the transformations are often missing, outdated or abandoned. In addition, there is a lack of case studies that evaluate the quality of the proposed transformations. We have implemented a *bidirectional metamodel based transformation tool* between UML and OWL to contribute to a better understanding of to what extent it is possible to automatically transform UML models into OWL ontologies and vice versa. Based on roundtrip transformations we found out that there is a significant overlap between UML and OWL, making it possible to automatically transform a significant part of UML models into *syntactically* valid OWL ontologies and vice versa. But, there is also a significant loss of information due to a lack of features. Finally, using case studies we found that despite being syntactically valid, peculiarities exist in automatically transformed models and ontologies. These peculiarities exist due to slight differences in semantics of similar constructs, and differences in modeling approaches between UML and OWL.

Acknowledgements

First of all, I would like to thank the members of my graduation committee: Luís Ferreira Pires, Ansgar Fehnker, Joep van Genuchten, Michiel Stornebrink and Paul Stapersma for guiding me through this thesis writing process. Their feedback, insights, comments and suggestions have contributed greatly to the writing, readability and quality of this thesis. I am also grateful to Marcel Olij, for his practical suggestions and help. Second, I would like to thank my friends for their help, support and company. Thank you for making my time as a student so enjoyable. Last, but definitely not least, I want to thank my family and girlfriend, for their unconditional love and support.

Executive Summary

This master's thesis was motivated by the Linked Energy Data (LinkED) project, a collaboration between TNO, Alliander and Enexis. In LinkED, they identified that Web Ontology Language (OWL) ontologies can facilitate the publication of meaningful information. However, most existing domain models used in utility companies are maintained in the Unified Modeling Language (UML). To leverage existing modeling work done in UML, a proof-of-concept of an UML to OWL metamodel transformation was implemented. This thesis contributes to metamodel transformations between UML and OWL by evaluating to what extent it is possible to automatically transform between UML and OWL.

Problem statement

We evaluated the academic literature on model transformations between UML and OWL. We found out that most studies propose similar mappings but vary in degree of detail. We observed multiple problems in the current academic literature. First, most studies focus solely on unidirectional transformations from UML to OWL, making verification of the transformation rules difficult. Second, there is a lack of case studies. Finally, tooling lacks behind the mappings. Most existing tools have limited documentation, have not been updated in a long time, or have been abandoned completely.

Contributions

We have implemented a bidirectional metamodel based transformation tool, based on the state-of-the-art mappings presented in the academic literature. Our tool overcomes technical limitations of existing tools by providing support for the parsing and writing of OWL ontologies in commonly used OWL serializations. This allowed us to verify the state-of-the-art mappings more thoroughly, using case studies.

First, we performed roundtrip transformation on a set of test models and ontologies from $UML \rightarrow OWL \rightarrow UML$ and $OWL \rightarrow UML \rightarrow OWL$, to see which constructs are preserved and lost during the transformations. We found out that most UML constructs are preserved after the roundtrip transformation, but details about the structuring and implementation of the model are lost. There are many OWL constructs that are lost after the roundtrip transformation, because OWL is based on description logics, which provides various constructs for which

UML has no equivalent. However, only a small subset of OWL constructs is commonly used in practice. Most of these commonly used constructs are preserved after the roundtrip transformation, which indicates that there is overlap between UML and OWL that is commonly used for conceptual modeling. In addition, it could indicate that this overlap is intuitively easy to understand, whereas more complex OWL constructs require deeper understanding of description logics. An organization could benefit from using the intersection of constructs that is relatively easy to understand, because this could result in models that are easy to understand.

Second, that constructs are preserved after a roundtrip transformation does not necessarily mean that the automatically transformed models or ontologies make sense. We manually evaluated a few models and ontologies and found several peculiarities that exist due to slight differences in semantics of similar constructs, and differences in modeling approaches between UML and OWL.

Conclusion

To conclude, we found out that in both the UML to OWL transformation and vice versa information is lost. However, there is also significant overlap between UML and OWL that allows automatic transformation. Our tool provides the means to automatically transform this overlap between UML and OWL, which prevents having to remodel everything by hand when switching between modeling languages.

Contents

Chapter 1

Introduction

This chapter sets the stage for this master's thesis. Section 1.1 motivates why model transformations between the static elements of UML class diagrams and OWL are important. Section 1.2 introduces limitations and problems in the current body of academic research. Section 1.3 presents research questions according to the introduced problems. Section 1.4 presents the approach we took to answer these research questions. Finally, the outline of this thesis is given in Section 1.5.

1.1 Motivation

Both the *Unified Modeling Language* (UML) [29] and the *Web Ontology Language* (OWL) [22] are used for conceptual modeling, but each of them originated from a different paradigm. Atkinson and Kiko [19] describe UML as the flagship language of the *Model-Driven Engineering* (MDE) paradigm, which places models at a central position in the software development process, and OWL as the flagship language of the *ontology engineering* paradigm, which originated from the artificial intelligence community. Both UML and OWL have their own benefits. Whereas UML has an intuitive graphical representation, OWL has formal semantics, which allows reasoning.

OWL plays an important role in the *Semantic Web*. The Semantic Web was introduced by Tim Berners-Lee as an extension of the current World Wide Web, with the aim of enabling machines to comprehend web content [3]. One of the key concepts to achieve the Semantic Web is *Linked Data*, which is data that are modeled in the *Resource Description Framework* (RDF) and semantically annotated with OWL ontologies [37]. Currently, a push towards Linked Data can be observed. Various organizations, such as the Dutch and Flemish governments have initiatives to make data accessible as Linked Data[1,2], hence making the availability of relevant OWL ontologies ever more important.

However, the availability of OWL ontologies is currently a bottleneck, as people have observed that tooling and adoption of OWL is limited [25, 37]. UML on the other hand, has been an established modeling language for conceptual domain models and industry standards for decades [25]. In order to leverage existing modeling work done in UML, aligning UML and OWL has become an active area of research. Specifically, automatically transforming UML models into OWL ontologies.

1.2 Problem statement

This master's thesis builds on the *Linked Energy Data* project (LinkED), a collaboration between *TNO*[3], *Alliander*[4] and *Enexis*[5]. Within LinkED, a method was developed to transform the static elements of UML class diagrams into OWL ontologies using metamodel transformations. UML consists of a variety of different diagrams. The static elements of class diagrams are used for conceptual modeling. In this thesis, we refer to the static elements of UML class diagrams when referring to UML. LinkED was not the first attempt to automatically transform UML into OWL. As early as 2006, researchers have investigated the possibility of automatically transforming UML into OWL. However, as others have observed, implementations of transformations tools often have limited documentation, do not produce valid output, have not been updated in a long time, or have been abandoned completely [10]. LinkED attempted to fill this gap, by documenting and implementing a transformation tool based on the UML to OWL mappings as suggested by the *Ontology Definition Metamodel* (ODM) [25].

In addition to technical limitations, there is still no consensus on how UML should be transformed into OWL and vice versa. A study from 2016 mentions that there are still no effective proposals for UML to OWL transformations that could be considered standard methods that preserve the original structure of the source UML diagram [13]. More recently, a 2019 study states that even though there are many publications on UML to OWL transformations, to the best of the authors knowledge, there is no study that investigates a complete mapping emphasized by pragmatic needs [30]. Thus, despite various publications spanning one and a half decade, there is still no consensus on how UML should be transformed into OWL and vice versa. ODM can help us understand why there

[1] https://www.omgevingswetportaal.nl/
[2] https://overheid.vlaanderen.be/producten-diensten/OSLO2
[3] https://www.tno.nl
[4] https://www.alliander.com
[5] https://www.enexis.nl

is still no consensus on how to transform UML into OWL and vice versa. In addition to their proposed mappings between UML and OWL, ODM states the following:

> "mappings based solely on the general structure of the languages will often lead to less than ideal choices for mapping some structures. Any particular mapping project will have additional constraints arising from the structure of the particular models to be mapped and the purposes of the project, so will very likely make different mapping choices than those provided in the ODM. [27]"

This is one of the problems in the current body of research on mappings between UML and OWL. Most studies present mappings solely based on the general structure of UML and OWL. In addition, most studies present only unidirectional transformations from UML and OWL, which makes it difficult to verify whether information is lost during the transformation. Finally, most studies do not verify their mappings using case studies. This is evident in projects that attempted to use OWL ontologies that were created by automatically transforming UML models. For example, within the NEN3610[6], a standard for the exchange of geo-information, automatically generated ontologies were evaluated. They state that although automatic transformations between UML and OWL can result in correct ontologies, these ontologies have little practical value, because the ontologies are modeled using an UML perspective rather than a Linked Data perspective[7]. Thus, although there are many studies that provide some mappings between UML and OWL, implementations are often missing, outdated or abandoned, not applicable in practice and lack adequate verification.

1.3 Research questions

This master's thesis aims to contribute to the problems mentioned before, by examining the difficulties of transforming UML to OWL and vice versa, and why these difficulties exist. The overarching research question guiding this thesis is:

> **Research question**: To what extent is it possible to automatically transform UML models into OWL ontologies and vice versa?

To tackle the main research question the following subquestions are answered:

> **Subquestion 1**: What are the state-of-the-art mappings between UML and OWL?

[6] https://geonovum.github.io/NEN3610-Linkeddata/#iso19150-2
[7] See Footnote 1

> **Subquestion 2**: What are the most commonly used constructs in UML and OWL?

SQ1 and **SQ2** helped us evaluate the current state-of-the-art in model transformations between UML and OWL, and get a better understanding of whether these transformations are relevant for models and ontologies used in practice.

> **Subquestion 3**: How can we implement a metamodel based bidirectional transformation tool between UML and OWL?

> **Subquestion 4**: What information is lost in a roundtrip transformation from UML to OWL and OWL to UML based on the state-of-the-art mappings?

> **Subquestion 5**: What peculiarities exist in automatically transformed UML models and OWL ontologies?

SQ4 and **SQ5** forced us to move away from evaluating transformations based on the general structures of UML and OWL, to examining the transformations based on case studies.

1.4 Approach

Part of this research is performed at TNO and Alliander, the Dutch organization for applied scientific research and a Dutch utility company. Because of the applied nature of these organizations, this thesis takes a rather pragmatic approach. The following steps were taken to answer the research questions:

1. Study the background information on important concepts used throughout this thesis.
2. Gather and analyze the current academic research on model transformations between UML and OWL.
3. Gather test sets of relevant UML models and OWL ontologies, and count the metamodel constructs used in these test sets.
4. Implement a bidirectional transformation tool between UML and OWL, based on the state-of-the-art mappings between UML and OWL presented in the literature, and the metamodel constructs used in practice.
5. Use our transformation tool to perform roundtrip transformations from $UML \rightarrow OWL \rightarrow UML$, and $OWL \rightarrow UML \rightarrow OWL$, in order to evaluate whether information is lost in the roundtrip transformations, and if information is lost, evaluate why this is the case.

6. Use our transformation tool to evaluate automatically transformed UML models and OWL ontologies. In this step, we manually evaluate whether there are any peculiarities in the automatically transformed results. This requires a more in-depth evaluation of the epistemic differences in modeling approach between UML and OWL.

1.5 Outline

This thesis is structured in the following manner. *Chapter 2* discusses the background information on MDE, the Semantic Web and LinkED (Step 1). *Chapter 3* answers **SQ1**, by giving an overview and comparing proposed mappings from the literature (Step 2). *Chapter 4* answers **SQ2**, by presenting an overview of used metamodel construct in a test set of relevant UML models and OWL ontologies (Step 3). *Chapter 5* gives a technical overview of our implementation of a bidirectional transformation tool between UML and OWL and answers **SQ3** (Step 4). *Chapter 6* answers **SQ4** by presenting the evaluation of the roundtrip transformations (Step 5). *Chapter 7* answers **SQ5** by discussing the evaluation of the automatically transformed UML models and OWL ontologies (Step 6). Finally, *Chapter 8* concludes by answering the main research question, discussing limitations, and providing directions for future work.

Chapter 2

Background

This chapter introduces the most important concepts used throughout this thesis. Section 2.1 gives an overview of Model-Driven Engineering. Section 2.2 briefly discusses the Semantic Web, in which OWL plays an important role. Finally, Section 2.3 gives a brief overview of LinkED.

2.1 Model-Driven Engineering

This section gives an overview of *Model-Driven Engineering* (MDE). Brambilla et al. [5] observed that a common problem that practitioners encounter when they first enter the model-driven universe, is differentiating between all the different acronyms. They define *Model-Driven Development* (MDD) as a development paradigm that uses models as primary artifacts in the development process. The *Model-Driven Architecture* (MDA) has been proposed by the Object Management Group (OMG), which describes their vision of MDD. Last, they describe *Model-Driven Engineering* (MDE) as a superset of MDD, because it goes beyond development activities and includes other modeling tasks in the complete software engineering process.

2.1.1 MDE Rationale

MDE was introduced as an approach to deal with the complexity of developing and maintaining complex software. Throughout the history of software engineering, researchers and developers have been creating abstractions to help them program in terms of their design intent, instead of the underlying computing environment [31]. For example, programming in assembly instead of writing machine code. However, these abstractions still had a computing-oriented focus, rather than a problem space focus [31]. As a consequence, the computing solution is often intertwined with the problem solution.

Due to the intertwining of the computing and the problem solution, software engineers encountered various integration, interoperability and maintenance problems. Software technology is constantly changing, for instance, due to new middleware platforms. As a result of the intertwining of the computing and the problem solution, software engineers often spend considerable amounts of time to port applications to different platforms, or newer versions of platforms [31]. For example, porting a Java application to a web-based application can take a considerable amount of time if the solution has platform-specific dependencies, which require a different system architecture on a different platform. Furthermore, if only the computing solution exists, knowledge about the problem solution might get lost when the developer of a system leaves the organization or company.

MDE aims to solve these integration, interoperability and maintenance problems, by separating functionality, or behaviour, from specific platforms and technologies. MDE does this by considering models not only as documentation artifacts, but as central artifacts with a direct role in the software engineering process [5, 7, 18]. Vital in this MDE approach is the distinction between *platform independent models* (PIM) and *platform-specific models* (PSM) [18]. PIMs are formal specifications of the structure and behaviour of a system. MDE claims that by using PIMs that abstract away platform-specific details, integration and interoperability across various platforms should become easier to produce [18].

2.1.2 OMG Model-Driven Architecture

The Model-Driven Architecture (MDA) is the specific view of Model-Driven Development (MDD) as proposed by the Object Management Group (OMG), which is one of the most popular modeling frameworks in the industry [5]. In this section, MDE is discussed in more detail using this architecture. MDA is used because of the importance of the OMG in the software industry.

Models and Metamodels

Models have a central role in MDA and MDE. MDA describes models used in the context of software engineering as “information selectively representing some aspect of a system based on a specific set of concerns” [26]. These models only have value within MDE if they are expressed in a language that is understood by stakeholders and relevant technologies [26]. For the Model-Driven Architecture, this implies that “the structure, terms, notations, syntax, semantics, and integrity rules of the information in the model are well defined and consistently represented” [26]. In other words, the model should be expressed using a commonly agreed upon modeling language. Within the object-oriented modeling world, the most well-known general purpose modeling language is the Unified Modeling Language (UML) [29]. MDA prescribes that a modeling language should be defined in a model [26]. A model that defines a modeling language is called a *metamodel*, and this metamodel is also expressed in a modeling language. Figure 2.1 illustrates this concept of metamodel.

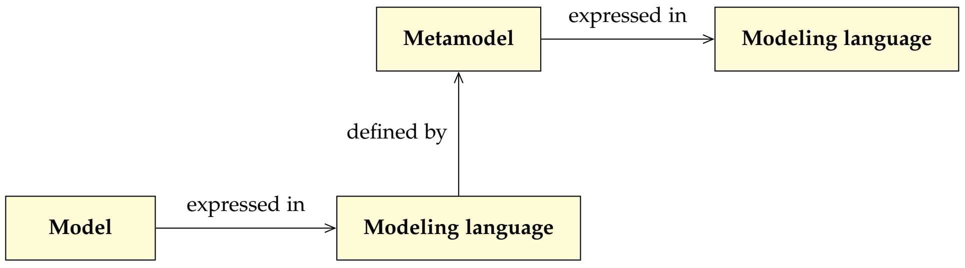


Figure 2.1: Illustration of Models and Metamodels

This definition of metamodel could result in a problem: if each metamodel is expressed using a modeling language, this could lead to a never ending hierarchy of meta-meta models. A common solution to this problem is to let a modeling language define itself at a certain level in the hierarchy [7].

Meta Object Facility

The Model-Driven Architecture is not a single standard, but a family of standards [26]. *Meta Object Facility* (MOF) lies at the foundation of MDA. MOF is used to define metamodels.

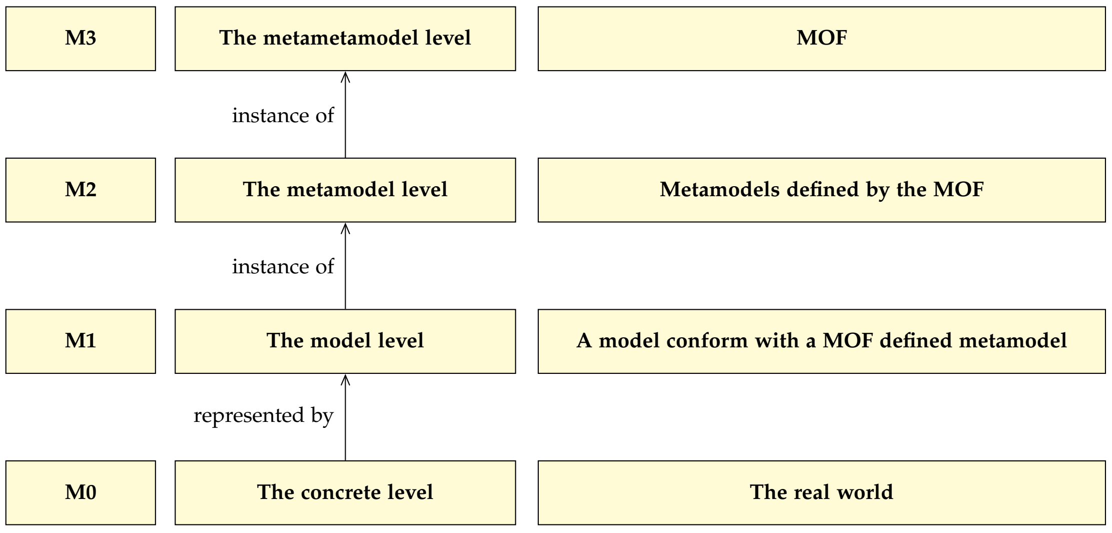


Figure 2.2: OMG metamodel hierarchy as dicussed in [4]

MOF is used in the OMG metamodel hierarchy, which is depicted in Figure 2.2. MOF is positioned at the highest level in the hierarchy, namely the metametamodel (M3) layer. It is used to model metamodels, or in other words, it is the language that is used to define the modeling languages. MOF is defined using MOF to make M3 the top-layer in the OMG metamodel hierarchy

[7]. The M2 layer contains modeling languages, which are instances of MOF. The most prominent example of a modeling language instantiated from MOF is UML. An M1 model is an instance of a M2 layer model that provides a language and representation of a user-domain of interest, which conforms to a modeling language as defined at layer M2. Finally, the M0 is the concrete level, which contains real situations that are represented by models on the M1 layer. Another interpretation of the M0 level states that the M0 level contains objects that are instances of the M1 level objects [1]. To keep the scope of this report clear, the M0 level is described as defined in [4], as illustrated in Figure 2.2.

Model transformations

Besides models and metamodels, MDA describes another key element of MDE: model transformations, which define mappings between different source and target models. In the literature about model transformations two main types of model transformations are identified: model-to-model and model-to-text [7].

Model-to-model transformations Generally speaking, a model transformation takes one or multiple source models and converts them into one or multiple target models. The UML to OWL transformation developed within the LinkED project is an example of a model-to-model transformation. Model transformations work by specifying transformation rules from metamodel elements of the source model to the metamodel elements of the target metamodel. These transformation rules themselves can be viewed as models, which are instances of a transformation language. Figure 2.3 illustrates the model transformation process.

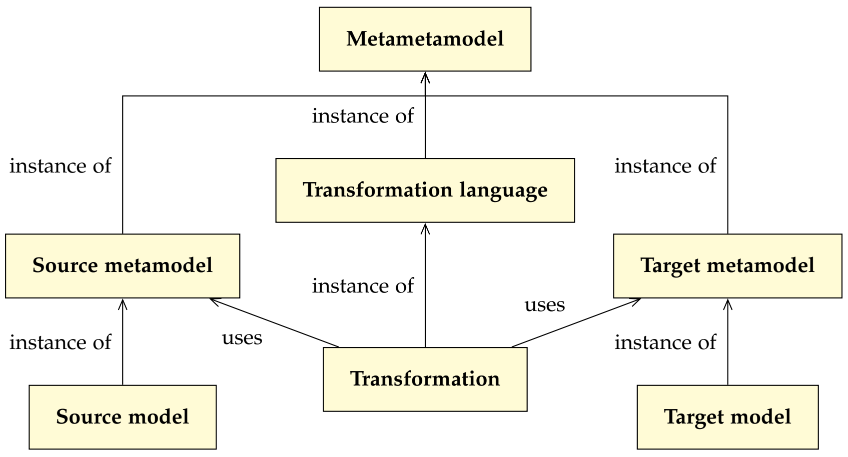


Figure 2.3: Illustration of the model transformation process

Mainstream programming languages such as Java, or C# can in principle be used as transformation language. However, languages developed specifically for model transformations exist. MDA recommends the OMG *Query/View/Transform*

(QVT) [28] standard for model-to-model transformations [26]. QVT consists of two declarative languages: QVTr – Relations Language and QVTc – Core Language, and one imperative language: QVTo – Operational Mapping Language. Collectively, these languages provide a hybrid language of both declarative and imperative constructs.

Model-to-text transformations One could argue that model-to-text transformations are actually model-to-model transformation as well, since text could be seen as just another type of model. However, model transformations that generate software artifacts are generally considered a distinct type of model transformation (model-to-text) [7]. One of the most well-known type of model-to-text transformation is code generation from models[7], but model-to-text transformations are not limited to code generation. Other examples of model-to-text transformations include models to XML, JSON and HTML transformations.

2.1.3 MDE tools

Although MDA defines useful standards for MDE, the implementation of these standards is realized by third-parties. Various proprietary tools for MDE exists, but the open source *Eclipse Modeling Framework* (EMF) has become one of the most popular tooling platforms for MDE [5]. The LinkED project also leveraged the EMF for their MDE activities. Our bidirectional transformation tool also relies heavily on EMF. Therefore, instead of listing various MDE tools, EMF is discussed in more detail in this section.

The core of EMF is Ecore, which is an implementation of the most essential constructs of MOF, namely the Essential MOF (EMOF). In EMF, Ecore can exist on both the M3 and M2 level of the metamodel hierarchy, as it allows for the definition of metamodels, but may also be used to define models at the M1 level. This is different from MOF, which may only be used to define metamodels. EMF comes with capabilities to serialize and deserialize models defined in Ecore to and from XMI [5]. Besides Ecore, there are several tools and frameworks developed on top of EMF that enable model-to-model and model-to-text transformations.

For model-to-model transformations EMF offers a QVT implementation for each sub language. EMF also provides an alternative to QVT, namely the ATLAS transformation language (ATL) [17]. ATL is a hybrid language, containing a mix of declarative and imperative constructs. ATL was inspired by QVT and therefore conforms to OMG standards such as XMI and OCL.

For model-to-text transformations EMF provides languages such as Acceleo [23] and Xpand [8]. However, since this research essay is primarily concerned with model transformations between UML and OWL, model-to-text transformations are less relevant here than model-to-model transformations.

2.2 The Semantic Web

In a famous article from 2001, Tim Berners-Lee introduced the concept of the Semantic Web [3]. Berners-Lee discussed how at that time, Web content was being designed for humans to read. Consequently, computers had no reliable way to process the semantics of Web content. He envisioned the Semantic Web as an extension of the World Wide Web, in which information is given well-defined meaning. The goal of the Semantic Web was to enable computers to comprehend the information available on the Web.

2.2.1 Technology stack

An overview of the Semantic Web technology stack is given in Figure 2.4. Only the technologies relevant for the scope of this thesis are discussed.

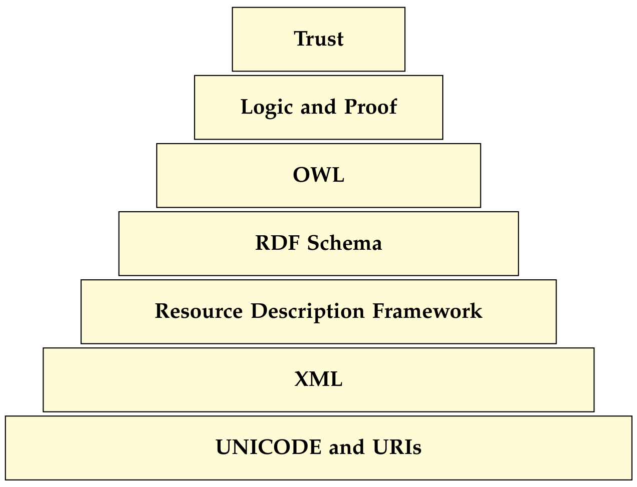


Figure 2.4: Semantic Web technology stack as introduced in [21]

XML The eXtensible Markup language (XML) allows users to add arbitrary structure to their documents, using tags or labels [3]. However, although XML provides the means to structure documents using tags, it does not represent what these structures mean [3, 21]. In other words, XML is used to structure data, but does not say what the data mean.

RDF and RDFS What data mean is expressed by the Resource Description Framework (RDF). RDF is a simple metadata representation framework, which uses a graph model to describe relationships between resources [21]. RDF allows the definition of information in sets of triples, which can be written using XML tags [3]. For example, using RDF one can state that "William Shakespeare is the author of Hamlet".

RDF Schema (RDFS) extends RDF by adding class- and property-structuring capabilities [16]. It provides the basic capabilities for the creation of so-called ontologies. Berners-Lee defines ontologies as "a document or file that formally defines the relations among terms" [3]. Using RDFS one can go beyond the capabilities of RDF, and capture relations among terms. For instance, one can define that William Shakespeare is an author, and using the rdfs:subClassOf construct define that an author is a person.

OWL and OWL2 RDFS provides the basic capabilities for the definition of ontologies. OWL takes these basic capabilities and extends them. OWL is a family of three language of increasing expressive power: OWL Lite, OWL DL, and OWL Full [22]. One of its major extensions is the capability to provide restrictions on how properties behave that are local to a class [16]. Motik et al. [9] discuss how although OWL has been successful, a number of problems have been identified in its design. They discuss limitations in expressivity with respect to qualified cardinality restrictions, relational and datatype expressivity. OWL2 extends and revises OWL with the aim of resolving these limitations.

2.2.2 Push towards ontologies

Although the complete vision of the Semantic Web is quite far from being achieved, Semantic Web technologies are gaining in popularity. This section discusses a number of reasons for the push towards OWL ontologies specifically.

Data integration and alignment In 2006, Berners-Lee et al. [32] observed an increased need for data integration, shared semantics and a web of data. They discussed how multiple heterogeneous datasets from various sources have to be integrated. Ontologies are commonly used to integrate datasets. For instance, OWL is extensively used in the life sciences community for ontology development and data interchange [9, 32].

Reasoners and inferences Besides data integration and alignment, another appealing aspect of ontologies is the support for automatic reasoning and inferences. The OWL2 semantics is based on Description logic [9], which allows the use of reasoners to deduce implicit facts. Because of this capability, there is no need to specify explicitly what is already implicitly available in the ontology. This reduces redundancy in the knowledge base and makes ontologies easier to maintain [14]. Many reasoners for OWL have been developed, like HermiT [33] and Pellet [35].

Government push Bauer et al. [2] discuss the Open Government Data movement, which they describe as the movement that aims to open up data and information from governmental institutions and other relevant stakeholders such as business/industry, citizens, NPOs and NGOs, for use and re-use by civil society, economy media, academia as well as politicians and public administrators. Bauer et al. argue that in order to fully benefit from Open Data, information should be put into context, by moving towards Linked Open Data. They suggest that data should be published in RDF, and various links should be created between different datasets, creating a web of data, or Linked Data.

2.3 Linked Energy Data

LinkED can be positioned in the movement towards OWL ontologies. LinkED was motivated by the revised Environment and Planning Act (omgevingswet)[1], which makes it mandatory to publish more and more data from the energy sector. However, instead of publishing raw data, data must be published in a meaningful manner. Like the Platform Linked Data Nederland[2], within LinkED they identified that Linked Data can facilitate the publication of meaningful information. More specifically, this entails publishing data on the internet conform with an ontology defined in OWL. Currently, Enexis is working on publishing their Open Data as Linked Open Data. Thus, within LinkED the need for conceptual domain models defined in OWL ontologies has been observed.

However, most existing domain models used in utility companies are often maintained in UML or easily convertible into UML. To prevent having to remodel the existing domain knowledge available in these models in OWL, within LinkED a method was developed to automatically transform UML models to OWL ontologies. Furthermore, Olije et al. [25] discuss that it is unlikely that existing modeling communities will leave model engineering in UML in favour of ontology engineering. Therefore, within the LinkED project, an attempt was made to develop a method to simultaneously maintain UML models and corresponding OWL ontologies.

[1]https://www.government.nl/topics/spatial-planning-and-infrastructure/revision-of-environment-planning-laws

[2]http://www.pilod.nl/wiki/Boek_5/Thema:_De_rol_van_Linked_Data_en_REST_APIs_bij_de_totstandkoming_van_de_Omgevingswet

2.3.1 Achievements

Within LinkED, a method was developed to transform conceptual domain models maintained in Sparx Enterprise Architect (EA) into OWL ontologies. The source code of the transformation method is available at the Linked Energy Data Transformations Github repository[3]. The transformation process consists of the following steps:

Preprocessing Models maintained in EA cannot be directly loaded in the Eclipse EMF environment. In order to use EA UML models, the models are transformed into an Eclipse UML2 (Ecore) compatible UML model using an XSLT transformation.

$$\begin{tikzcd}\boxed{\textbf{UML}_{EA}} \arrow[r, "\text{XSLT}"] & \boxed{\textbf{UML}_{Ecore}}\end{tikzcd}$$

Figure 2.5: Preprocessing EA UML models

UML to OWL transformation The preprocessed UML models are transformed into OWL ontologies using an ATL transformation, based on the transformation suggested by ODM [27].

$$\begin{tikzcd}\boxed{\textbf{UML}_{Ecore}} \arrow[r, "\text{ATL}"] & \boxed{\textbf{OWL}_{Ecore}}\end{tikzcd}$$

Figure 2.6: UML to OWL using ATL

OWL serialization Using the transformation as illustrated in Figure 2.6, UML models that conform with the Eclipse (Ecore) UML2 metamodel are transformed into OWL ontologies that conform with an OWL Ecore metamodel[4]. Although the transformed OWL ontologies correctly represent an ontology, they are not serialized in an accepted OWL serialization. Using another ATL transformation and a dedicated projector the OWL$_{Ecore}$ is transformed into valid OWL$_{RDF/XML}$.

$$\begin{tikzcd}\boxed{\textbf{OWL}_{Ecore}} \arrow[r, "\text{ATL}"] & \boxed{\textbf{XML}_{Ecore}} \arrow[r, "\text{projector.jar}"] & \boxed{\textbf{OWL}_{RDF/XML}}\end{tikzcd}$$

Figure 2.7: OWL serialization process

Using this approach, a number of UML models maintained in EA were successfully transformed into syntactically valid OWL ontologies. The LinkED project also briefly examined the possibility of using an UML profile (a generic extension mechanism for UML) to express OWL ontologies in UML. They did

[3]https://github.com/linkedenergydata/transformations

[4]Our transformation tool uses a different OWL metamodel, a corrected version of the metamodel used here

this by examining the well-known pizza ontology[5], and recreating this pizza ontology in UML by hand. They found a number of OWL constructs, such as equivalence between classes, that they were unable to express in UML. Olij et al. [25] created an UML profile that aims to bridge the semantic gap between UML and OWL, while leaving the original UML model intact.

2.3.2 Limitations

It seems that the LinkED transformation method is, at least syntactically, able to transform UML models into OWL ontologies. However, due to the currently used serialization method it is not possible to transform OWL ontologies back to UML models. Because the used projector can only transform from XML_{Ecore} to $OWL_{RDF/XML}$ and not back from $OWL_{RDF/XML}$ to XML_{Ecore}. In addition, only a subset of the transformation rules suggested by ODM have been implemented, and some implementations differ from the ODM suggestions. Finally, note that the LinkED transformation method was based on an ATL use case[6] that transformed between UML and OWL. The aim was to create a proof of concept of an UML to OWL metamodel transformation, writing a jar that can transform $OWL_{RDF/XML}$ back to XML_{Ecore} was outside the scope of the project. The LinkED transformation method could be improved significantly, which we have attempted as described in Chapter 5.

[5] https://protege.stanford.edu/ontologies/pizza/pizza.owl

[6] http://www.eclipse.org/atl/usecases/ODMImplementation/

Chapter 3

Related work

Besides LinkED, various studies have investigated model transformations between the static elements of UML Class diagrams and OWL. This chapter discusses this related work. Section 3.1 and 3.2 give an overview of mappings between UML and OWL constructs, and briefly explains the purpose of the constructs. We use the OWL Functional Syntax[1] to notate OWL examples. Section 3.3 discusses tools that implement these mappings. Finally, Section 3.4 concludes this chapter by answering **SQ1**: What are the state-of-the-art mappings between UML and OWL?

3.1 UML to OWL

In a recent paper, Sadowska and Huzar [30] give an overview of mappings from UML to OWL, based on a literature study that examined 18 studies of transformations between UML and OWL. This section presents a high level overview of their mappings, and compares these to mappings proposed in OMG's ODM [27] and Jesper Zedlitz's PhD thesis [38]. Appendix A.1 presents the proposed mappings in more detail. Since the LinkED mappings are a subset of the ODM mappings we do not include these separately in our comparison, but do mention where the mappings of LinkED differ from ODM.

3.1.1 UML Packages

Namespaces in UML are represented using the `Package` construct. Packages provide the main structuring and organizing capability of UML [29]. Most UML constructs are packageble elements, which can be contained by a `Package`. OWL ontologies are structured using the `Ontology` and `Axiom` constructs. In [27, 30, 38], each `Package` is mapped to a new `Ontology`. LinkED differs from this, in that they map each `Package` in an UML model to the same `Ontology` construct.

[1] https://www.w3.org/TR/owl2-syntax/#Ontologies

3.1.2 UML Class

Both UML and OWL have a `Class` construct, which is used to model a set of objects. For instance, one could create an UML or OWL `Class` that represents students. Figure 3.1 and 3.2 illustrate the relevant metaconstructs to define a `Class` in UML and OWL.

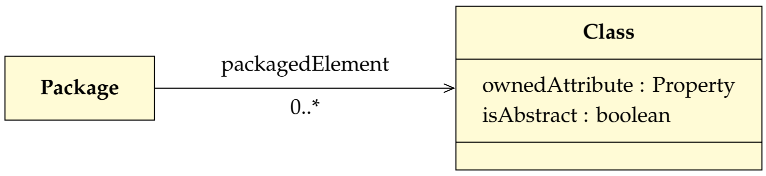


Figure 3.1: Simplified[2] metamodel of UML `Class`

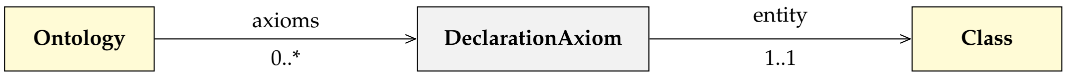
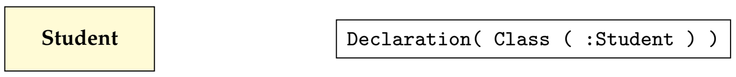


Figure 3.2: Simplified metamodel of OWL `Class`

In [27, 30, 38], an UML `Class` is mapped to an OWL `Class`. For each UML `Class` in an UML `Package`, an OWL `DeclarationAxiom` of an OWL `Class` is added to an `Ontology`. Figure 3.3 illustrates the student example in UML and its transformed OWL equivalent.



```
Declaration( Class ( :Student ) )
```

Figure 3.3: Student `Class` example in UML and OWL

Abstract Classes

It is possible to declare `Abstract Classes` in UML using the `isAbstract` property (see Figure 3.1). This entails that it is not possible to instantiate the `Class`. In [27, 30], no mapping for `Abstract Classes` is suggested, since they state that OWL lacks this feature. In [38], `Abstract Classes` are mapped to regular `Classes`, but they argue that it cannot be ensured that these OWL `Classes` are not instantiated. We discuss the usage and semantics of `Abstract Classes` in more detail in Section 6.1.2.

Attributes

In UML, a `Class` can have `ownedAttributes` (See Figure 3.1). An UML `Attribute` has an UML `Property` as `Type`. Both UML and OWL have constructs to represent `Properties`. Figure 3.4 and 3.5 illustrate how `Properties` are modeled in UML and OWL, respectively. An example of an `Attribute` or `Property` is a student number of a student.

[2]We omitted some constructs and properties to improve readability

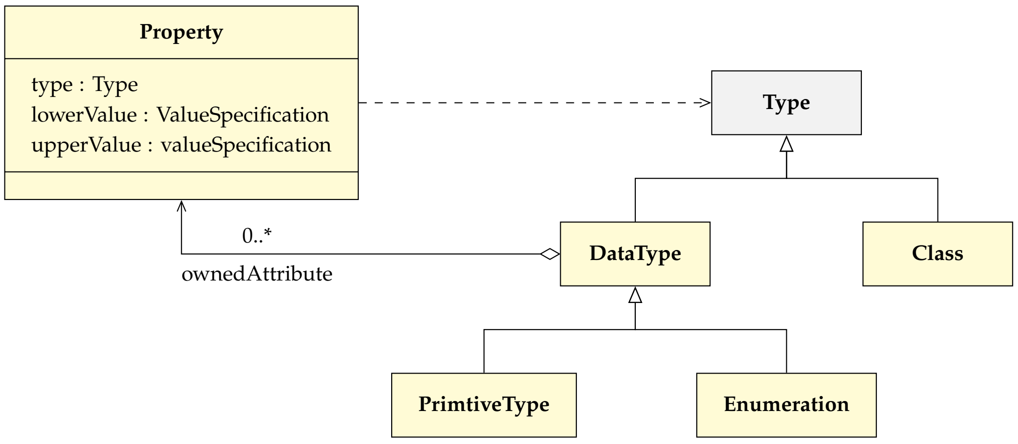


Figure 3.4: Simplified metamodel of UML `Property` and `Type`

While UML has one metaconstruct for `Property`, OWL has a seperate construct for `ObjectProperties` and `DataProperties`. In [27, 30, 38], `Properties` are mapped to OWL `Object` or `DataProperties` depending on the `Type` of the `Property`. If the `Type` is a complex `DataType` (that is when a `DataType` has `ownedAttributes`), or the `Type` is a `Class`, an `ObjectProperty` is used in the mapping. If the `Type` is a `PrimitiveType` or `Enumeration`, a `DataProperty` is used.

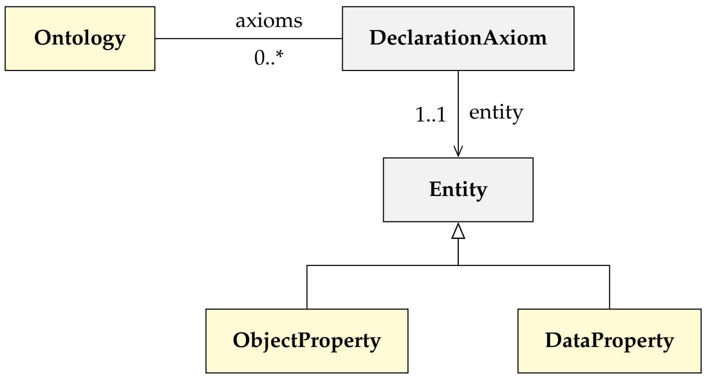
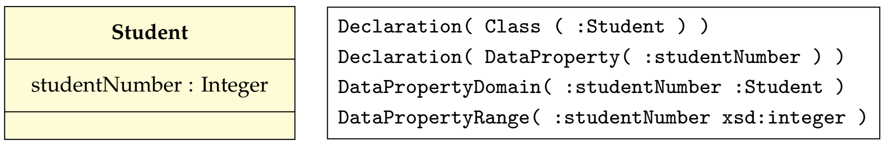


Figure 3.5: Simplified metamodel of OWL `Properties`

In addition, the domain and range of the UML `Property` are mapped to OWL `ObjectPropertyRange` and `ObjectPropertyDomain`, or `DataPropertyRange` and `DataPropertyDomain` constructs, respectively. In [38], the suggestion is made to make OWL `Properties` disjoint, to prevent OWL from interpreting UML `Properties` with the same name as semantically equivalent. Within LinkED each `Property` was prefixed with the containing `Class` name to achieve the same. Figure 3.6 illustrates the transformation of a student `Class` with `Attribute` student number into OWL.



```
Declaration( Class ( :Student ) )
Declaration( DataProperty( :studentNumber ) )
DataPropertyDomain( :studentNumber :Student )
DataPropertyRange( :studentNumber xsd:integer )
```

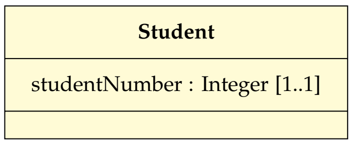
Figure 3.6: Student number example in UML and OWL

DataTypes

As Figure 3.4 illustrates, UML differentiates between three types of datatypes: `DataType`, `PrimitiveType` and `Enumeration`. In [30, 38], a `DataType` is mapped to a regular OWL `Class` with an OWL `HasKey` axiom. A `PrimitiveType` is mapped to a corresponding `XSD DataType`, or a new OWL `DataTypeDefinition` if no equivalent `XSD DataType` exists. In [27, 30, 38], `Enumerations` are mapped to `DataTypeDefinition` constructs with a `DataOneOf` expression. LinkED mapped `Enumerations` using an `Instanceof` expression instead of a `DataOneOf` expression.

Cardinality constraints

In UML, it is possible to specify cardinality constraints on properties using the `lowerValue` and `upperValue` properties (see Figure 3.4). Cardinality constraints are used to constrain the number of values that may be contained by an property [29]. For instance, one could specify that a student can only have one student number. In [27, 30, 38], UML cardinality constraints are mapped to the `Data` or `Object MinCardinality`, `MaxCardinality`, and `ExactCadinality` constructs, depending on whether the `Property` is mapped to an `Object` or `DataProperty`. Figure 3.7 gives an example of this mapping.



```
Declaration( Class ( :Student ) )
Declaration( DataProperty( :studentNumber ) )
DataPropertyDomain( :studentNumber :Student )
DataPropertyRange( :studentNumber xsd:integer )
DataExactCardinality( :studentNumber 1 )
```

Figure 3.7: Cardinality example in UML and OWL

3.1.3 UML Associations

Figure 3.8 illustrates the metamodel constructs relevant to an UML `Associations`. `Associations` refer to `Properties` in their definition.

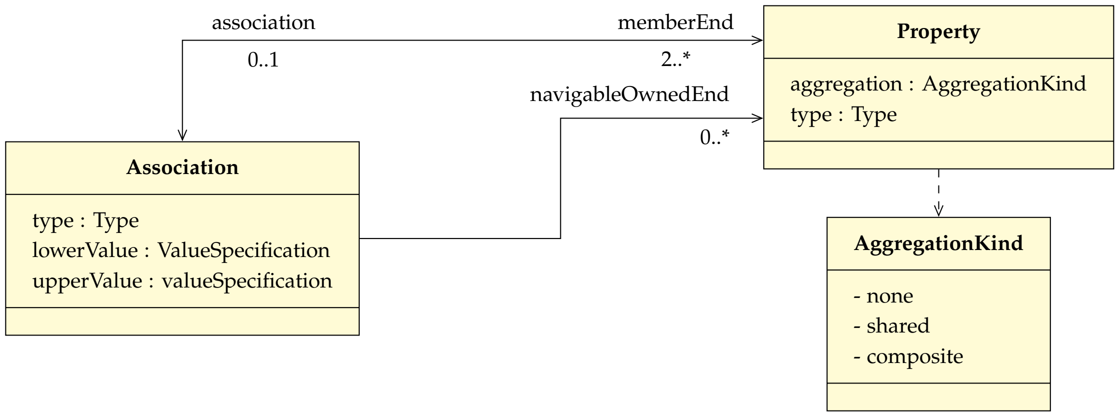
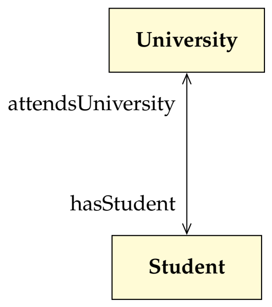


Figure 3.8: Simplified metamodel of UML `Association`

Like Attributes, Associations are used to model relationships between Classifiers such as Classes. We discuss the difference between Attributes and Associations in Section 6.1.2. If an Association has two memberEnds, it is commonly called a Binary Association. An example is the relation between a student and a university. In [30, 38], Binary Associations are mapped to an ObjectProperty for each memberEnd, and an InverseObjectProperties axiom. In [27], different mappings are suggested, depending on the navigableOwnedEnd association. If both the memberEnds of the Associations are navigableOwnedEnds, the mapping is identical to the one suggested in [30, 38]. If only one memberEnd is a navigableOwnedEnd, a single objectProperty is created instead.

Associations with more than two memberEnds are N-ary Associations. In [38], it is suggested that N-ary Associations can be transformed into Binary Associations. The mapping suggested in [30] differs, in that they state that it is not possible to directly transform N-ary Associations and they only present mapping that maps parts of the semantics of an N-ary Association. In addition to Binary and N-ary Associations, [27, 30] present a mapping for Association Classes.

Finally, UML allows the specification of AggregationKind on Properties. Shared represents an UML Aggregation end, and Composite represents an UML Composition. In [27, 30], no mapping for Aggregation and Composition is suggested, as they argue that OWL lacks these features. In [38], mappings for the constraints defined by Aggregations and Compositions are suggested. Figure 3.9 presents an example of the Binary Association mapping.



```
Declaration( Class ( :Student ) )
Declaration( Class ( :University ) )
Declaration( ObjectProperty ( :hasStudent ) )
Declaration( ObjectProperty ( :attendsUniversity ) )
ObjectPropertyDomain ( :hasStudent :University )
ObjectPropertyDomain ( :attendsUniversity :Student )
ObjectPropertyRange ( :hasStudent :Student )
ObjectPropertyRange ( :attendsUniversity :University )
InverseObjectProperties ( :hasStudent :attendsUniversity )
```

Figure 3.9: Binary Association example in UML and OWL

3.1.4 UML Generalizations

GeneralizationSet and Generalization constructs are used to model UML Generalizations, which is illustrated in Figure 3.10. A Generalization can be used to specify that a Classifier inherits from another Classifier. For instance, one can specify that a student inherits from person. Or in other words, that a student is a subset of people. In [27, 30, 38], mappings are suggested for Classifiers of type Class and Association. A Generalization between two

UML `Classes` is mapped to the OWL `SubClassOf` axiom. A `Generalization` between `Associations` is mapped to the OWL `SubPropertyOf` axiom. In addition, [38] presents a mapping for a `Generalization` between `Enumerations`.

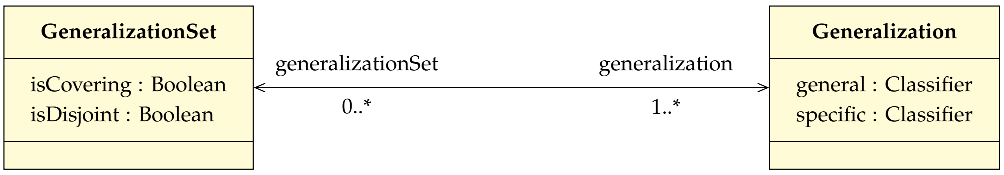
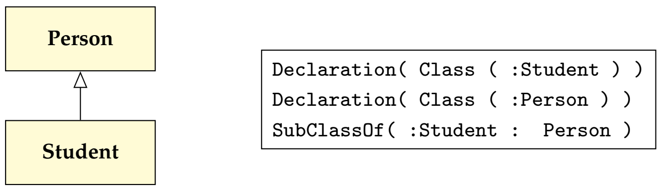


Figure 3.10: Simplified metamodel of UML `Generalizations`

Figure 3.10 illustrates the possibility of specifying whether `Generalizations` in a `GeneralizationSet` are `Covering` or `Disjoint`. Depending on the values of the `isCovering` and `isDisjoint` properties, `GeneralizationSet` is mapped to `DisjointClasses`, `DisjointUnion`, `EquivalentClasses` and `ObjectUnionOf` expressions. Figure 3.11 shows an example of a `Generalization` and its OWL mapping.



```
Declaration( Class ( :Student ) )
Declaration( Class ( :Person ) )
SubClassOf( :Student :  Person )
```

Figure 3.11: `Generalization` example in UML and OWL

3.2 OWL to UML

Sadowska and Huzar only discuss unidirectional mappings from UML to OWL, but Zedlitz and ODM discuss bidirectional mappings. This section gives an overview and comparison of their mappings from OWL to UML. More detail is given in Appendix A.2.

3.2.1 Inverse Mappings

The following mappings from OWL to UML are the inverse of previously discussed mappings in the UML to OWL section:

- In [27, 38], OWL `Ontology` is mapped to UML `Package`.
- In [27, 38], OWL `Classes` are mapped to UML `Classes`.
- In [27, 38], OWL `DataProperties` are mapped to UML `Class Attributes`.
- In [27, 38], OWL `Objectproperties` are mapped to UML `Class Attributes`. Except when the `ObjectProperty` is `InverseFunctional` or has an `Inverse`, then the `ObjectProperty` is mapped to an UML `Association`.

- In [27, 38], OWL `SubObjectProperties` are mapped to UML `Property Generalizations`.
- In [27, 38], OWL `DataOneOf` is mapped to an UML `Enumeration`.
- In [38], OWL `HasValue` is mapped to an UML `Attribute` with a default value equal to the specified `HasValue`.
- In [38], OWL `HasKey` is mapped to UML `Key`.
- In [38], OWL `DataTypes` are mapped to established UML libraries for XML DataTypes, such as the XML DataTypes library of Enterprise Architect.

3.2.2 OWL ClassAxioms

SubClassOf

In Section 3.1.4, `SubClassOf` was only used with OWL `Classes`, which can be mapped to UML `Classes`. Figure 3.12 illustrates the relevant metamodel constructs of `SubClassOf` and a subset of the possible `ClassExpression` types, which are discussed in Section 3.2.3. `SubClassOf` has associations with `ClassExpression`, which do not necessarily have to be `Classses`.

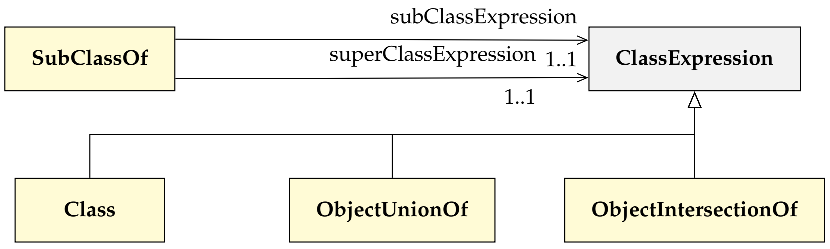


Figure 3.12: Simplified metamodel of `SubClassOf` and a subset of `ClassExpression`

If the `subClassExpression` and `superClassExpression` are of type `Class`, [27, 38] suggest mapping `SubClassOf` to an UML `Generalization`. In [38], a mapping is suggested for a `SubClassOf` with a necessary condition as `ClassExpression`.

EquivalentClasses

Similar to `SubClassOf`, `EquivalentClasses` takes a number of `ClassExpressions`. In [27, 38], `EquivalentClasses` with `ClassExpressions` that are mapped to UML `Classes` are mapped to an UML `Generalization`.

3.2.3 OWL ClassExpressions

`ObjectUnionOf` and `ObjectIntersectionOf` are `ClassExpressions` that have associations with `ClassExpressions`, as illustrated in Figure 3.13. The upcoming mappings are suggested when the associated `ClassExpressions` are of type `Class`.

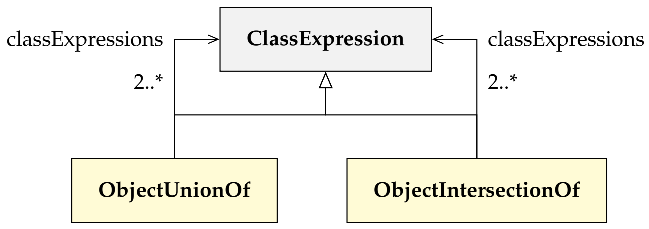
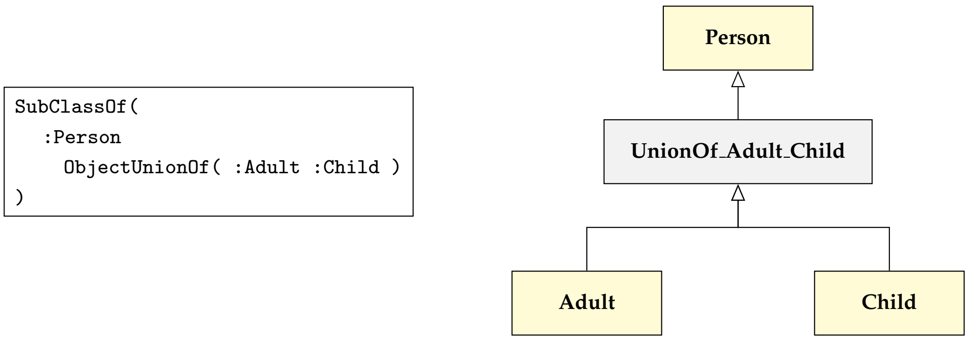


Figure 3.13: Simplified metamodel of `ObjectUnionOf` and `ObjectIntersectionOf`

ObjectUnionOf

An `ObjectUnionOf` expression contains all the instances that are instances of at least one of the `ClassExpressions`. For example, `ObjectUnionOf` could be used to get the set of all individuals that are adults or children. In [27], each `Class` in the `ObjectUnionOf` is mapped to an UML `SubClass` in a `GeneralizationSet` and a helper class that represents the union. In [38] a slightly different mapping is proposed, in which the `SuperClass` in the `GeneralizationSet` is set to `Abstract`. Figure 3.14 gives an example of an `ObjectUnionOf` and the mapping to UML.



```
SubClassOf(
   :Person
     ObjectUnionOf( :Adult :Child )
)
```

Figure 3.14: `ObjectUnionOf` example in OWL and UML

ObjectIntersectionOf

An `ObjectIntersectionOf` expression contains all the instances that are instances of all the `ClassExpressions`. For example `ObjectIntersectionOf( :Person :Student )` contains all the instances that are both a person and a student. In [27, 38], each `Class` in the `ObjectIntersectionOf` is mapped to a `SubClass` in an UML `Generalization` and an `Abstract SuperClass`.

3.2.4 OWL Range and Domain

In [27, 38], three cases of OWL `Domain` and `Ranges` of `Properties` are discussed:

1. One class is declared for `Domain` and `Range`, this is mapped to regular UML `Attributes and Associations`.
2. In OWL it is allowed to declare `Properties` without specifying a `Domain` and `Range`. In this case, the `Domain` and `Range` are specified as `OWL:thing`, which can be mapped to an UML `Class` named Thing.
3. In OWL it is possible to declare more than one `Class` as `Domain` or `Range`. In this case the `Domain` and `Ranges` are mapped to an UML `Generalization` construct as `Range` or `Domain` for UML `Properties`.

3.2.5 OWL Cardinality Constraints

Most of the OWL Cardinality Constraints mappings are the inverse of earlier discussed mappings, namely `Min, Max` and `ExactCardinality` for `DataProperties` and `ObjectProperties`. In [27, 38], mappings for `FunctionalProperty` and `InverseFunctionalProperty` are suggested.

3.3 Transformation tools

Tools are necessary to automate model transformations, which are especially useful for the transformation of large models. Andreas Grunwald [10] gives an extensive overview of existing transformation tools between UML and OWL. He describes many tools that claim to be able to transform UML into OWL, such as CIMTool, CODIP, DIA and OntoStudio. He also mentions a Protegé 4.0 plugin that transforms OWL into UML using the ODM OWL UML profile. However, Grünwald deemed the currently available tools insufficient for his specific requirements: correctly transforming Visual Paradigm UML models to OWL ontologies. Furthermore, he found that most tools were not up to date or disbanded. He presents his own UML to OWL transformation tool that uses a UML metamodel and Java to transform UML into OWL. However, his tool only supports unidirectional transformations from UML to OWL.

Although various UML to OWL transformation tools exist, no working bidirectional transformation tools could be found. ODM [27] and Zedlitz [38] present bidirectional mappings between UML and OWL, but no working implementation could be found. Most tools focus solely on transforming UML into OWL, mainly because they are only interested in extracting information from UML models to use this in OWL ontologies, or due to technical limitations, such as the transformation tools presented by LinkED and Grünwald.

3.4 Conclusion

To answer **SQ1**: What are the state-of-the-art mappings between UML and OWL? We found that the mappings of Sadowska and Huzar [30], ODM [27] and Zedlitz [38] describe the current state-of-the-art mappings between UML and OWL. We gave an overview and comparison of their presented mappings in this chapter, and provided the mappings in more detail in **mappings 1-59** in Appendix A.1 and A.2. We found that most of the time, the different studies propose similar mappings. Sadowska and Huzar include and discuss the mappings presented in a paper based on the thesis of Zedlitz in their work. Zedlitz refers to ODM in his work, explaining the similarity between the mappings of Sadowska and Huzar, ODM and Zedlitz. In some cases, there are slight differences between the mappings, and some studies cover mappings that others do not. We found that Sadowska and Huzar provide the most extensive account of UML to OWL mappings and they include mappings and considerations that ODM and Zedlitz do not. However, they do not consider transformations from OWL to UML at all. Zedlitz gives the most extensive account of OWL to UML mappings. He considered the ODM mappings and in addition, presented different mappings, more details and considerations.

Chapter 4

Evaluating UML and OWL usage

Since this master's thesis aims to analyze the state-of-the-art mappings using case studies, we decided to investigate which UML and OWL constructs are commonly used in UML models and OWL ontologies. Section 4.1 examines UML models used within Alliander and Enexis. Section 4.2 examines representative OWL ontologies. Finally, Section 4.3 concludes this chapter by answering **SQ2**: What are the most commonly used constructs in UML and OWL?

4.1 Evaluating UML

UML consists of a wide variety of 14 diagrams such as class, activity and state diagrams. In this master's thesis, we focus on the static elements of class diagrams, which are commonly used in business and conceptual modeling. Sadowska and Huzar [30] identified a number of UML constructs that are suggested in the literature as most important for conceptual modeling. We evaluated whether these findings corresponded with UML constructs used in the UML models used by Alliander and Enexis.

4.1.1 Preparing a test set

Many UML domain models exist. We focus on UML models used in the energy sector, specifically at Alliander and Enexis. Most of the UML models used within Alliander and Enexis are maintained and distributed in Sparx Enterprise Architect. Although many UML tools serialize their models as XMI, which was introduced as an interchange format, different tools adopt XMI in different variants, making interoperability between UML tools difficult. We encountered this problem as well, as we used the Eclipse UML2 project, which currently provides an implementation of the UML2.5 specification, that is not directly compatible with Sparx Enterprise Architect. Within the LinkED project, XSLT was used to transform Enterprise Architect models to Eclipse UML2 models [25]. Our UML test set consists of the following UML models that were successfully transformed using the LinkED XSLT preprocessor[1]:

- **Common Information Model (CIM)**[2]: a standard developed by the electric power industry, which facilitates information exchange about electrical networks.
- **Inspire**[3]: a standard that facilitates interoperability of spatial data sets and services. We evaluated only a subset of the Inspire UML model that is relevant for the energy sector.
- **SEAL**: an UML model based on CIM, which facilitates data exchange within Alliander.

4.1.2 Results

Using the Eclipse UML2 project, we iterated over the UML constructs in the test set. The results are presented in this section. Figure 4.1 shows the count of all UML metaconstructs used in the test set.

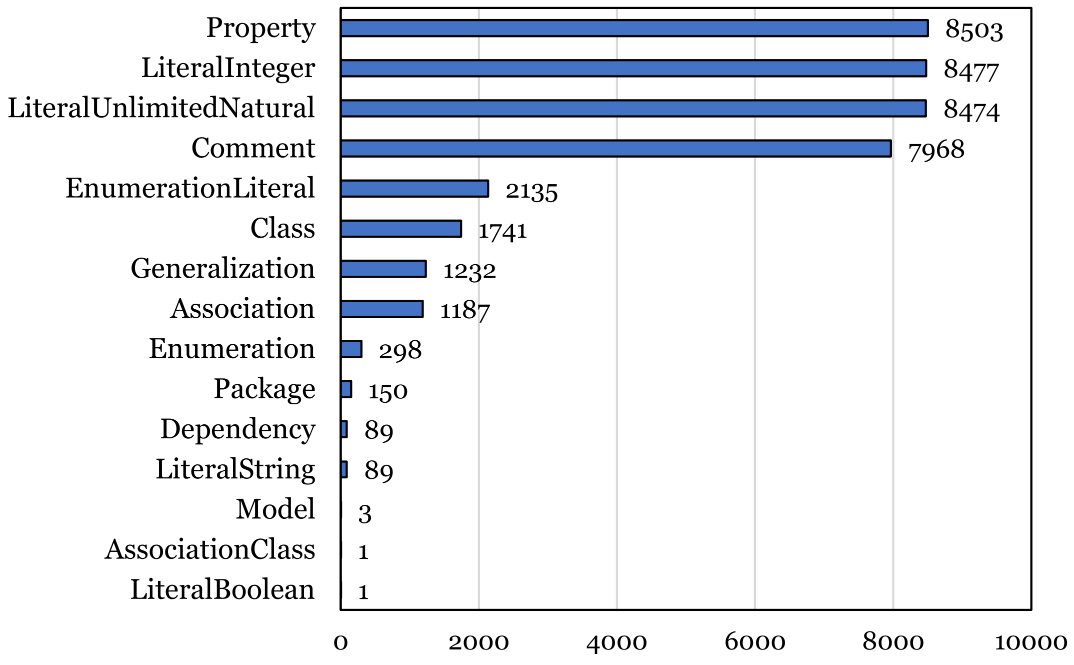


Figure 4.1: Count of UML constructs present in the test set

[1] Our test set is available at https://github.com/reycs/Transformations
[2] https://www.iec.ch/smartgrid/standards/
[3] https://inspire.ec.europa.eu/portfolio/data-models

As discussed in Section 3.8, `Properties` are used as `Class Attributes` and in `Associations`. Figure 4.2 breaks down which `Properties` are used as `Class Attributes` and which function as `Association memberEnds`. Section 3.8 also discussed that `Associations` can be `Aggregations` and `Compositions`. Figure 4.3 breaks down which `Associations` are `Aggregations`, `Compositions` and regular `Associations`. Note that our test set did not contain any `Compositions`.

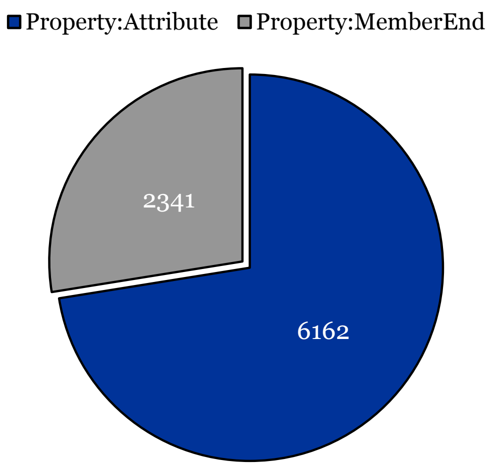


Figure 4.2: Breakdown of UML `Property`

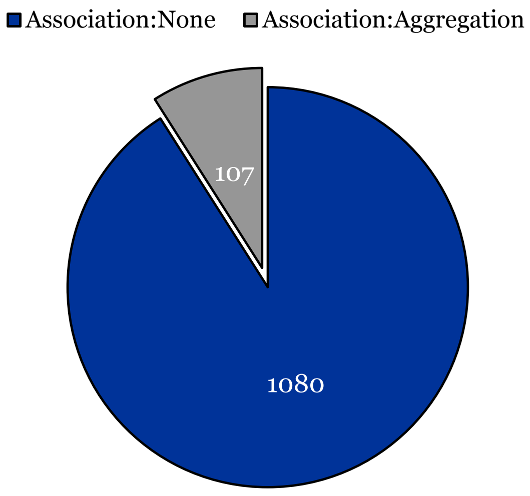


Figure 4.3: Breakdown of UML `Association`

In UML, it is possible to specify `Generalization` relationships between various constructs. Figure 4.4 presents the different usages of `Generalization` in the UML test set. Finally, it is possible to assign UML `Comments` to various constructs. Figure 4.5 presents the different constructs which have `Comments` in the UML test set.

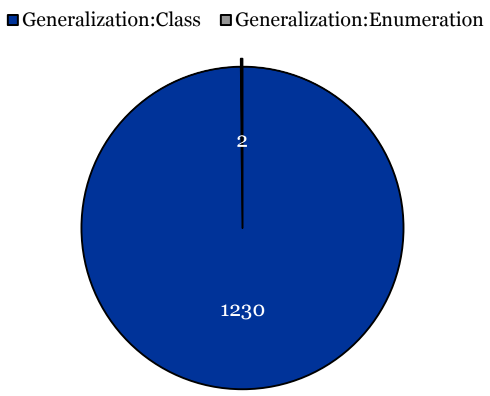


Figure 4.4: Breakdown of UML `Generalization`

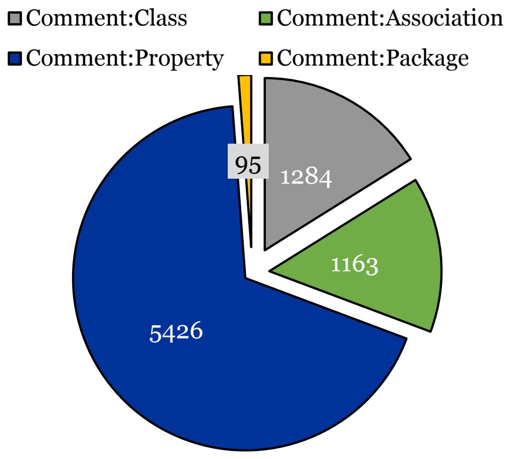


Figure 4.5: Breakdown of UML `Comment`

4.1.3 Discussion

Sadowska and Huzar identified the following most popular UML constructs: classes, attributes of classes, associations (including aggregation), cardinality of properties and generalization relationships [30]. These constructs correspond closely to the results of our UML evaluation see Figure 4.1 and 4.3. Note that cardinality of properties is represented using the `LiteralUnlimitedNatural` and `LiteralInteger` constructs. In addition to these constructs, we observed that packages, comments, enumerations and dependencies are often used as well.

4.2 Evaluating OWL

Several studies have examined the usage of OWL. Wang et al. [36] published a survey which examines the OWL landscape. However, this paper was published before the introduction of OWL2, making the results less relevant for our study. A more recent study by Matentzoglu et al. [20] examines a large number (4547) of OWL2 ontologies. Although their results are relevant, they do not present results for each individual OWL construct, but present more general categories, namely, entity usage, constructors and axiom types. We present an evaluation of a set of ontologies on both the axiom type level and at the level of each metamodel construct that can be instantiated.

4.2.1 Preparing a test set

We scraped `.ttl` `.rdf` and `.owl` files from w3.org and w3id.org, which resulted in a set of 242 ontologies. We removed all the ontologies that the OWL API was unable to parse, which left us with 216 ontologies. As Matentzoglu et al. observed, most ontologies that the OWL API was unable to parse were due to unavailable imports [20]. We added the well-known pizza ontology[4] to our set which raised our set to 217 ontologies. We removed all the ontologies without axioms, which left us with 176 ontologies. Finally, we only kept ontologies in the test set with an unique ontology IRI, to filter out duplicates in our test set, which left us with a final test set of 147 ontologies.

4.2.2 Results

Metamodel coverage

The OWL metamodel has 74 constructs that can be instantiated. We found out that added together the ontologies in the test set covered 66/74 (89%) of these constructs. We visualized the metamodel coverage of each ontology in the test set in Figure 4.6.

[4] https://protege.stanford.edu/ontologies/pizza/pizza.owl

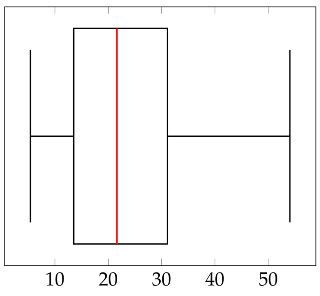


Figure 4.6: Boxplot of the metamodel coverage (in percentage) of the ontologies in the test set

Construct usage

We measured the used OWL constructs in the ontology test set, by iterating over the ontologies. The OWL metamodel construct count is presented in Figure 4.7 and 4.8. We also present a list of percentages of ontologies that use a construct for each instantiable OWL construct in Appendix C.

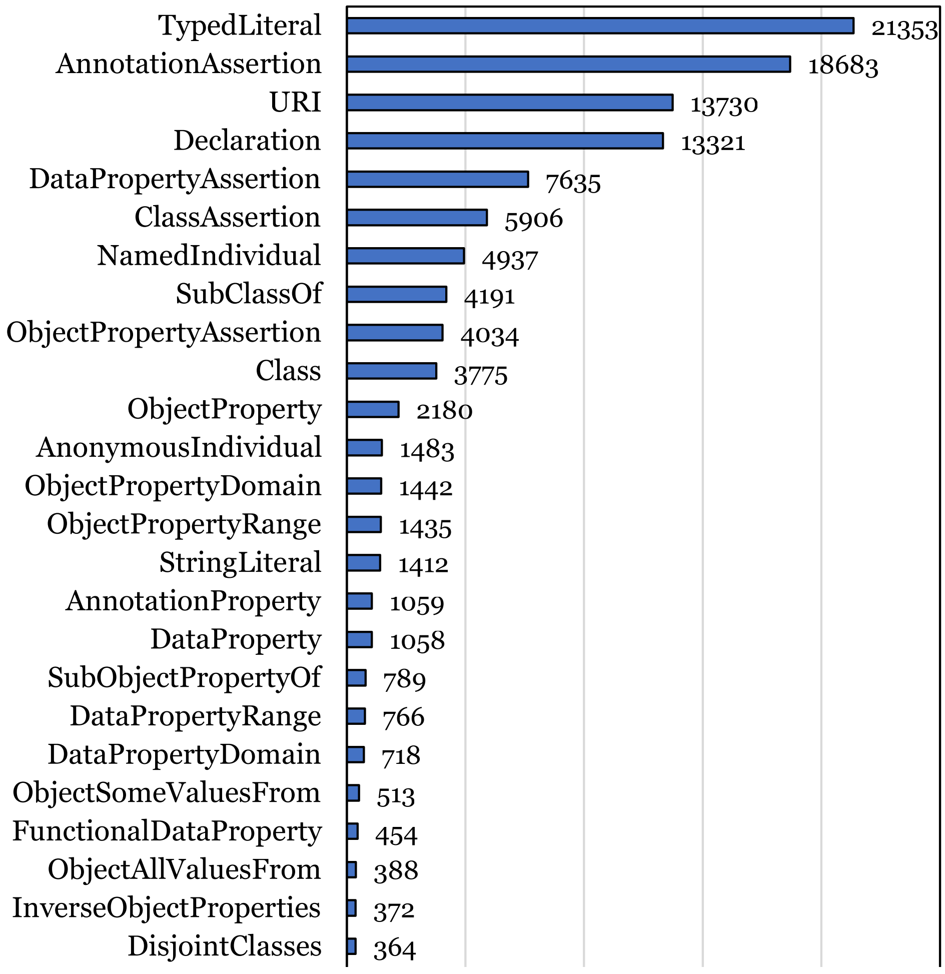


Figure 4.7: Count of OWL constructs present in the test set (continues next page)

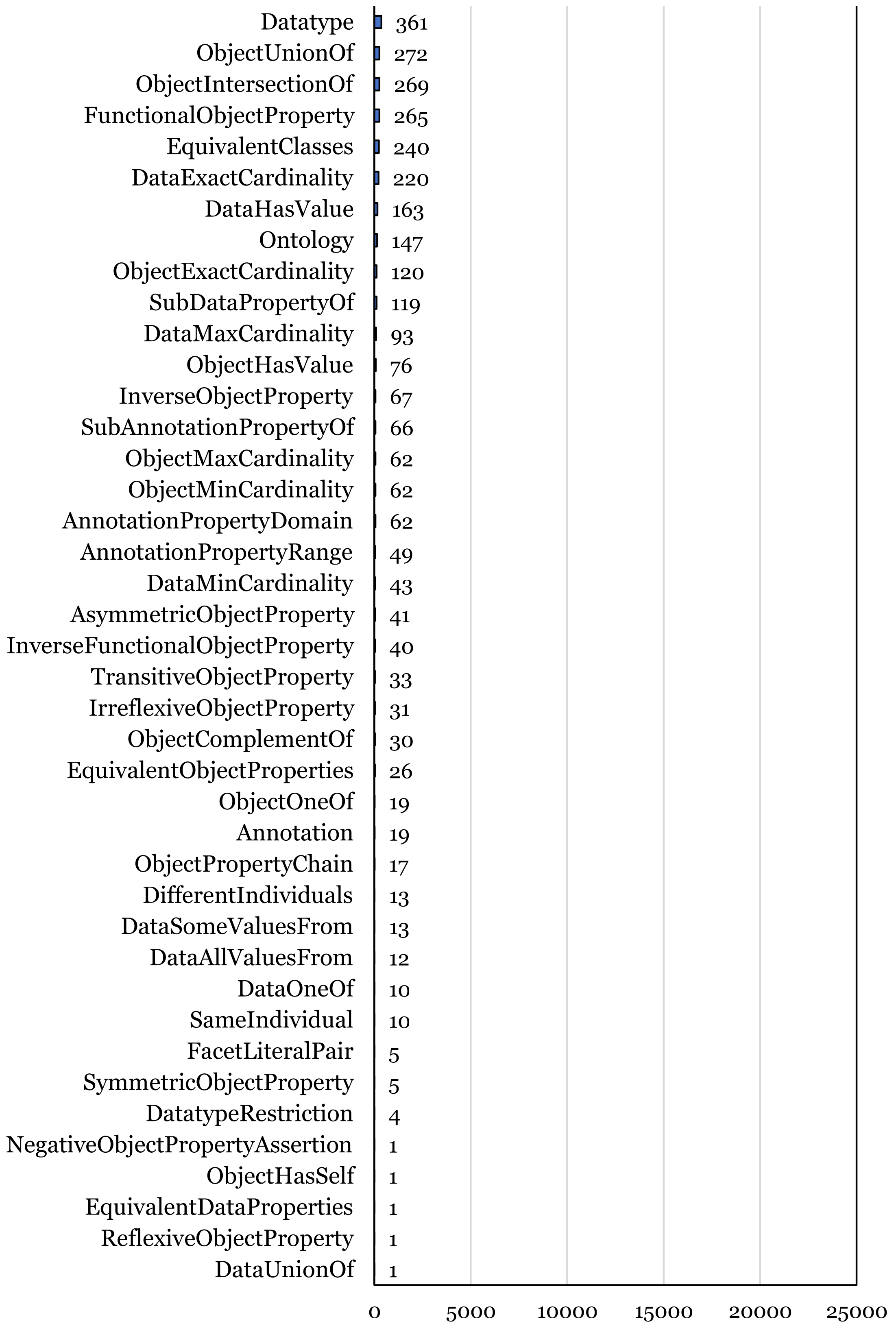


Figure 4.8: Count of OWL constructs (continued)

We observed that the following constructs did not appear in the test set at all: DisjointDataProperties, NegativeDataPropertyAssertion, DisjointObjectProperties, HasKey, DisjointUnion, DataIntersectionOf, DataComplementOf, DataTypeDefinition.

Axiom types

As mentioned in Section 3.1.1, ontologies are structured using `Axioms`. In Section 6.2 we discuss which information is lost in a roundtrip transformation from $OWL \rightarrow UML \rightarrow OWL$ using a list of lost constructs ordered by `Axiom Type`. This, because a comparison list of all OWL constructs is difficult to overview. For completeness, Figure 4.9 presents the count of `Axiom Types` in the test set.

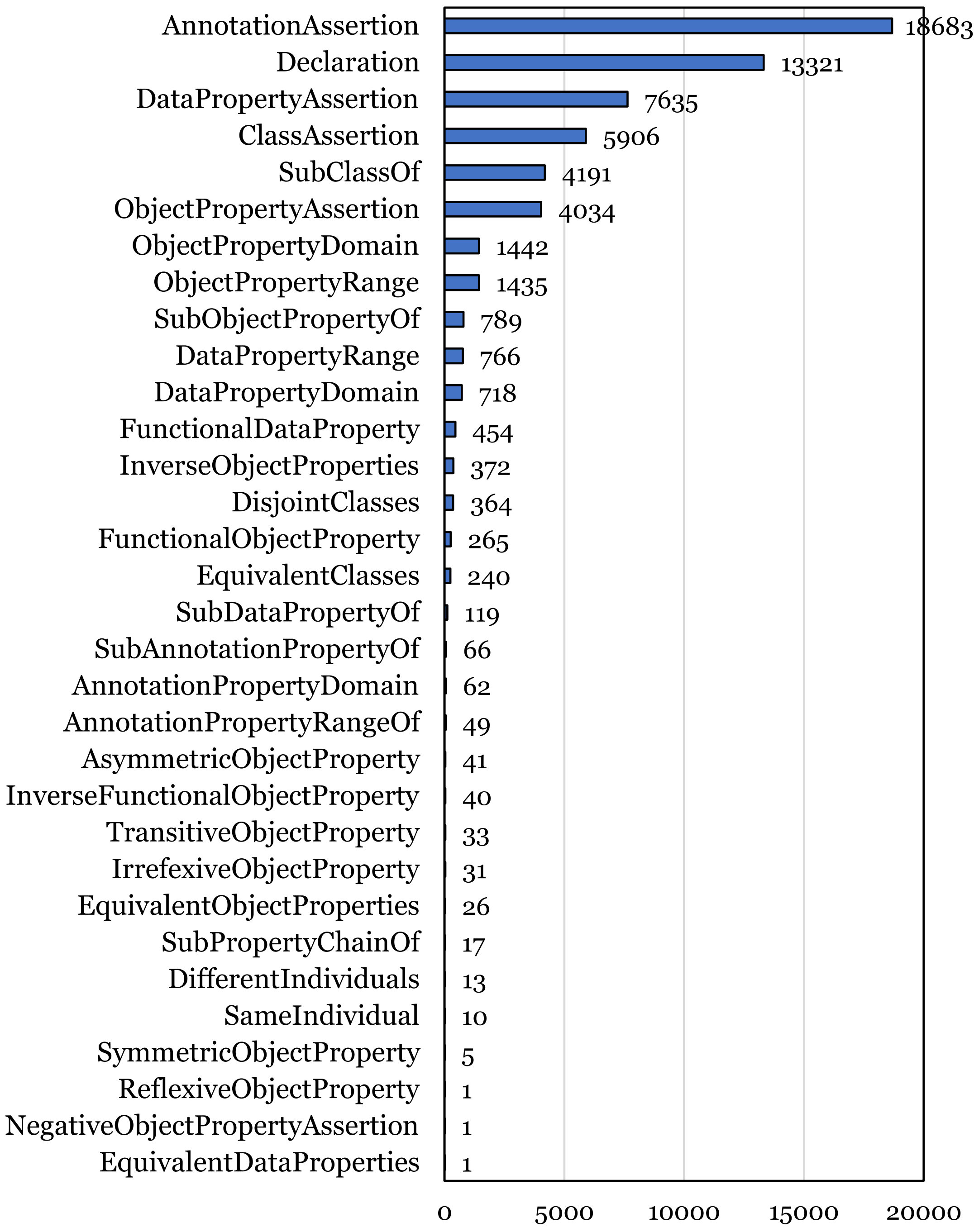


Figure 4.9: Count of OWL `Axiom Types`

4.2.3 Discussion

Our findings are similar to the results of Matentzoglu et al. [20]. Figure 4.7 and 4.8 show that `Classes, ObjectProperties and Individuals`, are used often. Figure 4.9 shows that simple axioms such as `ClassAssertion` and `SubClassOf` are often used as well. In addition, Figure 4.7 and 4.8 show that almost all OWL constructs are used at least ones in the test set. However, the majority of ontologies uses only a relatively small number of constructs, as Figure 4.6 illustrates.

4.3 Conclusion

To answer **SQ2**: What are the most commonly used constructs in UML and OWL? We presented the most commonly used UML and OWL constructs in Figure 4.1, 4.7, and 4.8. These figures are based on a UML test set consisting of three UML models that are used within Alliander and Enexis and an OWL test set of 147 OWL ontologies. Although the numbers suggest that there is a high discrepancy between evaluated UML models and OWL ontologies, the absolute count of constructs show that the difference is relatively small. A possible explanation could be that the UML test set contains 150 packages, which is comparable to 147 ontologies. We found that the UML models used within Alliander and Enexis use only a small subset of UML: the static elements of UML Class diagrams. Our findings align closely to the list of most popular UML constructs for conceptual modeling as suggested by Sadowska and Huzar [30]. We found that our OWL test set uses almost every instantiable OWL construct and our findings are similar to the results of Matentzoglu et al. [20]. However, despite the wide variety of constructs used in the test set, only a small subset of OWL constructs is commonly used, as Figure 4.6, 4.7 and 4.8 illustrate.

Chapter 5

A bidirectional transformation tool

This chapter discusses the implementation of our bidirectional transformation tool. Section 5.1 gives a high level overview of the architecture of our tool. Section 5.2 describes the implementation of the state-of-the-art transformation rules as found in Chapter 3. Section 5.3 discusses our changes to the OWL metamodel, to make it conform with the latest specification. Section 5.4 discusses our extension of the OWL API, which allows us to serialize and deserialize ontologies conforming with the OWL_{Ecore} metamodel. Section 5.5 presents the GUI of our transformation tool. Finally, Section 5.6 concludes this chapter by answering **SQ3**: How can we implement a metamodel based bidirectional transformation tool between UML and OWL?

5.1 Architecture

As discussed earlier in Section 3.3, we were unable to find a tool that can transform between UML and OWL bidirectionally. Within LinkED, the transformation from OWL to UML was not realized due to the complicated serialization process of ontologies. Our tool solves this serialization problem and realizes transformations between UML and OWL in both directions. Figure 5.1 illustrates the architecture of our tool. The tools consists of two main components: QVT transformations and the OWL (de)serialization.

QVT transformations The bidirectional transformations are written in the open source Eclipse QVT Operational implementation[1]. The transformations require an UML and an OWL metamodel. We used the UML2.5 metamodel from the

[1] https://projects.eclipse.org/projects/modeling.mmt.qvt-oml

Eclipse UML2 project[2], and the OWL2 model from the W3 Wiki[3]. We found out that the OWL2 metamodel from the Wiki did not conform with the latest OWL specification. We updated the metamodel, which is described in Section 5.3. Both the UML and the OWL metamodel are in Ecore format, the metamodel specification language of EMF.

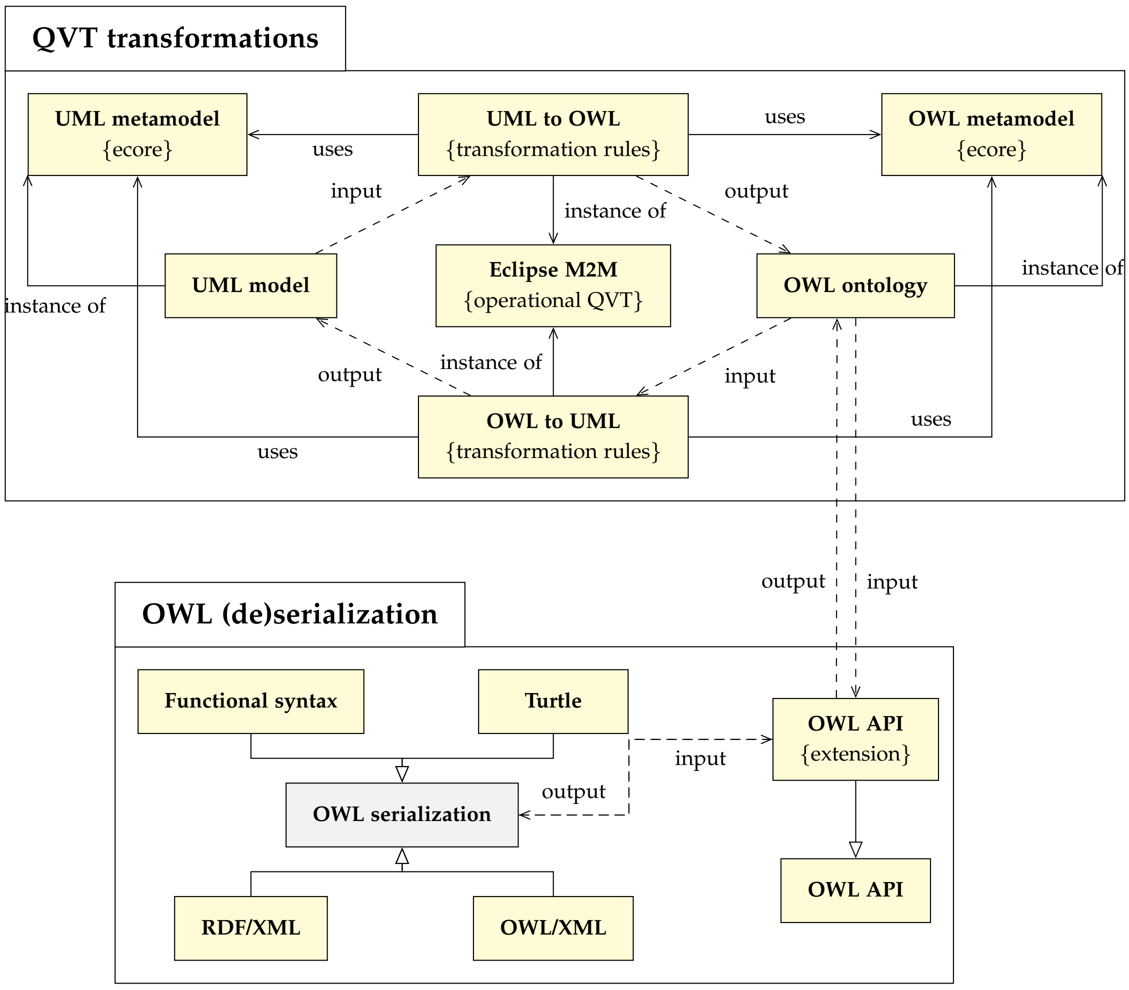


Figure 5.1: Transformation tool architecture

As mentioned in Chapter 4, different UML tools implement the XMI serialization standard differently. To prevent having to deal with UML tool interoperability issues, our tool only transforms UML models that conform with the Eclipse UML metamodel. Our UML to OWL transformation takes an UML model that conforms with the Eclipse UML metamodel as input and transforms it to an OWL ontology that conforms with the OWL metamodel. The OWL to UML transformation does the same in the opposite direction.

[2] https://www.eclipse.org/modeling/mdt/?project=uml2
[3] https://www.w3.org/2007/OWL/wiki/MOF-Based_Metamodel

OWL (de)serialization Although the QVT transformation component produces semantically valid OWL ontologies, ontologies that conform with the OWL2 Ecore metamodel are not serialized in a format that is used in practice. The OWL (de)serialization component solves this problem. This component extends the OWL API, a high-level API implemented in Java, which is closely aligned to the OWL2 structural specification [15]. The OWL API is able to parse and write various OWL serialization formats. We wrote an extension that is able to parse and write OWL ontologies conform the OWL Ecore metamodel to the OWL API, making it possible to use and serialize ontologies in popular serializations, such as RDF/XML and Turtle.

5.2 Implementing the transformation rules

We relied heavily on the research done in Chapter 3 and 4 for the implementation of the transformation rules. First, we evaluated whether the state-of-the-art mappings cover the used UML constructs that were used in the UML test set. Table 5.1 presents the UML constructs with corrosponding mappings as specified in Appendix A.1 which are implemented as QVT rules in our transformation tool.

Table 5.1: Implemented UML QVT rules in our transformation tool

UML Construct	**Mapping**
Package	Mapping32
Comment	
on Classes	Mapping33
Class	Mapping1
Property	Mapping4-5
Cardinality	Mapping23-27
Association	Mapping7-8
AssociationClass	Mapping9
Generalization	
between Classes	Mapping17
between Enumerations	Mapping19
between Properties	Mapping18
Enumeration	Mapping31

We did the same for OWL constructs. Table 5.2 presents the available state-of-the-art mappings as specified in Appendix A.2 for OWL constructs used in the test set. The mappings specified in Table 5.2 are implemented as QVT rules in our tool.

Table 5.2: Implemented OWL QVT rules in our transformation tool

OWL Construct	Mapping
Ontology	Mapping34
Class	Mapping35
SubClassOf	
with declared classes	Mapping36
with necessary conditions as SuperClass	Mapping38
SubObjectProperty	Mapping50
EquivalentClasses	
with declared classes	Mapping39
ObjectUnionOf	
with declared classes	Mapping41-42
ObjectIntersectionOf	
with declared classes	Mapping43
DataOneOf	Mapping44
DataProperty	Mapping45
DataPropertyRange	Mapping47-49
DataPropertyDomain	Mapping47-49
DataExactCardinality	Mapping55
DataMinCardinality	Mapping53
DataMaxCardinality	Mapping54
ObjectProperty	Mapping45-46
ObjectPropertyRange	Mapping47-49
ObjectPropertyDomain	Mapping47-49
ObjectExactCardinality	Mapping55
ObjectMinCardinality	Mapping53
ObjectMaxCardinality	Mapping54
InverseObjectProperties	Mapping57
FunctionalProperty	Mapping51
InverseFunctionalProperty	Mapping52
HasValue	
Object	Mapping56
Data	Mapping56

5.3 Updating the OWL metamodel

We thoroughly verified whether the OWL2 metamodel from the W3 Wiki[4] conforms with the latest structural specification of December 11, 2012[5] and found out, as mentioned earlier, that the metamodel does not conform with the latest specification. We found a number of constructs that did not conform with the specification, which can be categorized in: name changes, missing constructs and constructs modeled differently. We present an overview of our changes to the OWL2 metamodel from the W3 Wiki in this section, which makes the OWL metamodel conform with the latest structural specification.

Name changes

We changed the names of several constructs. Some of the names had to be changed because their name has been changed in the latest structural specification, others had to be renamed because a name different from any structural specification was given. Although ontologies can still be expressed semantically correct using these constructs, working with the constructs can be confusing. `KeyFor` was renamed to `HasKey`, `ObjectExistSelf` to `ObjectHasSelf` and `Constant` to `Literal`.

Missing constructs

Some constructs available in the latest structural specification were missing. We added `AnnotationAxiom`, `DataTypeDefinition` and `DataRange subclasses`.

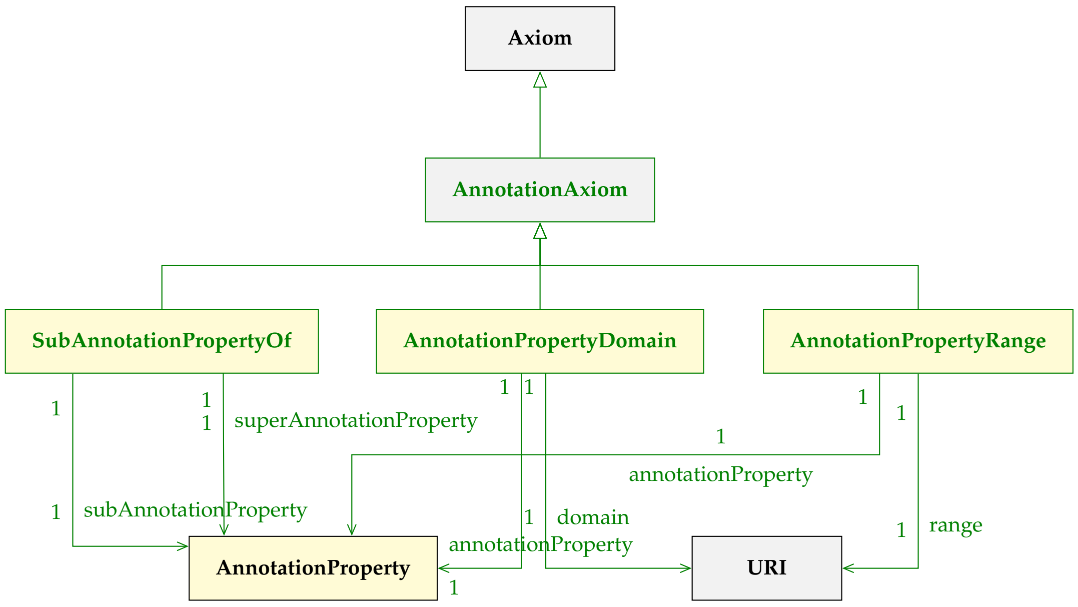


Figure 5.2: Added `AnnotationAxiom` part (1/2)

[4] https://www.w3.org/2007/OWL/wiki/MOF-Based_Metamodel
[5] https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

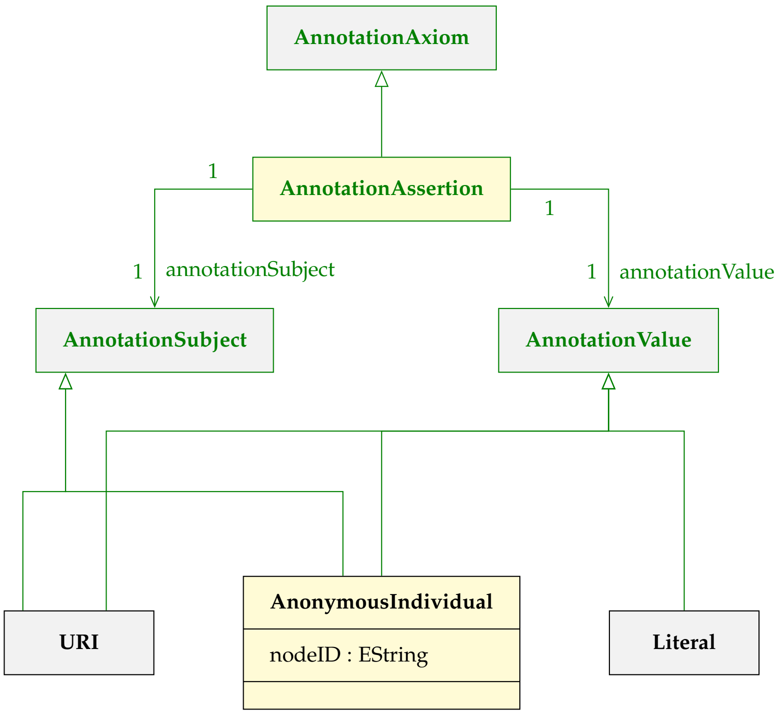


Figure 5.3: Added `AnnotationAxiom` part (2/2)

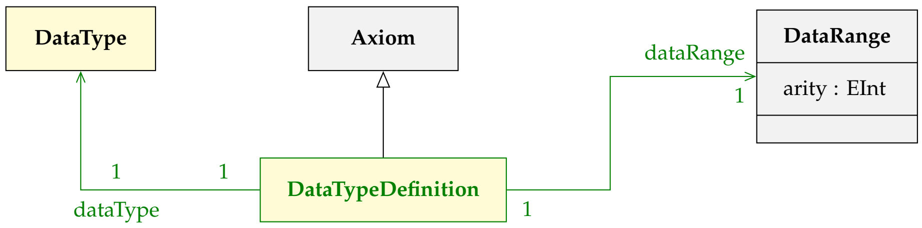


Figure 5.4: `DataTypeDefinition` has been added

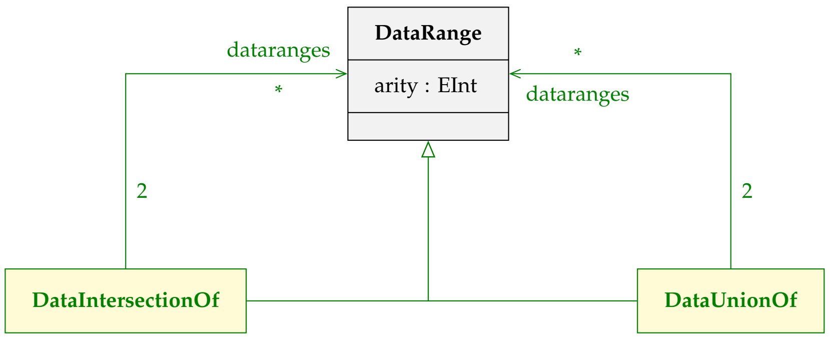


Figure 5.5: Added `DataIntersectionOf` and `DataUnionOf`

Constructs modelled differently

We found several constructs that were modeled differently. Some of them were modeled incorrectly, while others were modeled conform with older specifications. We updated the following constructs:

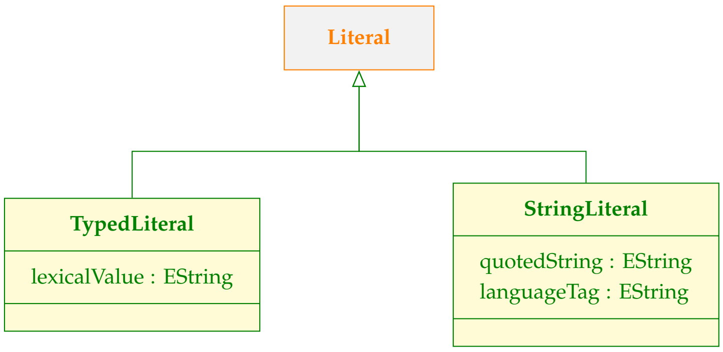


Figure 5.6: Changes to `Literal`

Figure 5.6 shows the changes to `Literal`. In the latest specification `Literal` can be either a `TypedLiteral` or a `StringLiteral`, whereas in older specification a single `Constant` construct was used. Figure 5.7 and 5.8 illustrate similar changes to `URI`. In the latest specification a `URI` can be a `FullURI` or an `AbbreviatedURI`.

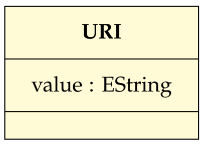


Figure 5.7: `URI` before changes.

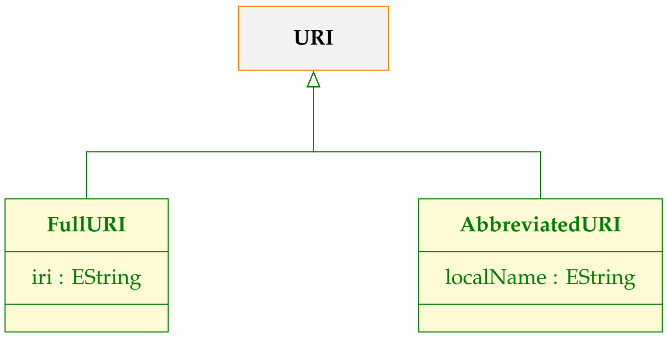


Figure 5.8: `URI` after changes.

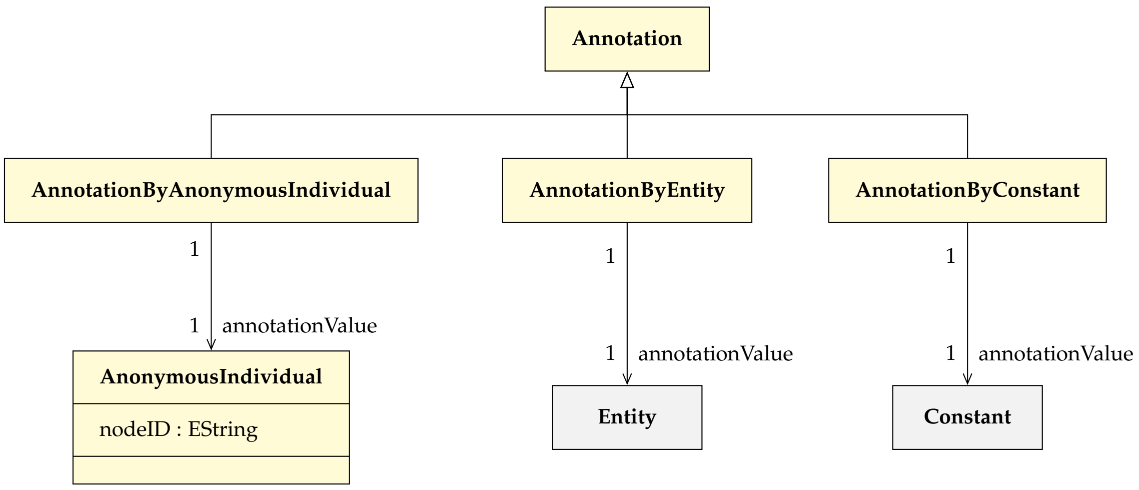


Figure 5.9: `Annotation` before changes

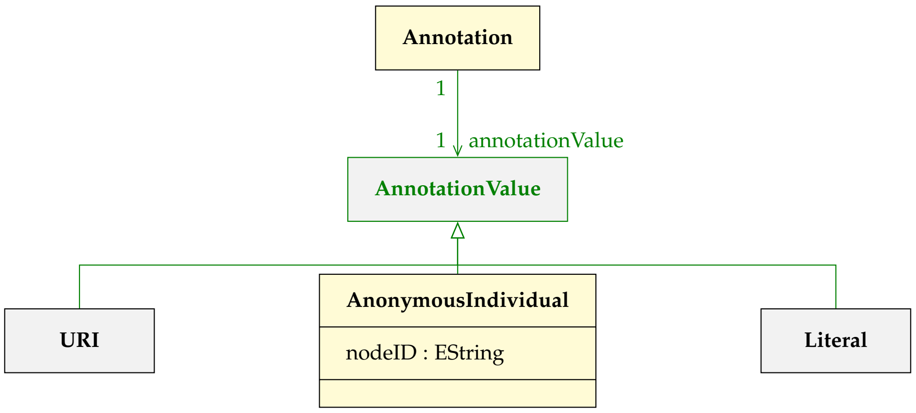


Figure 5.10: `Annotation` after changes

Figure 5.9 and 5.10 show the changes to `Annotation`. The latest specification simplified the `Annotation` construct by creating a super class that replaces the `AnnotationBy` classes for `AnonymousIndividual`, `Entity` and `Constant`. Due to the addition of the `AnnotationAxiom` the `AnonymousIndividualAnnotation` and `EntityAnnotation` were removed, this is shown in Figure 5.11.

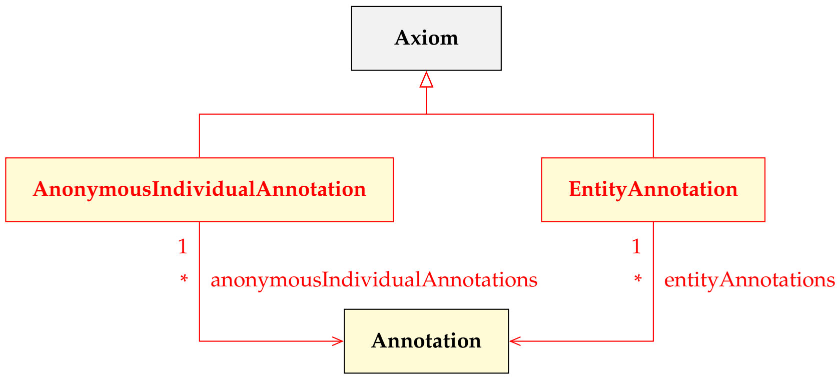


Figure 5.11: Removed `AnonymousIndividualAnnotation` and `EntityAnnotation`

Figure 5.12, 5.13, 5.14 and 5.15 show the changes to `ClassAssertion` and `SameIndvidual`. Both `ClassAssertion` and `SameIndividual` had the same modeling mistake. Instead of having an association with the super class `Individual`, the association was modeled to `NamedIndividual`.

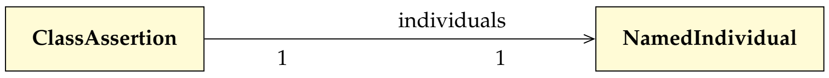


Figure 5.12: `ClassAssertion` before changes

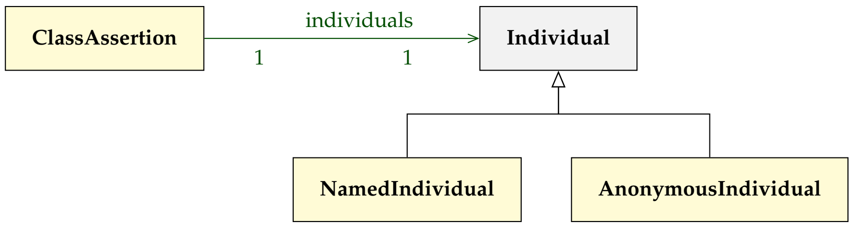


Figure 5.13: `ClassAssertion` after changes

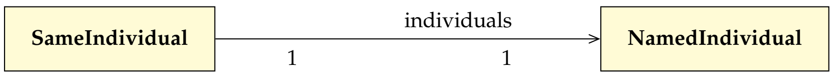


Figure 5.14: `SameIndividual` before changes

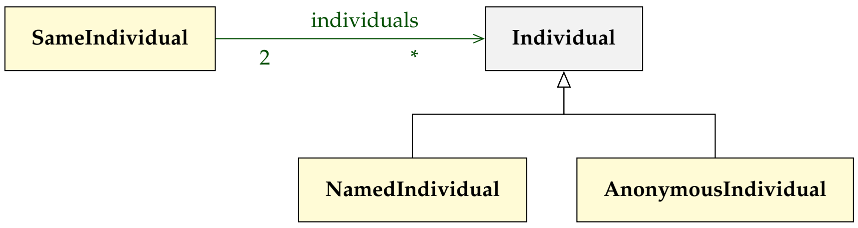


Figure 5.15: `SameIndividual` after changes

Finally, `FacetLiteralPair` had a minor mistake. `FacetLiteralPair` had a property with type EString, whereas in the latest specification, an association with `Literal` was modeled instead. Figure 5.16 and 5.17 illustrate this change.

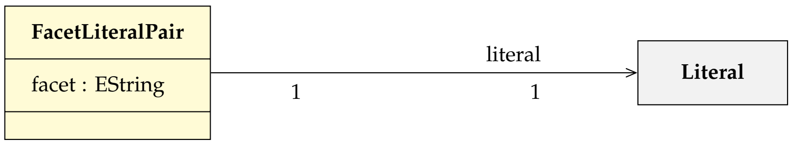


Figure 5.16: `FacetLiteralPair` before changes

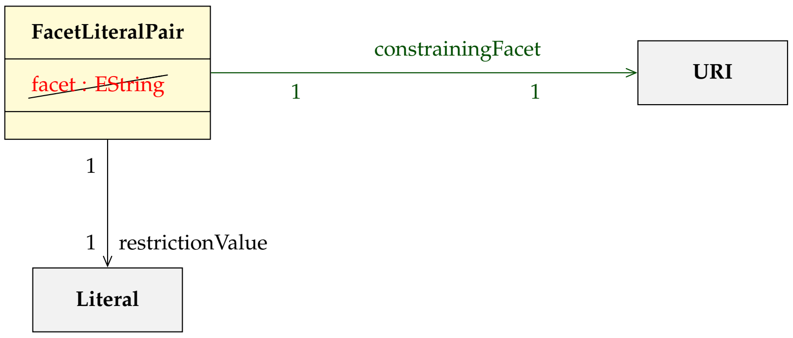


Figure 5.17: `FacetLiteralPair` after changes

5.4 Extending the OWL API

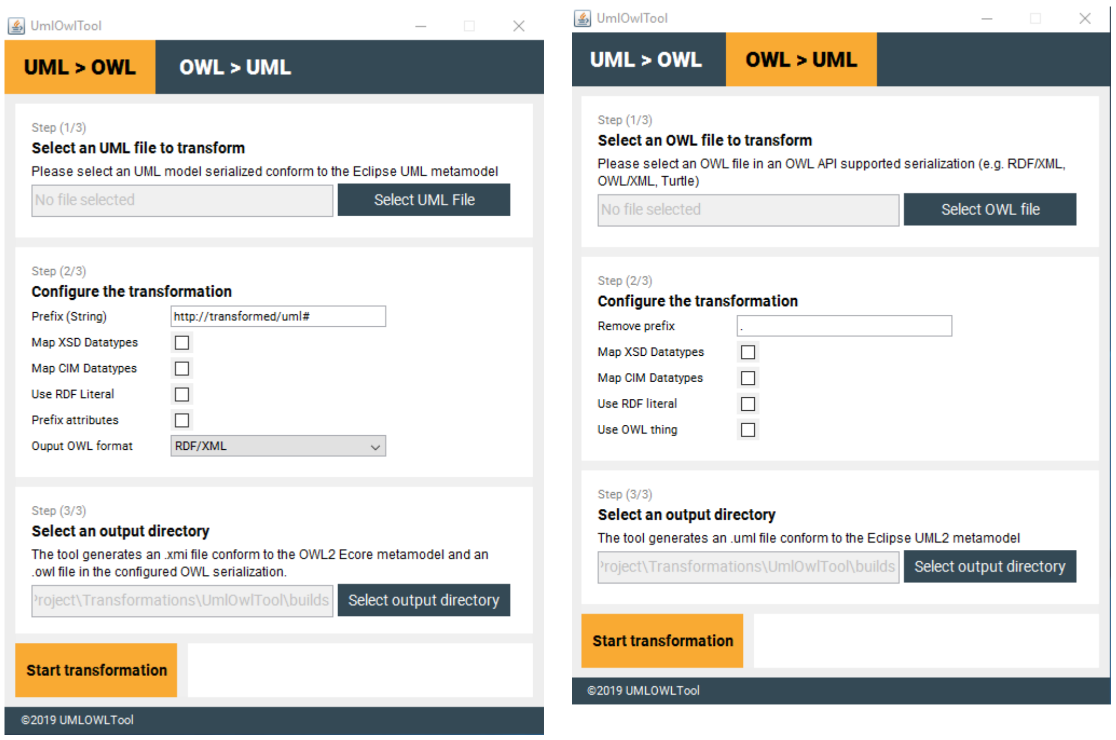
Our OWL (de)serialization component extends version 5.1.10 of the OWL API[6] with a parser and writer for ontologies conform the OWL metamodel. Using EMF, we generated a Java representation of the corrected OWL metamodel. In addition to the representation, EMF also generates a `Switch` class that allows us to iterate over all the constructs of an ontology and handle each construct in a separate case function. We mapped each construct in the OWL metamodel to the corresponding construct in the OWL API. This allowed us to parse ontologies that conform with the OWL metamodel to the OWL API.

[6] https://github.com/owlcs/owlapi

Ontologies parsed by the OWL API are written to ontologies that conform with the OWL metamodel in a similar manner. The OWL API uses the visitor design pattern[7] for most of its writers and parsers. We extended the base `OWLObjectVisitor` class of the OWL API, which allowed us to iterate over the OWL API objects and map it to the appropriate OWL metamodel constructs. This parser and writer make it possible to serialize ontologies resulting from our UML to OWL transformation in every available OWL API serialization and makes it possible to use ontologies serialized in OWL API parseable serializations in our OWL to UML transformation.

5.5 Transformation tool GUI

One obstacle with existing transformation tools is that they can be difficult to use due to platform dependencies. The LinkED transformation method for instance, requires the Eclipse environment with ATL dependencies installed. We wanted to create a tool that is as easy to use as possible. We used the QVTo `TransformationExecutor`[8] class to launch QVTo programmatically using java. In addition, we build a GUI on top of our tool using Java Swing[9]. All the required dependencies are packaged in a Jar that requires only Java 8 to be run. Figure 5.18 shows the GUI of our transformation tool.



Step (1/3)
Select an UML file to transform
Please select an UML model serialized conform to the Eclipse UML metamodel
No file selected | Select UML File

Step (2/3)
Configure the transformation

Prefix (String)	http://transformed/uml#
Map XSD Datatypes	☐
Map CIM Datatypes	☐
Use RDF Literal	☐
Prefix attributes	☐
Ouput OWL format	RDF/XML

Step (3/3)
Select an output directory
The tool generates an .xmi file conform to the OWL2 Ecore metamodel and an .owl file in the configured OWL serialization.
'roject\Transformations\UmlOwlTool\builds | Select output directory

Start transformation

©2019 UMLOWLTool

Step (1/3)
Select an OWL file to transform
Please select an OWL file in an OWL API supported serialization (e.g. RDF/XML, OWL/XML, Turtle)
No file selected | Select OWL file

Step (2/3)
Configure the transformation

Remove prefix	.
Map XSD Datatypes	☐
Map CIM Datatypes	☐
Use RDF literal	☐
Use OWL thing	☐

Step (3/3)
Select an output directory
The tool generates an .uml file conform to the Eclipse UML2 metamodel
'roject\Transformations\UmlOwlTool\builds | Select output directory

Start transformation

©2019 UMLOWLTool

Figure 5.18: GUI of our transformation tool

[7] https://en.wikipedia.org/wiki/Visitor_pattern
[8] https://wiki.eclipse.org/QVTOML/Examples/InvokeInJava
[9] https://docs.oracle.com/javase/tutorial/uiswing/

5.6 Conclusion

To answer **SQ3**: How can we implement a metamodel based bidirectional transformation tool between UML and OWL? We have implemented a bidirectional transformation tool between UML and OWL using QVTo and the OWL API, based on the state-of-the-art mappings and the commonly used UML and OWL constructs in practice. We extended the OWL API to make it possible to use and serialize ontologies in commonly used serialization formats, such as RDF/XML and Turtle. This extension made it possible to bidirectionally transform between UML models and OWL ontologies. We found out that for most of the used UML constructs in our test set, mappings to OWL can be found in the literature. Finally, we observed that the state-of-the-art mappings do not cover all the used OWL constructs, which could indicate that UML lacks features to express OWL concepts. We investigate this in detail in Chapter 6.

Chapter 6

Roundtrip transformations

This chapter evaluates the implemented state-of-the-art mappings, by performing roundtrip transformations on the UML and OWL test sets introduced in Chapter 4. Section 6.1 presents the UML results and discussion. Section 6.2 discusses the OWL results. Finally, Section 6.3 concludes this chapter by answering **SQ4**: What information is lost in a roundtrip transformation from UML to OWL and OWL to UML based on the state-of-the-art mappings?

ODM [27] identified several common problems in metamodel transformations.We use some of their terminology throughout this chapter:

- **Structure conflation**: two constructs in the source metamodel map to a single construct in the target metamodel.
- **Loss of structure**: a complex construct is mapped to a collection of simpler constructs. However, there is insufficient information to map the simple constructs back to the complex constructs.
- **Trapdoor mappings**: a construct in the source metamodel is mapped to a very specific arrangement of constructs in the target metamodel.
- **Lack of features**: the target metamodel lacks a construct that is available in the source metamodel.

6.1 UML roundtrip

Figure 6.1 illustrates our UML roundtrip transformation method. We used the UML test set consisting of INSPIRE, SEAL and CIM, serialized in Eclipse$_{xmi}$, as introduced in Chapter 4. First, we transformed the UML test set into OWL using our transformation tool. Second, we transformed the OWL test set back into UML. Finally, we compared the original UML test set and the roundtrip test set. This comparison consists of counting the relevant UML constructs in the original and the roundtrip UML test set. In addition, we evaluated whether the constructs model the same instance, by comparing the instance names.

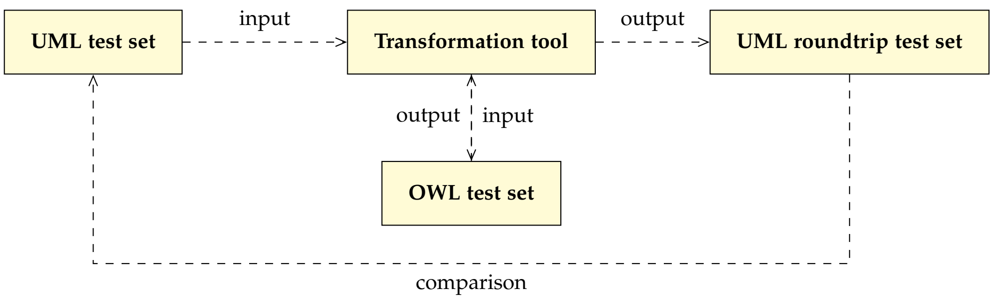


Figure 6.1: Illustration of UML roundtrip data gathering process

6.1.1 Results

Table 6.1 presents the count of UML constructs before and after the roundtrip transformation for each of the UML models in the test set. Some constructs are broken down into properties of the construct. For instance, whether a `Class` was set to `Abstract`. We visualized the accumulative counts of the constructs before and after the roundtrip in Figure 6.2.

UML Construct	INSPIRE	INSPIRE roundtrip	SEAL	SEAL roundtrip	CIM	CIM roundtrip
Package	2	1	16	1	132	1
Class	91	92	113	113	1527	1374
isAbstract	27	0	-	-	-	-
Association[1]	24	25	15	15	1148	1148
Aggregation	13	0	11	0	94	0
Unidirectional	15	0	11	0	0	0
Bidirectional	4	0	-	-	-	-
Not specified	5	25	4	15	1148	1148
AssociationClass	1	0	-	-	-	-
Generalization	49	49	120	120	1063	1063
Property[2]	246	158	122	122	8135	8129
ownedAttribute	221	108	103	92	5839	5833
Comment	-	-	23	0	7945	0
Enumeration	2	2	5	4	291	291
Dependency	-	-	-	-	89	0
DataType	-	-	0	1	-	-

Table 6.1: Used UML constructs before and after the roundtrip transformation

[1]Association names are lost in the roundtrip transformation

[2]The names of properties that are used in associations are different

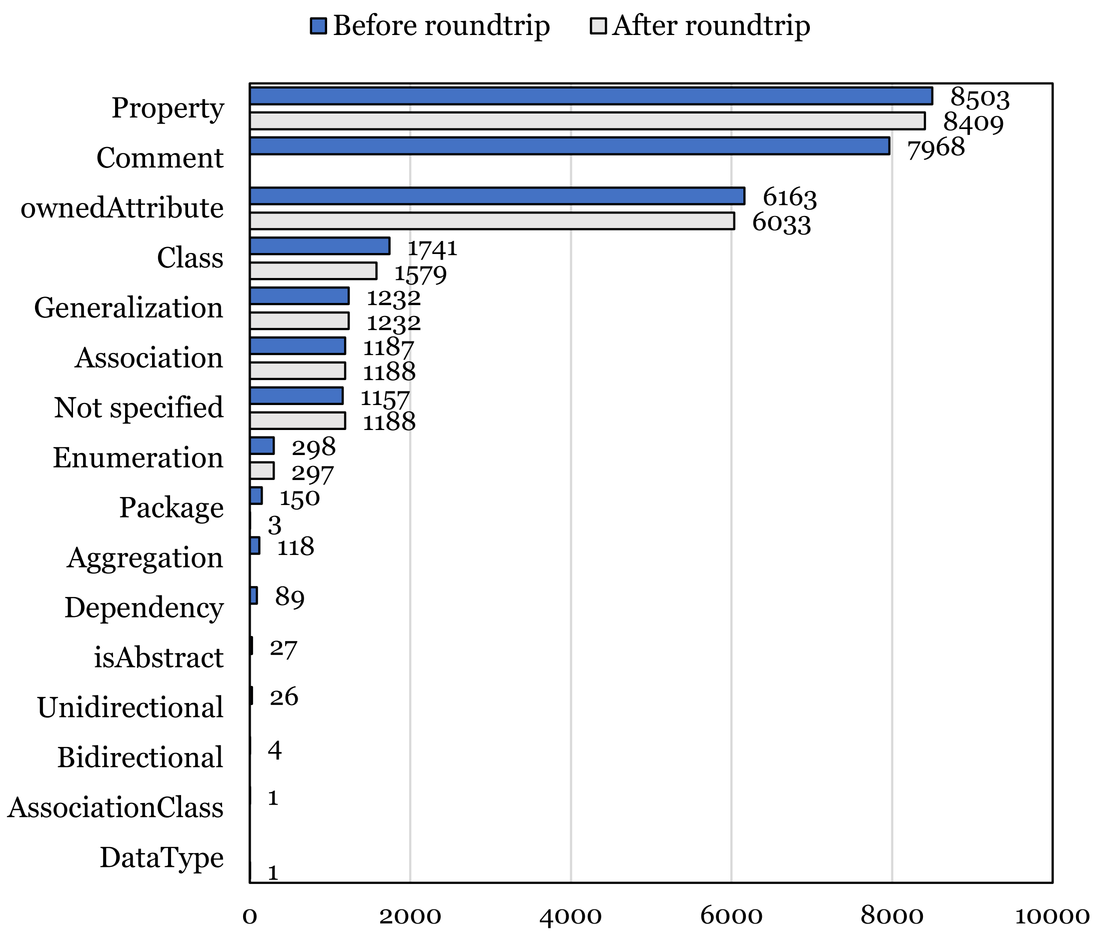


Figure 6.2: Total constructs before and after the roundtrip

6.1.2 Discussion

Table 6.1 and Figure 6.2 show that many constructs are preserved in the roundtrip transformation. However, there are also various constructs for which the numbers differ, which could indicate that information is lost. We evaluate which constructs are lost, and why these constructs are lost.

Packages

The state-of-the-art mappings briefly discuss `Packages` and package imports. However, they do not go into detail on how to resolve nested packages. In the current implementation of our transformation tool, all the elements in packages, including nested packages are mapped to a single `Ontology`. Consequently, the package structure is lost after the roundtrip transformation, as only a single package remains. In addition, all `Dependency` constructs are lost during the roundtrip, because each of these dependencies specifies a dependency between packages.

Alternatively, one could map each `Package` to a new `Ontology` with appropriate imports. By doing this, the package structure can be preserved. Furthermore, the dependencies on packages can then be preserved as well. ODM [27] suggests mapping dependencies to an `OWL:Annotation` with `RDF:Property` as `OWL:AnnotationProperty`. We suggest a different mapping. Instead of mapping to an `RDF:Property`, we suggest using `http://purl.org/dc/terms/requires` as `AnnotationProperty`, which is defined as "A related resource that is required by the described resource to support its function, delivery, or coherence"[3]. This definition is close to the meaning of a `Dependency`.

Classes

The number of classes in CIM and INSPIRE before the roundtrip is different from those after the roundtrip. There are two explanations for this difference in number of classes:

1. CIM contains 153 classes without names. These nameless classes are not mapped, because each `OWL:Class` requires a unique `IRI` which is based on the name of the `UML:Class`. This results in fewer classes after the roundtrip.
2. INSPIRE contains an extra `Class` after the roundtrip transformation. This, due to the `UML:AssociationClass` that is mapped to a `UML:Class` and an `UML:Association` in the OWL $\rightarrow$ UML step of the roundtrip transformation.

Abstract classes

Before the roundtrip transformation, INSPIRE contained 27 classes that were marked as abstract. After the roundtrip, these classes were no longer marked as abstract due to lack of features. OWL has no equivalent construct to mark classes as abstract. Guizzardi et al. [12] discuss the meaning of abstract classes. They discuss that the UML specification states that abstract classes are not instantiable and are commonly used to reify constructs, share structure and organize a model. They argue that abstract classes are conceptual constructions in the form of a hierarchy whose bottom elements denote universals. In other words, abstract classes are used in a hierarchy to specify a more general conceptualization of more concrete classes. Figure 6.3 shows an example of the usage of an abstract class in INSPIRE.

[3] http://purl.org/dc/terms/requires

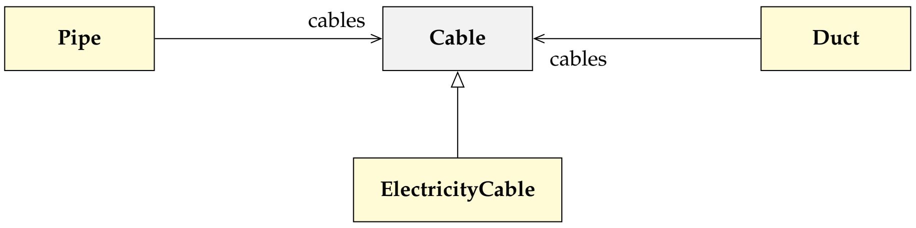


Figure 6.3: Subset of INSPIRE that shows an abstract class

It shows that a pipe and duct can have cables, and one particular type of cable is an electricity cable. The abstract class cable is a more general conceptualization of specific cables. After the roundtrip, the cable class is no longer marked abstract. However, despite no longer being marked abstract, the cable class is still a conceptual construction in the form of hierarchy with more specific classes. We claim that the loss of abstract class markings does not imply a loss of information about the conceptual domain, but rather has practical implications for the instances of the model. For instance, the reason why the abstract class cable was added, could be to make relations between pipes, ducts and cables easier to maintain. Instead of specifying associations between every type of cable and ducts and pipes, only associations with the conceptual construction cable have to be made. However, this is still possible without making the cable class abstract. The loss of abstract class has consequences for the instance level. Imagine that this slice of the INSPIRE model is used to create a database. If cable is an abstract class, we can only instantiate or populate the database with concrete cables. Whereas if cable is not an abstract class, we can find cables in the database, which could be undesirable. Thus, by losing the abstract class property, no conceptual information is lost. Instead, a constraint on the instances, or population of the model, is lost.

Associations and attributes

Table 6.1 shows that associations can have unidirectional, bidirectional or unspecified navigability. After the roundtrip, all associations have unspecified navigability. As discussed in Chapter 3, ODM is the only state-of-the-art mapping that briefly considers the difference between uni- and bidirectional navigability of associations, but this discussion is not taken into account in their mapping of associations. Consequently, our transformation tool does not preserve the navigability of associations in a roundtrip transformation. There are two ways to specify the navigability of an association:

1. An associationEnd property that is owned by a class (ownedAttributes) is navigable.

2. An associationEnd property that is in the navigableEnd set is navigable (See Figure 3.8 for more details).

If both associationEnd properties are navigable, the association is bidirectional, if one is navigable, unidirectional and otherwise unspecified. This explains the difference in properties that are an owned attribute from before and after the roundtrip transformation. In our test set, the navigability of associations is specified by ownership of properties by classes. SEAL has 11 unidirectional associations, which results in 11 more properties that are owned by a class in the original test set than in the roundtrip test set. The differences in properties and owned attributes in INSPIRE is partly due to the navigability of associations but has an additional reason. INSPIRE contains properties without a type. These properties are not transformed because the transformation requires a type to determine whether to map to an `OWL:DataProperty` or `OWL:ObjectProperty`.

We suggest a mapping for navigability of associations like ODM suggested, but with additional details. An unidirectional navigable association is mapped to an `OWL:ObjectProperty` with the non-navigable end as domain. Both associations whose navigability is unspecified and bidirectional are mapped to `OWL:Objectproperties` with an `OWL:InverseObjectProperties` axiom. In the transformation from OWL $\rightarrow$ UML we suggest setting the navigability of mapped associations to bidirectional by default, instead of unspecified.

The implementation of this mapping resolves some of the previously mentioned roundtrip issues but introduces new ones. All associations after the roundtrip that had unspecified navigability will have bidirectional navigability. However, when navigability is not specified in our UML test set, bidirectional navigability is meant in most cases. Therefore we see the transformation from unspecified navigability to bidirectional as desirable. The new mappings also introduce structure conflation. Since both uni-directional associations and class owned attributes are mapped to OWL:Properties, it is unclear whether an OWL:Property should be transformed into an association or a class owned attribute in the OWL $\rightarrow$ UML transformation. In the current implementation, both unidirectional associations and class owned attributes will become class owned attributes after the roundtrip.

The question then becomes whether information is lost when unidirectional associations are transformed into class owned attributes. Colomb et al. [6], have evaluated the distinction between unidirectional associations and class attributes. They argue that they are semantically equivalent, hence no information about the conceptual domain is lost when choosing between unidirectional associations and class attributes. They give two practical explanations why the distinction between unidirectional associations and class attributes can be useful:

1. If the UML model is used to implement a relational database, attributes are usually implemented as columns in a table, whereas associations are implemented as multiple tables with foreign keys. Multiple tables with foreign keys are heavier to process, so in this case, the choice between attributes and associations is an engineering decision.

2. The distinction between associations and attributes can help with the clarity of the model. Associations can represent the most important connections and attributes the less important, which can then be hidden in the visual overview of the model.

Finally, a best practice in industry is to represent `UML:Properties` that have a `Class` as `Type` as an `Association` and `UML:Properties` that have a `DataType` as `Type` as `Class Attributes`[4]. We suggest following this best practice instead of the current mapping proposed in the state-of-the-art mappings. So, instead of mapping all `OWL:Properties` except for those that have `OWL:InverseProperties` to attributes, we suggest mapping `OWL:DataProperties` to class attributes and `OWL:ObjectProperties` to associations.

Aggregation

Associations can represent an aggregation relation. These aggregation relations are lost in the roundtrip transformation due to a lack of a feature. Despite this lack, the state-of-the art mappings suggest that the restrictions that an aggregation relation imposes can be transformed. Zedlitz states that aggregations are antisymmetric and an object must not be in an aggregation relation to itself [38]. He suggests adding an AsymmetricObjectProperty axiom and an irreflexiveObjectProperty axiom to satisfy these constraints.

However, although the constraints that an aggregation relation imposes can be transformed, the semantics of the aggregation relation are lost in a roundtrip transformation. The semantics of an aggregation relation are commonly understood as an association that specifies that a type of object *consists of* other types of objects [24], or as others have put it a *component of* relation [11]. Figure 6.4 gives an example of an aggregation relation in CIM.

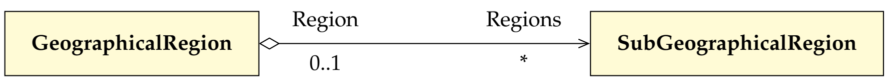


Figure 6.4: Example of an aggregation association in CIM

Figure 6.4 represents that a GeographicalRegion consists of SubGeographicalRegions, or in other words, that SubGeogrpahicalRegions are components

[4] https://bellekens.com/2011/08/10/uml-best-practice-attribute-or-association/

of a GeographicalRegion. This information is lost because after the roundtrip aggregation relations become regular associations.

Association class

INSPIRE has one association class that is lost in the roundtrip due to loss of structure. After the roundtrip, this association class is transformed into a regular class and association, as illustrated in Figure 6.5. The association class adds an extra constraint on the subunit association between UnitOfMeasure. For each UnitOfMeasure that is a subunit of another UnitOfMeasure, there may only exist one SubUnitsperUnit class that adds information to this association, whereas in the model after the roundtrip, the SubUnitsPerUnit can be associated with as many UnitOfMeasure classes as the cardinalities permit.

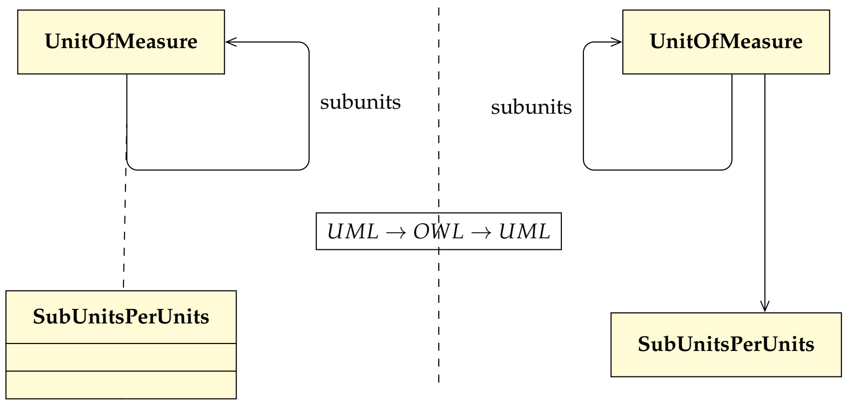


Figure 6.5: The association class in INSPIRE before and after the roundtrip

A clearer example is presented on etutorial.org[5]. They present an association class between a person and a skill, which describes the competence of someone in that skill. For each association from a person and a skill, a person can only have one competence. If this was modeled using a regular class instead of an association class a person could have multiple competences for a given skill. So, the association class adds an extra constraint on the instances or population of the model. This constraint is lost in the roundtrip transformation.

Enumerations, datatypes and comments

In SEAL one enumeration is missing after the roundtrip, which has become a datatype instead. The enumeration became a datatype because the enumeration was declared without any literals (or values). Our mapping from OWL → UML only transforms DataOneOf axioms into enumerations, but because

[5]http://etutorials.org/Programming/UML/Chapter+6.+Class+Diagrams+Advanced+Concepts/Association+Class/

the enumeration has no literals, only a datatype definition was added without datatypedefinition that specifies the dataoneof axiom.

Finally, the comments are missing after the roundtrip. Although a mapping from UML:Comments to OWL:Annotations has been implemented, no mapping from OWL:Annotations to UML:Comments has been suggested in the state-of-the-art mappings. We discuss this in more detail in Section 6.2.2.

6.1.3 Metamodel overview

Figure 6.6 gives a compact overview of our UML roundtrip evaluation using a simplified version of the UML metamodel. It presents which constructs are preserved and lost after the roundtrip transformation.

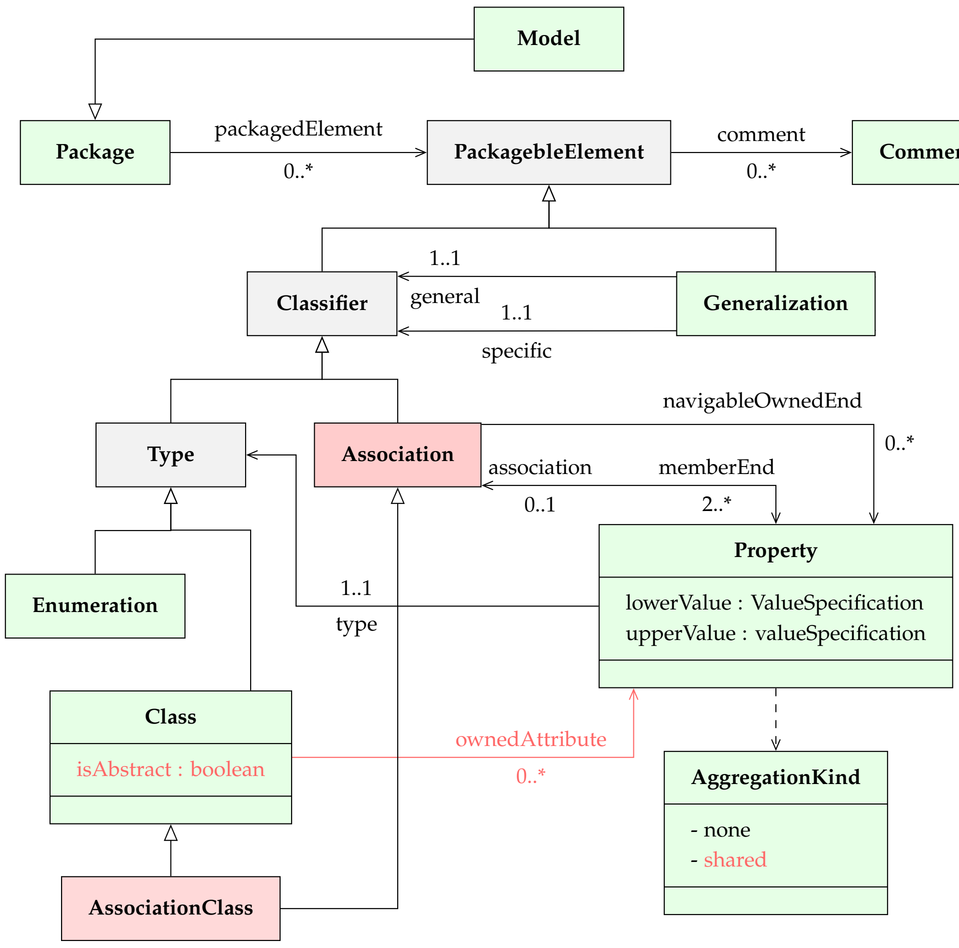


Figure 6.6: Overview of preserved and lost constructs after the roundtrip transformation

Constructs colored green are preserved after the roundtrip transformation. Constructs and associations colored red have missing information. As discussed in Section 6.1.2, the problem of choosing between associations and class owned attributes can be resolved by creating associations for relations between classes and class owned attributes for relations with types. We lost constraints on the population of the models by losing the association class and the abstract property of classes. Finally, we lost the semantics of the aggregation relation.

6.2 OWL roundtrip

Figure 6.7 illustrates our OWL roundtrip transformation method. We used the OWL test set introduced in Chapter 4. First, the OWL test set was transformed into OWL serialized as Eclipse$_{xmi}$ using our OWL API extension. Second, the OWL test set was transformed into UML using our transformation tool. Third, the UML test set was transformed back into OWL serialized as Eclipse$_{xmi}$. Fourth, the OWL was serialized as RDF/XML using our OWL API extension. Finally, we compared the original OWL test set with the roundtrip test set using the equal comparison function of the OWL API. This function compares two ontologies and returns the axioms that are missing and those that are added.

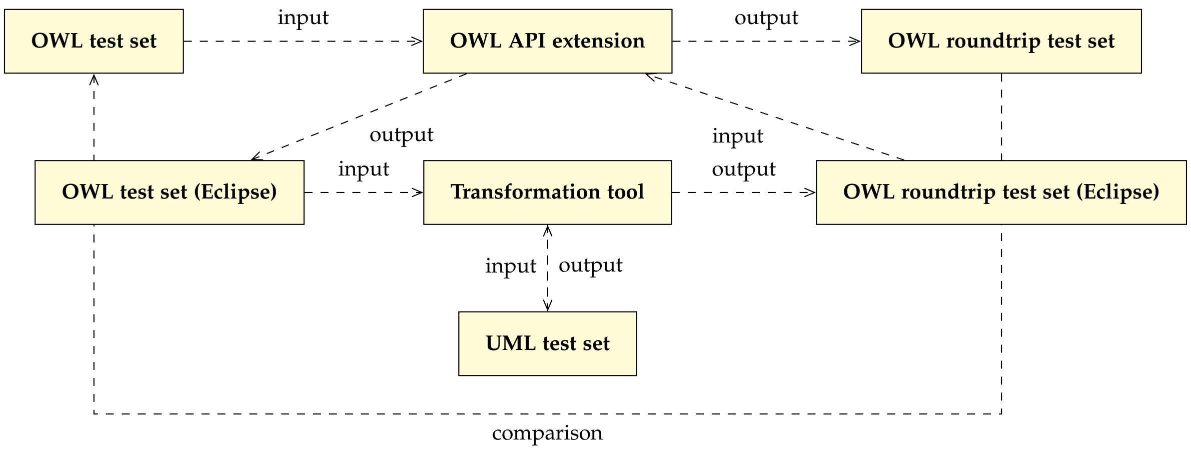


Figure 6.7: Illustration of OWL roundtrip process

6.2.1 Results

For clarity, we structured the results of the OWL roundtrip transformation based on axiom type. Figure 6.8 presents the cumulative counts of axiom types before and after the roundtrip test. The "Axioms after roundtrip" numbers were acquired by subtracting the missing axioms, and adding the added axioms to the original count. Figure 6.9 shows the cumulative counts of the missing and added axiom types, which we gathered using the comparison function provided in the OWL API.

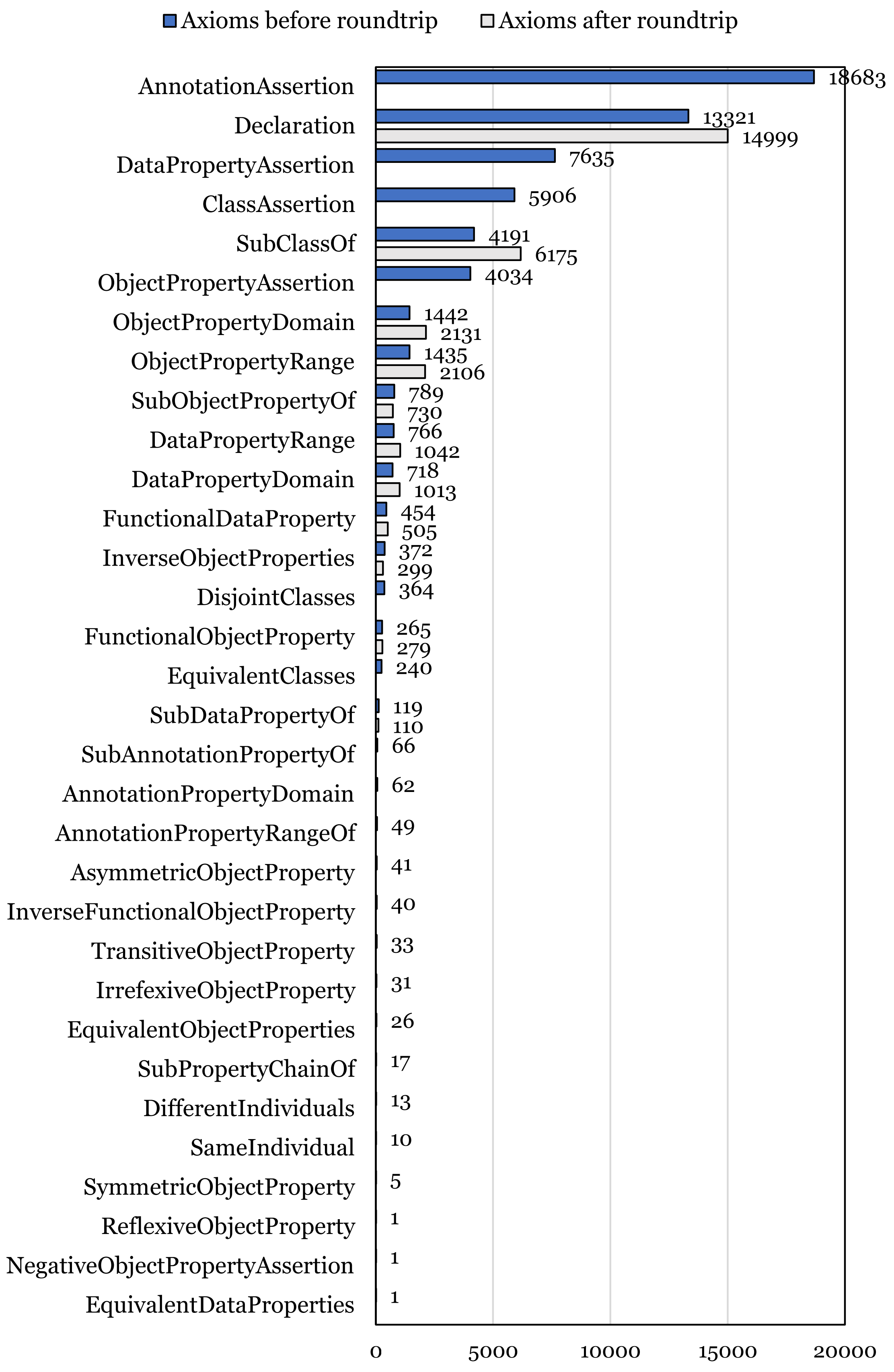


Figure 6.8: Axioms before and after the roundtrip

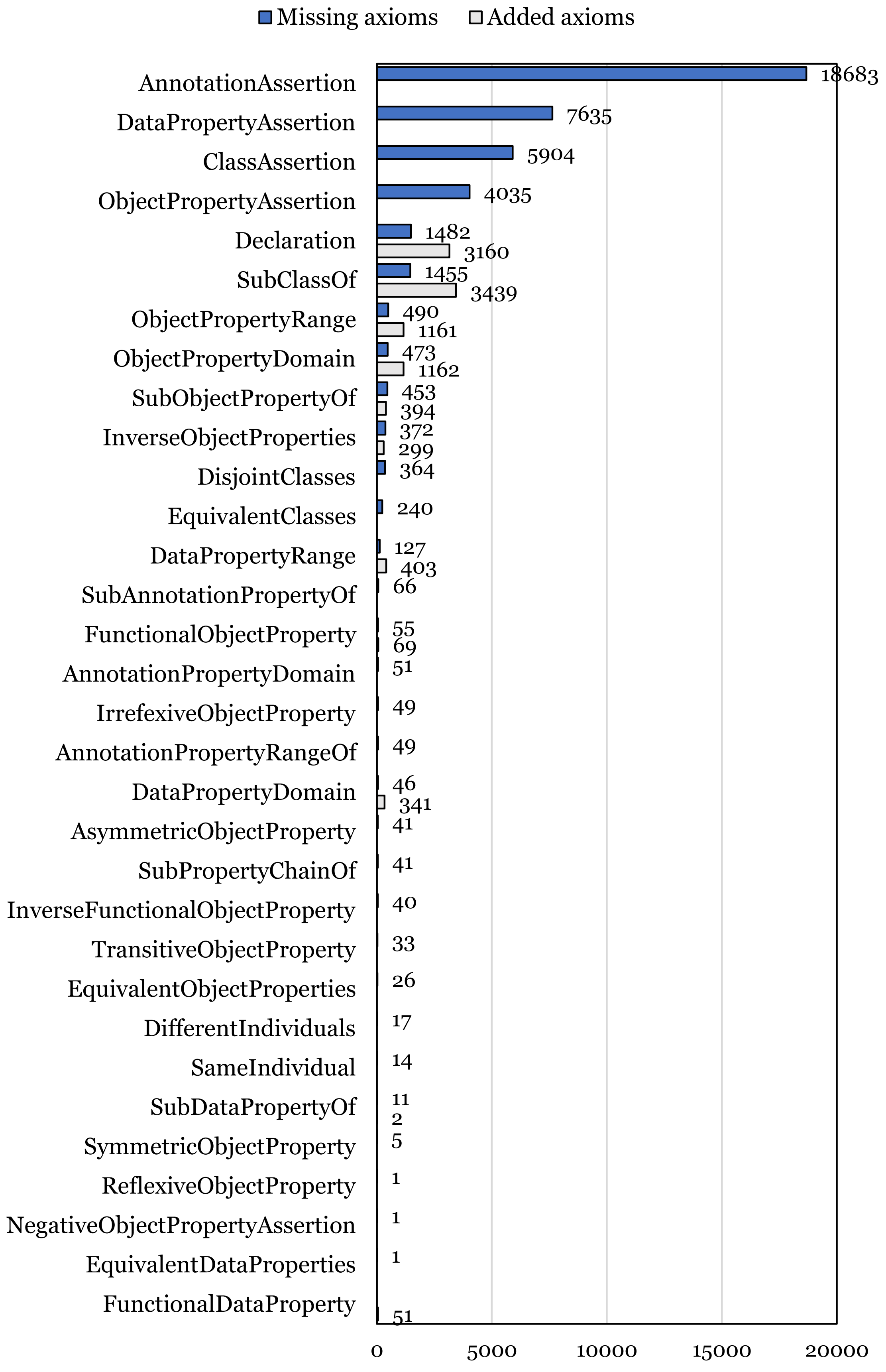


Figure 6.9: Missing and added axioms

Figure 6.10 illustrates the cumulative counts of lost and preserved axioms after the roundtrip. Figure 6.11 breaks down the lost axioms in axioms related to individuals, annotations and other axiom types.

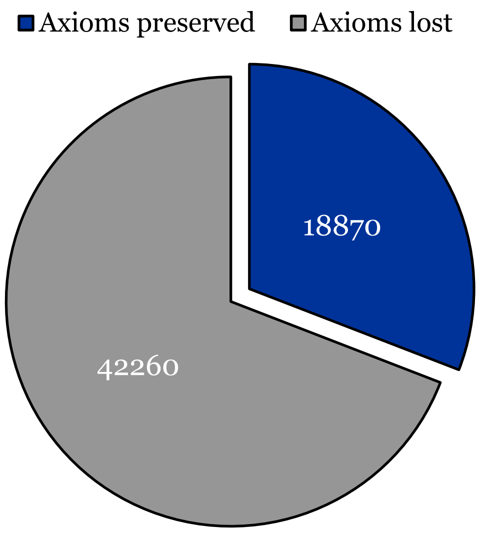


Figure 6.10: Cumulative count of lost ans preserved axioms

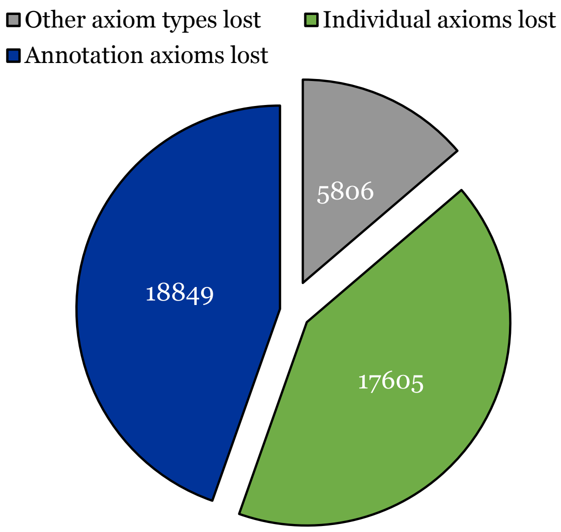


Figure 6.11: Breakdown of lost axioms

6.2.2 Discussion

Figure 6.8 and 6.10 show that a significant part of the axioms in the OWL test set is lost after the roundtrip transformation. However, Figure 6.11 shows that a large part of the lost axioms are related to Annotations and Individuals and only a small part is related to other axiom types. We discuss the missing annotation axioms in Section 6.2.2 and the missing individual axioms in Section 6.2.2. In the upcoming sections we evaluate the various missing and added axioms types shown in Figure 6.9.

Names of properties

If an `OWL:Property` has an inverse relationship with another property, these properties are mapped to an `UML:Association` in the OWL → UML transformation. In the transformation back from UML → OWL the names of the properties are different than those in the original test set. Consequently, there is a large number of missing Declaration(ObjectProperty), InverseobjectProperties, ObjectPropertyDomain, ObjectPropertyRange and SubObjectPropertyOf axioms. However, semantically equivalent axioms do exist in the roundtrip set, the only difference is that the properties are named differently.

Domain and range of properties

The state-of-the-art mappings suggest setting the domain and range of properties without range or domain specification to OWL:Thing. However, this introduces structure conflation in the UML $\rightarrow$ OWL mapping. If both data and object properties are transformed to UML:Properties with `OWL:Thing` as type, it is unclear whether the `UML:Property` should be mapped to a data or object property. We suggest mapping data properties without range to `UML:Properties` with `xsd:anytype` as type. By doing this, it is unambiguous whether an UML property should be transformed into an object or data property.

A consequence of this mapping is that in the roundtrip transformation information that is implicit in an OWL ontology is made explicit, namely, extra axioms are added that state the range of object and data properties are `OWL:Thing` and `xsd:anyType`. We suggest slightly adjusting the transformation of `UML:Property` types, in the UML $\rightarrow$ OWL transformation, by adding a constraint that an object or axiom range axiom is only added if the type is not `OWL:Thing` or `xsd:anyType`.

Cardinality constraints

Some axioms have cardinality constraints specified with `OWL:Thing` as range, even when a property range to a more specific class is specified. An example is the class key-entity pair from the prov#dictionary ontology[6]. The `OWL:Class` key-entity pair has the following SubClass axiom: `SubClassOf(:Key Entity Pair ObjectExactCardinality(1, pair-key, OWL:Thing)`. After the roundtrip transformation, `OWL:Thing` is replaced with a more specific class: `prov:entity`. This information was also available in the original ontology through reasoning, but the roundtrip makes this implicit information explicit.

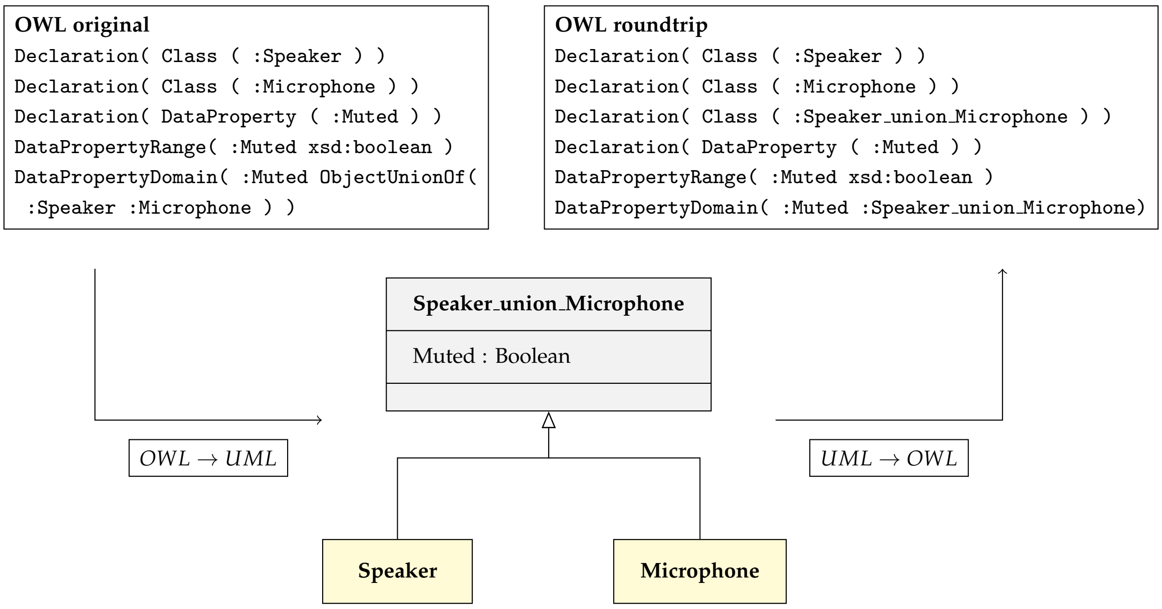
In Figure 6.9, we observed that a large number of DataMinCardinality and ObjectMinCardinality axioms were added. These added axioms specify that the min cardinality of object and data properties is 0. However, this information was also implicitly available in the original ontology. We suggest adding a constraint in the mapping from UML $\rightarrow$ OWL that does not map lower bound cardinality constraints of 0.

Union and intersection

OWL has constructs to express the union and intersection of classes. We implemented a transformation rule based on the state-of-the-art mappings that maps union and intersection constructs for classes. However, in Figure 6.9 we observed various missing axioms that use object union and intersection constructs,

[6] https://www.w3.org/ns/prov#KeyEntityPair

because the mappings of union and intersections of object properties are trapdoor mappings. We explain this using an example roundtrip transformation of a union constructs from an ontology that represents hardware[7].



OWL original

```
Declaration( Class ( :Speaker ) )
Declaration( Class ( :Microphone ) )
Declaration( DataProperty ( :Muted ) )
DataPropertyRange( :Muted xsd:boolean )
DataPropertyDomain( :Muted ObjectUnionOf(
 :Speaker :Microphone ) )
```

OWL roundtrip

```
Declaration( Class ( :Speaker ) )
Declaration( Class ( :Microphone ) )
Declaration( Class ( :Speaker_union_Microphone ) )
Declaration( DataProperty ( :Muted ) )
DataPropertyRange( :Muted xsd:boolean )
DataPropertyDomain( :Muted :Speaker_union_Microphone)
```

Figure 6.12: Roundtrip transformation of the hardware example

As the hardware example illustrates, the axiom containing the union construct is missing after the roundtrip. Instead a new class declaration is added that represents the union constructs. This construct cannot be transformed back into a union construct based on the available meta construct data alone. The same happens for the ObjectIntersection construct. Semantically, the specific arrangement of union and intersection representations in UML are equivalent to the OWL union and intersection, but the use of the OWL:ObjectUnion, OWL:ObjectIntersection constructs is lost in the roundtrip transformation.

Equivalences

OWL has properties to define that classes, data and object properties are equivalent to each other. This is done using the EquivalentDataProperties, EquivalentObjectProperties and EquivalentClasses axioms. After the roundtrip, these axioms are missing due to structure loss. A mapping for the equivalences between classes, data and object properties is implemented. By making equivalent data properties both a sub data property, and a super data property of the other, the semantics of the equivalence axiom can be preserved. However, a different construction is used and the equivalence axioms are lost in the roundtrip transformation.

[7] https://www.w3.org/2007/uwa/context/hardware.owl

Disjoint classes

Axioms that specify that classes are disjoint are missing after the roundtrip transformation. The state-of-the art does consider disjoint classes axioms. Though, Zedlitz discusses that in general, UML classes with different names are considered disjoint, hence there is no need to make explicit that classes are disjoint in the OWL $\rightarrow$ UML transformation [38]. However, the assumption that UML classes are disjoint is not made explicit in the UML $\rightarrow$ OWL transformation, resulting in the loss of disjoint classes axioms.

Individuals

Individuals are not mapped to UML in the state-of-the art mappings. As a consequence, all axioms that assert something about individuals are lost in the roundtrip transformation, namely: ClassAssertion, DataPropertyAssertion, ObjectPropertyAssertion, DifferentIndividuals and SameIndividuals.

Lack of features

There are several OWL constructs for which UML does not have an equivalent. If one of these constructs is used in an axiom for which UML does not have an equivalent, this axiom will be lost after the roundtrip transformation. The following constructs are not mapped due to a lack of features: someValuesFrom, allValuesFrom, inverseOf, complementOf. In addition the following axioms are not mapped due to a lack of features: Asymmetric, Transitive, Irreflexive, Symmetric, and ReflexiveObjectProperty axioms.

Annotation assertions

Finally, all annotation assertions are lost during the roundtrip transformation, since they are not discussed in the state-of-the art mappings. UML comments are similar to annotation assertions, but annotation assertions provide more functionality. Comments simply present a string with a comment (text), whereas with annotation assertions it is possible to specify an annotation property, which adds semantic information to the annotation. Examples of annotation properties are rdfs:label and rdfs:isDefinedBy. UML lacks the features to represent annotation properties. We suggest mapping all annotation properties to UML comments to preserve as much information as possible. Although the annotation property is lost during the roundtrip, the textual information is preserved. In addition, implementing this mapping preserves UML comments over a roundtrip transformation.

6.3 Conclusion

To answer **SQ4**: What information is lost in a roundtrip transformation from UML to OWL and OWL to UML based on the state-of-the-art mappings? Figure 6.2, 6.8 and 6.9 show the number of missing UML constructs and OWL axioms after the roundtrip transformations. Some constructs were not mapped in the state-of-the-art mappings, such as annotation assertions and navigability of associations. We suggested additional transformation rules and slight changes to the state-of-the-art mappings to preserves as much information as possible. These mappings can be found in Appendix B.

In both the UML and the OWL roundtrip, some information is lost. Most of the information in the UML models is preserved after a roundtrip transformation, except for the semantics of an aggregation association. In addition, constraints on the instances of the model and visual structuring of the model are lost after the roundtrip transformation. On the other hand, we found out that UML has severe feature lack to express OWL concepts. A large part of the OWL axioms does not have an UML equivalent. Furthermore, in some cases ontologies are more verbose after the roundtrip or include axioms that explicitly state what can be inferred through reasoning. However, despite this lack of features, a large number of axioms is preserved after the roundtrip, which indicates that there is an intersection between UML and OWL that is commonly used.

Chapter 7

Evaluating transformation results

Although the roundtrip evaluation gives an impression of which information is lost in the transformation process, it is insufficient to evaluate whether the automatically transformed UML models and OWL ontologies make sense in UML and OWL, respectively. We discuss a few peculiarities in the models and ontologies to explain subtle differences in semantics and modeling approaches and the implications of these differences. Section 7.1 discusses a number of observations made in automatically transformed OWL ontologies from UML models. Section 7.2 discusses a few observations made in automatically transformed UML models from OWL ontologies. Finally, Section 7.3 concludes this chapter by answering **SQ5**: What peculiarities exist in automatically transformed UML models and OWL ontologies?

7.1 UML to OWL evaluation

7.1.1 Stereotypes

When evaluating the resulting ontologies from the UML test set transformation we observed a number of peculiar OWL:Class instances. We observed class declarations for what we would commonly consider datatypes. For instance, Declaration(Class (:Boolean)) and Declaration(Class (:DateTime)). Furthermore, due to the modeling of datatypes as classes, each property that we would except to become a data property, turns into an object property. We can understand why this happens when looking at the original UML models. In all the UML models in the test set, datatypes have been defined as classes, which are annotated with a stereotype which classifies them as primitive or datatype. This way of modeling datatypes in UML is not captured by the current state-of-the-art mappings.

We have resolved this issue by adding project-specific transformation rules for our UML test set that transform datatypes that are modeled as classes to equivalent XSD datatypes where possible. In the back transformation from OWL → UML we propose mapping the XSD datatypes to the UML metamodel construct datatype instead of defining ones own datatypes using stereotypes. This example shows that when bringing metamodel transformations between UML and OWL to the industry, there could be a need for project specific transformations based on the specific use of UML and OWL. For example, NEN3610[1], a standard for the exchange of geo-information, contains stereotypes that could be mapped using project specific mappings.

7.1.2 Inheritance

Another peculiarity can be observed related to inheritance. Figure 7.1 represents a subset of CIM. After the transformation from UML → OWL this results in an ontology with three class declarations for document, skill and bank account. Furthermore, two OWL:SubClassOf axioms are added to represent the inheritance relations between document and skill, and document and bank account.

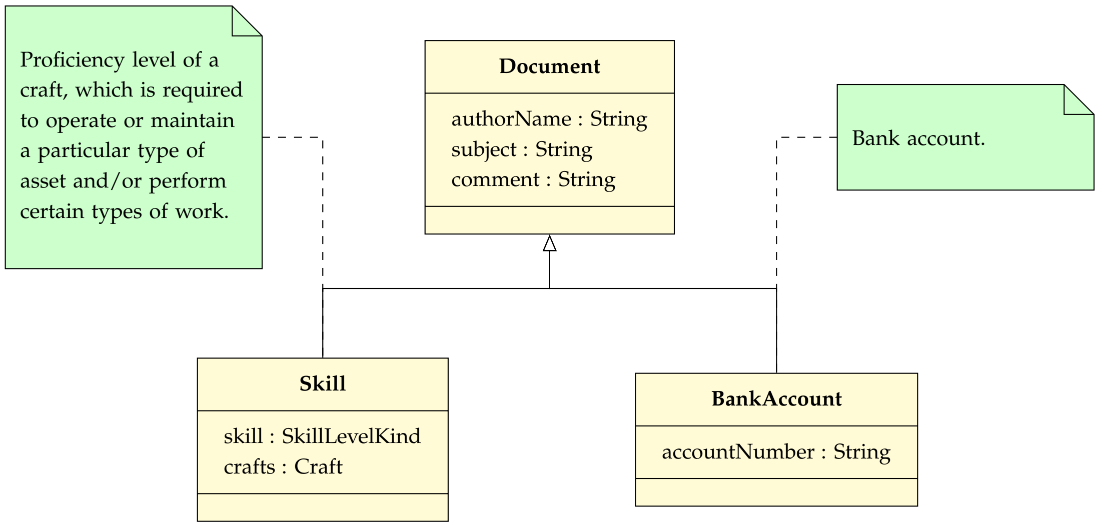


Figure 7.1: Representation of Skills and BankAccounts in CIM

However, the interpretation of the generated ontology might raise questions. In OWL, the subclass of relation implies that every sub class is of the same type of the super class. In other words, if we interpret the generated ontology along with the comments that were placed on the classes, we read that a bank account is some sort of document, and a proficiency level of a craft is some sort of document. In OWL, we assert knowledge, therefore asserting that a skill is some sort of document seems nonsensical conceptually. On the other hand, in the UML

[1] https://geonovum.github.io/NEN3610-Linkeddata/#iso19150-2

world, properties are local to classes and inherited through subclasses, which are visually represented (as it is shown in Figure 7.1). When looking at the UML diagram it is easier to understand, due to the specified properties within the classes, that we are not talking about the concept of a skill, or the concept of bank account, but rather a document that registers information about someone's skill in a certain craft, and a document that registers information about someone's bank account. We suggest that the abstraction level of class, which conceptualizes something, should be specified in more detail to prevent semantic misunderstandings in OWL. This could be done by creating class names that more explicitly represent the intended concept or adding a comment in which is clarified what the class aims to represent.

7.1.3 Properties

Another strange mapping occurs due to differences in the definitions of properties in UML and OWL. In UML, properties are defined locally to a class, whereas in OWL properties are standalone entities that can exist in the OWL universe [34]. After the transformation from UML $\rightarrow$ OWL, properties are modeled in a strange manner. We illustrate this using a CIM example presented in Figure 7.2.

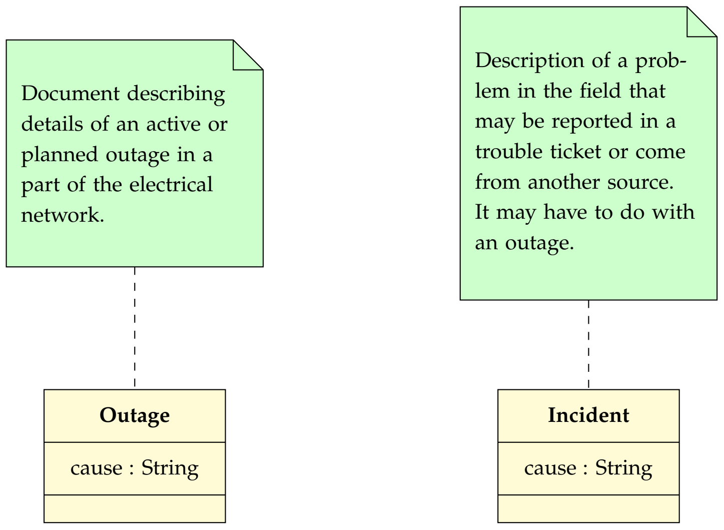

Document describing details of an active or planned outage in a part of the electrical network.
Description of a problem in the field that may be reported in a trouble ticket or come from another source. It may have to do with an outage.
Outage
cause : String
Incident
cause : String

Figure 7.2: CIM UML subset describing outage and incident

This example is transformed into two declared classes for outage and incident, cardinality constraint for the property, and two declarations for properties that are asserted as being disjoint. However, if we had started modeling in OWL we would have probably added only one axiom to represent the property cause, because one can argue that the property that models "being the cause of something" has the same semantic meaning for both outage and incident. However, this is not always the case, as Figure 7.3 illustrates.

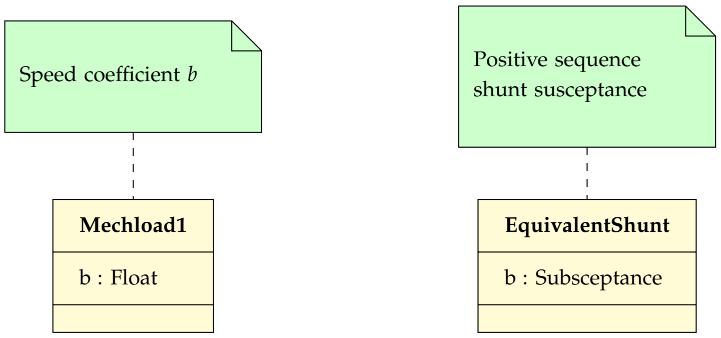


Figure 7.3: CIM UML subset presenting two classes with property named b

In Figure 7.3 we see that the property name b is both used to represent a property about ”speed coefficient (b)” and about ”positive sequence shunt susceptance”. It would be incorrect to mark these properties as semantically equivalent, therefore in this case, the added axioms for specifying properties as disjoint is required. We see that within the same UML model, both properties that are disjoint and properties that are semantically equivalent. A potential solution to this problem is making property names in UML more explicit, since by giving each property more descriptive names it becomes easier to determine whether the properties are semantically equivalent or not. In UML, there is no urgent need to give properties descriptive names, since they are always encapsulated within classes. A proper transformation to OWL, however, would benefit from more descriptive property names.

7.2 OWL to UML evaluation

7.2.1 Intersection and union in domain and ranges

We observed a number of odd constructions in the transformed UML models from OWL ontologies, such as the intersection helper class illustrated in Figure 7.4.

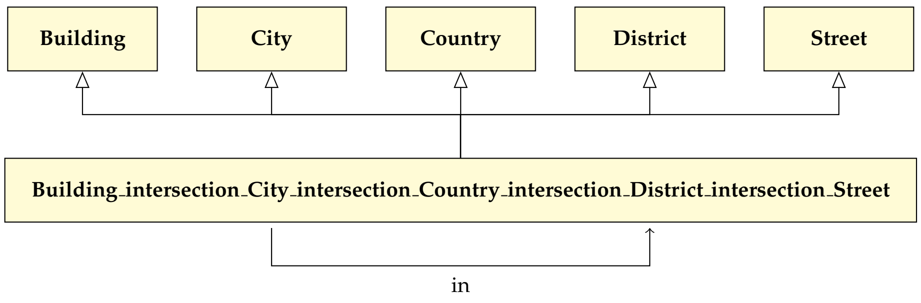

Building
City
Country
District
Street
Building_intersection_City_intersection_Country_intersection_District_intersection_Street
in

Figure 7.4: Subset of the transformed Hungarian postal address ontology

Figure 7.4 shows a subset of the in UML transformed Hungarian postal address ontology[2]. We can read this subset as: an object that is a building and a city and a country and a district and a street, can be located in another object that is a building a city a country a district and a street. However, semantically this does not make sense. If we look at the real world around us there are (to our knowledge) very few objects that are both a building, a city, a country, a district and a street at once. What was probably meant with the OWL ontology is that each of the classes building, city, country, district and street can be in a building, city, country, district or street. This makes more sense; a building and a street can be located in a city or country. In other words, in the original ontology the range and domain of the property ”in” is modeled incorrectly. What the modelers probably meant was the union of the classes instead of the intersection. This is a problem that we encountered in various ontologies in our test set. For instance, in an ontology about CT Eligibility[3], the object property named hasCTID has the intersection of exclusion and inclusion as domain. However, Exclusion and Inclusion is specified as disjoint in the ontology. Hence, the intersection of Exclusion and Inclusion is the empty set and the domain of ”hasCTID“ has been set to a domain that has no instances per definition. Which shows that modeling in OWL can be difficult.

7.2.2 Universal superclass

Another odd construction in the transformed OWL ontologies is the introduction of a universal superclass in the resulting UML model. Figure 7.5 shows a subset of a transformed ontology about wine[4] that uses a universal superclass in UML.

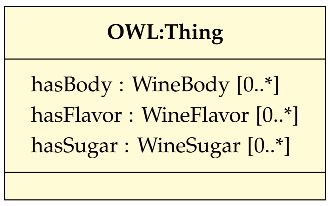

OWL:Thing
hasBody : WineBody [0..*]
hasFlavor : WineFlavor [0..*]
hasSugar : WineSugar [0..*]

Figure 7.5: Subset of the transformed wine ontology

In the wine ontology, the properties hasBody, hasFlavor and hasSugar have no domain specified, which results in them being encapsulated by an UML:Class that represents the OWL universal superclass. This peculiarity exists because of the difference in definition of properties between UML and OWL, as discussed

[2]https://www.w3.org/2001/sw/Europe/reports/dev_workshop_report_9/HungarianPostalAddress.owl

[3]https://www.w3.org/wiki/images/d/d8/HCLS%24%24ClinicalObservationsInteroperability%24CTEligibility.owl

[4]https://www.w3.org/TR/owl-guide/wine.rdf

in Section 7.1.3. Since in OWL properties are standalone entities that exist in the universe they can be specified without domain and range. In the OWL world, which is closely aligned to triple storage and graph databases, this is not a problem, since one can define arbitrary triple relations between objects and the standalone properties. For instance, one could specify in triple storage that the wine "Corbans Private Bin Sauvignon Blanc" hasFlavor "Strong". In UML, standalone properties without domain or range can become problematic, as UML is closely related to object-oriented programming languages, data schemas and relational databases. In these cases, we would like to know which specific classes can have these properties, to generate appropriate classes, data schemes and databases.

7.3 Conclusion

To answer **SQ5**: What peculiarities exist in automatically transformed UML models and OWL ontologies? By examining transformed UML models and OWL ontologies we found a number of peculiarities, which illustrate differences between UML and OWL. Even though automatically transformed constructs are syntactically correct, they do not necessarily make sense in UML and OWL. We explained these differences with the underlying modeling assumptions and relevant technologies of UML and OWL, respectively. In addition, we made a couple of suggestions to improve the automatically transformed models and ontologies. Namely, more expressive names for UML classes and properties, and explicit specification of domains and ranges of OWL properties.

Chapter 8

Final remarks

8.1 Conclusion

In this master's thesis we examined to what extent it is possible to automatically transform UML models into OWL ontologies and vice versa. In order to do this, we have implemented a bidirectional metamodel based transformation tool using QVT and the OWL API, based on the state-of-the-art mappings between UML and OWL proposed in the literature. Our transformation tool allowed us to overcome the current lack of case studies in academic work on metamodel transformations between UML and OWL. We moved away from examining metamodel transformations based on the general structure of UML and OWL, and instead performed case studies to evaluate the transformations. We performed roundtrip transformations and evaluated several transformed UML models and OWL ontologies. Based on the roundtrip transformations we have suggested additional transformation rules that were missing in the state-of-the-art mappings and some minor changes to some of the state-of-the-art mappings.

The roundtrip transformations showed that most UML constructs can be expressed in OWL, but OWL has various constructs for which UML has no equivalent. Despite this feature lack of UML, there are a significant number of constructs that are preserved throughout the roundtrip, which indicates that an intersection of similar modeling constructs exists between UML and OWL. Based on our evaluation of automatically transformed models and ontologies, we observed that even when syntactically information is preserved throughout a roundtrip transformation, peculiarities can be found in the models and ontologies obtained. These peculiarities exist due to slight differences in semantics of similar constructs, and differences in modeling approaches between UML and OWL.

To conclude, there is significant overlap between UML and OWL and most of this overlap allows for automatic transformation. However, there is also a significant loss of information due to lack of features, differences in semantics and modeling approaches. We believe that our transformation tool can function as a starting point for deriving OWL ontologies from UML models and models from ontologies. In addition, we hope that our observations, and explanations of the limitations and peculiarities in metamodel transformations between UML and OWL can assist modelers to create better ontologies and models.

8.2 Future work

Our work aimed to identify the problems and limitations of metamodel transformations between UML and OWL. We present some directions for future work:

- We discuss a notion of semantic equivalence based on interpretations and applications of models and ontologies. Future work could define a mathematical definition of semantic equivalence for a more formally grounded evaluation.
- A limitation of our research is in the evaluated UML models. In our research, we examined only a few UML models used within the energy domain, more specifically Alliander and Enexis. Future work could evaluate more UML models from different domains.
- One could research the possibility of implementing metamodel transformations between UML + OCL and OWL + SHACL. OCL could be used to capture the axioms of OWL, and SHACL could be used to capture the constraints defined by UML constructs. However, adoption of OCL by the industry is limited. Alternatively one could delve deeper into extending UML with profiles to capture OWL constructs, as others, such as LinkED, have attempted.
- One could try to bring the transformation tool to the industry. We are currently in contact with the CIM community to see if they can use our tool for a formal OWL publication of CIM. To stimulate this, one could try to improve the usability and quality of the transformation tool. For instance, by making explicit which UML constructs were mapped using which OWL constructs and vice versa.
- Finally, one could examine potential use cases for the automatically transformed ontologies and models. We identified one use case that is related to publishing data on the semantic web. For example, some researchers are using OWL ontologies to validate UML models [30].

Bibliography

[1] Colin Atkinson and Thomas Kühne. The essence of multilevel metamodeling. In *International Conference on the Unified Modeling Language*, pages 19–33. Springer, 2001.

[2] Florian Bauer and Martin Kaltenböck. Linked open data: The essentials.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[4] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 273–280. IEEE, 2001.

[5] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 3(1):1–207, 2017.

[6] Robert M Colomb, Anna Gerber, and Michael Lawley. Issues in mapping metamodels in the ontology development metamodel using qvt. In *The 1st International Workshop on the Model-Driven Semantic Web*, 2004.

[7] Alberto Rodrigues Da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.

[8] Sven Efftinge, Peter Friese, Arno Hase, Dennis Hübner, Clemens Kadura, Bernd Kolb, Jan Köhnlein, Dieter Moroff, Karsten Thoms, Markus Völter, et al. Xpand documentation. Technical report, Technical report, 2004-2010.(cited on page 64), 2004.

[9] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. Owl 2: The next step for owl. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, 2008.

[10] Andreas Grünwald. Evaluation of uml to owl approaches and implementation of a transformation tool for visual paradigm and ms visio. *Vienna, AT: Vienna University of Technology*, 2011.

[11] Giancarlo Guizzardi. Ontological foundations for structural conceptual models. 2005.

[12] Giancarlo Guizzardi, Heinrich Herre, and Gerd Wagner. On the general ontological foundations of conceptual modeling. In *International Conference on Conceptual Modeling*, pages 65–78. Springer, 2002.

[13] OUSSAMA EL HAJJAMY, Khadija Alaoui, Larbi Alaoui, and Mohamed Bahaj. Mapping uml to owl2 ontology. *Journal of Theoretical & Applied Information Technology*, 90(1), 2016.

[14] Martin Hepp. Ontologies: State of the art, business potential, and grand challenges. In *Ontology Management*, pages 3–22. Springer, 2008.

[15] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2(1):11–21, 2011.

[16] Ian Horrocks, Peter F Patel-Schneider, and Frank Van Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Web semantics: science, services and agents on the World Wide Web*, 1(1):7–26, 2003.

[17] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of computer programming*, 72(1-2):31–39, 2008.

[18] Stuart Kent. Model driven engineering. In *International Conference on Integrated Formal Methods*, pages 286–298. Springer, 2002.

[19] Kilian Kiko and Colin Atkinson. A detailed comparison of uml and owl. *University of Mannheim*, 2008.

[20] Nicolas Matentzoglu, Samantha Bail, and Bijan Parsia. A snapshot of the owl web. In *International Semantic Web Conference*, pages 331–346. Springer, 2013.

[21] Brian Matthews. Semantic web technologies. *E-learning*, 6(6):8, 2005.

[22] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, Carsten Lutz, et al. Owl 2 web ontology language profiles. *W3C recommendation*, 27:61, 2009.

[23] Jonathan Musset, Étienne Juliot, Stéphane Lacrampe, William Piers, Cédric Brun, Laurent Goubet, Yvan Lussaud, and Freddy Allilaire. Acceleo user guide. *See also http://acceleo. org/doc/obeo/en/acceleo-2.6-user-guide. pdf*, 2, 2006.

[24] James J Odell. *Advanced object-oriented analysis and design using UML*, volume 12. Cambridge University Press, 1998.

[25] Marcel Olij, Joep van Genuchten, Paul Stapersma, Jan Bruinenberg, and Arjan van Diemen. Tkilinked energy data. 2018.

[26] Object Management Group (OMG). Object management group model driven architecture, 2014.

[27] Object Management Group (OMG). Ontology definition metamodel version 1.1, 2014.

[28] Object Management Group (OMG). Meta object facility (mof) 2.0 query/view/transformation specification, 2016.

[29] Object Management Group (OMG). Omg unified modeling language (omg uml) version 2.5.1, 2017.

[30] Małgorzata Sadowska and Zbigniew Huzar. Representation of uml class diagrams in owl 2 on the background of domain ontologies. *E-INFORMATICA SOFTWARE ENGINEERING JOURNAL*, 13(1):63–103, 2019.

[31] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

[32] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE intelligent systems*, 21(3):96–101, 2006.

[33] Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In *OWLED*, volume 432, page 91, 2008.

[34] Dr Waralak V Siricharoen. Ontology modeling and object modeling in software engineering. *International Journal of Software Engineering and Its Applications*, 3(1):43–59, 2009.

[35] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

[36] Taowei David Wang, Bijan Parsia, and James Hendler. A survey of the web ontology landscape. In *International Semantic Web Conference*, pages 682–694. Springer, 2006.

[37] Zhuoming Xu, Yuyan Ni, Wenjie He, Lili Lin, and Qin Yan. Automatic extraction of owl ontologies from uml class diagrams: a semantics-preserving approach. *World Wide Web*, 15(5-6):517–545, 2012.

[38] Jesper Zedlitz. *Konzeptuelle Modellierung mit UML und OWL–Untersuchung der Gemeinsamkeiten und Unterschiede mit Hilfe von Modelltransformationen*. PhD thesis, Christian-Albrechts Universität Kiel, 2013.

Appendices

Appendix A

Mappings between UML and OWL

The Functional OWL Syntax[1] is used to represent OWL constructs. Mapping is abbreviated as map. The mappings are numbered incrementally.

A.1 UML to OWL

Table A.1: Mapping of UML Class

UML Construct	`Class`
Map1 [27, 30, 38]	`Declaration(Class(:Class))`

Table A.2: Mapping of UML Abstract Class

UML Construct	`Abstract Class`
Map2 [27, 30]	Not possible.
Map3 [38]	Could be mapped to a regular OWL Class, but there is no guarantee that the Class is not instantiated: `Declaration(Class(:Class))`

Table A.3: Mapping of UML Attributes

UML Construct	`Attributes`
Map4 [27, 30, 38]	If the attribute type is an UML Class or complex datatype:

[1] https://www.w3.org/TR/owl2-syntax/

	`Declaration(ObjectProperty(:Property))`
	`Declaration(ObjectPropertyDomain(:Property :Domain))`
	`Declaration(ObjectPropertyRange(:Property :Range))`

Map5 [27, 30, 38]	If the attribute type is a primitive type or enumeration:
	`Declaration(DataProperty(:Property))`
	`Declaration(DataPropertyDomain(:Property :Domain))`
	`Declaration(DataPropertyRange(:Property :Range))`

Notes	In [38] DisjointDataProperties or DisjointObjectProperties are added to prevent different properties from being interpreted as a single property.

Table A.4: Mapping of UML Binary Association (different classes)

UML Construct	`Binary Association (between two different classes)`

Map6 [27, 30, 38]	`Declaration(ObjectProperty(:Property1))`
	`Declaration(ObjectProperty(:Property2))`
	`Declaration(ObjectPropertyDomain(:Property1 :Domain))`
	`Declaration(ObjectPropertyDomain(:Property2 :Domain))`
	`Declaration(ObjectPropertyRange(:Property1 :Range))`
	`Declaration(ObjectPropertyRange(:Property2 :Range))`
	`InverseObjectProperties(:Property1 :Property2)`

Table A.5: Mapping of UML Binary Association (to itself)

UML Construct	`Binary Association (class to itself)`

Map7 [30]	**Map6** +
	`AsymmetricObjectProperty(:Property1)`
	`AsymmetricObjectProperty(:Property2)`

Table A.6: Mapping of N-ary Association

UML Construct	`N-ary Association`

Map8 [27, 30]	Partially possible, by declaring a new class and n in [3...*] properties with domain and ranges:
	`Declaration(Class(:Class))`
	`Declaration(ObjectProperty(:Property-n))`
	`Declaration(ObjectPropertyDomain(:Property-n :Domain))`
	`Declaration(ObjectPropertyRange(:Property-n :Range))`

Table A.7: Mapping of Association Class

UML Construct	`Association Class`
Map9 [27, 30]	**Map6** without the ObjectPropertyRange declarations + `ObjectPropertyDomain(:Property1 ObjectUnionOf(:Property2 :AssociationProperty)) ObjectPropertyDomain(:Property2 ObjectUnionOf(:Property1 :AssociationProperty)) Declaration(Class(:AssociationClass)) Declaration(ObjectProperty(:AssociationProperty)) ObjectPropertyDomain(:AssociationProperty ObjectUnionOf(:Property1 :Property2)) ObjectPropertyRange(:AssociationProperty :AssociationClass)`

Table A.8: Mapping of UML Ordered Association

UML Construct	`Ordered Association`
Map10 [38]	Not possible.

Table A.9: Mapping of UML Aggregation

UML Construct	`Aggregation`
Map11 [27, 30]	Not possible.
Map12 [38]	**Map6** + restrictions on aggregations by adding: `AsymmetricObjectProperty(:AggCompEnd) IrreflexiveObjectProperty(:AggCompEnd)`

Table A.10: Mapping of UML Composition

UML Construct	`Composition`
Map13 [27, 30]	Not possible.
Map14 [38]	If the composition is navigable from part to whole: **Map12** + : `FunctionalObjectProperty(:AggCompEnd)`
Map15 [38]	If the composition is navigable from whole to part: **Map12** + : `InverseFunctionalObjectProperty(:AggCompEnd)`
Map16 [38]	If the composition is bidirectionally navigable: choose between **Map14** & **Map15**.

Table A.11: Mapping of UML Generalization (between classes)

UML Construct	`Generalization (between classes)`
Map17 [27, 30, 38]	`SubClassOf(:SubClass :SuperClass)`

Table A.12: Mapping of UML Generalization (between associations)

UML Construct	`Generalization (between associations)`

Map18 [27, 30, 38]	`SubPropertyOf(:SubProperty1 :SuperProperty1)`
	`SubPropertyOf(:SubProperty2 :SuperProperty2)`

Table A.13: Mapping of UML Generalization (between datatypes)

UML Construct	`Generalization (between datatypes)`

Map19 [38]	In general not possible. Exception for generalization of enumerations:
	`Declaration(DataType(:SuperType))`
	`Declaration(DataType(:Enumeration1)`
	`Declaration(DataType(:Enumeration2)`
	`Declaration(DataTypeDefinition(:Enumeration1`
	`DataOneOf("value1", "value-n")))`
	`Declaration(DataTypeDefinition(:Enumeration2`
	`DataOneOf("value1", "value-n")))`
	`Declaration(DataTypeDefinition(:SuperType`
	`DataUnionOf(:Enumeration1 :Enumeration2)))`

Table A.14: Mapping of UML GeneralizationSet

UML Construct	`GeneralizationSet`
Map20 [27, 30, 38]	Case { Incomplete, Disjoint }. Specify for each pair of classes in the generalization: `DisjointClasses(:Class1 :Class2)`
Map21 [30, 38]	Case { Complete, Disjoint }: `DisjointUnion(:Class :SubClass1 :SubClass-n)`

Map22 [27, 30, 38]	Case { Complete, Overlapping }:
	`EquivalentClasses(:Class ObjectUnionOf(:SubClass1`
	`:SubClass-n))`

Table A.15: Mapping of UML Cardinality Restrictions

UML Construct	`Cardinality Restrictions`
Map23 [30, 38]	Case property with lower bound equal to the upper bound [x..x]: `SubClassOf(:Class ObjectExactCardinality(x :P :R))`

Map24 [38]	Case property with lower bound and upper bound [1..1]:
	`SubClassOf(:Class ObjectExactCardinality(1 :P :R))`
	`SubClassOf(:Class FunctionalObjectProperty(:Property))`

Map25 [27, 30, 38]	Case property with lower bound and upper bound [x..y]:
	`SubClassOf(:Class ObjectMinCardinality(x :P :R))`
	`SubClassOf(:Class ObjectMaxCardinality(y :P :R))`

Map26 [30]	Case property with lower bound and upper bound [x..*]: `SubClassOf(:Class ObjectMinCardinality(x :P :R))`
Map27 [30]	Case property with multiple value ranges e.g. [1, 6..7]: `SubClassOf(:Class ObjectUnionOf(...))`
Notes	**Mappings 23-27** describe mappings for properties that are transformed into OWL object properties. If properties are transformed into data properties, DataUnionOf, DataExactCardinality, DataMinCardinality and DataMaxCardinality is used instead.

Table A.16: Mapping of UML Primitive DataTypes

UML Construct	`Primitive DataTypes`
Map28 [30, 38]	If possible map the datatype to an equivalent XML datatype.
Map29 [38]	If the datatype is user defined: `DataTypeDefinition(...)`

Table A.17: Mapping of UML Structured DataTypes

UML Construct	`Strucutered DataTypes`

Map30 [30, 38]	Mapped to a class, key and properties with domain and range for each structured type attribute:
	`Declaration(Class(:Class)`
	`Declaration(DataProperty(:Attribute))`
	`DataPropertyDomain(:Attribute :Class)`
	`DataPropertyRange(:Attribute :Type)`
	`HasKey(:Class (:Attribute))`

Table A.18: Mapping of UML Enumeration

UML Construct	`Enumeration`
Map31 [27, 30, 38]	`Declaration(DataType(:EnumName))` `DataTypeDefinition(:EnumName DataOneOf("values"))`

Table A.19: Mapping of UML Package

UML Construct	`Package`
Map32 [27, 38]	`Ontology(...)`

Table A.20: Mapping of UML Comment (on class)

UML Construct	`Comment (on class)`
Map33 [30]	`AnnotationAssertion(rdfs:comment :Class "comment")`

A.2 OWL to UML

Table A.21: Mapping of OWL Ontology

OWL Construct	`Ontology`
Map34 [27, 38]	`UML Package`

Table A.22: Mapping of OWL Class

OWL Construct	`Class`
Map35 [27, 38]	`UML Class`

Table A.23: Mapping of OWL SubClassOf

OWL Construct	`SubClassOf`
Map36 [27, 38]	Case SubClassOf with declared classes: `UML Generalization`
Map37 [38]	Case SubClassOf with necessary conditions as SubClass: Not Possible.

Map38 [38]	Case SubClassOf with necessary conditions as SuperClass: Possible but not specified.

Table A.24: Mapping of OWL EquivalentClass

OWL Construct	`EquivalentClass`
Map39 [27, 38]	Case EquivalentClass with declared classes: A pair of `UML Generalization`

Table A.25: Mapping of owl:Thing

OWL Construct	`owl:Thing (universal superclass)`
Map40 [27, 38]	`UML Class` named "Thing"

Table A.26: Mapping of OWL ObjectUnionOf

OWL Construct	`ObjectUnionOf`
Map41 [27]	Case ObjectUnionOf with declared classes: `UML Generalization` with instances of the union as subclasses.
Map42 [38]	Case ObjectUnionOf with declared classes: `UML GeneralizationSet` with instances of the union as subclasses, and an abstract class as superclass.

Table A.27: Mapping of OWL ObjectIntersectionOf

OWL Construct	`ObjectIntersectionOf`
Map43 [27, 38]	Case ObjectIntersectionOf with declared classes: `UML Generalization` with instances of the intersection as superclasses and a new abstract class as subclass.

Table A.28: Mapping of OWL DatOneOf

OWL Construct	`DatOneOf`
Map44 [27, 38]	`UML Enumeration`

Table A.29: Mapping of OWL Properties

OWL Construct	`Properties`
Map45 [27, 38]	Case DataProperty or ObjectProperty with no inverse and not inverse functional: `UML Class Attribute`
Map46 [27, 38]	Case ObjectProperty with inverse or inverse functional: `UML Association`

Table A.30: Mapping of OWL Domain and Range

OWL Construct	`Domain and Range`
Map47 [27, 38]	Case no class declared: `UML Class named` "thing"
Map48 [27, 38]	Case one class declared: `UML Class`
Map49 [27, 38]	Case more than one class declared: Intersection of the declared classes using an `UML Generalization`

Table A.31: Mapping of OWL SubObjectProperty

OWL Construct	`SubObjectProperty`
Map50 [27, 38]	`UML Association Generalization`

Table A.32: Mapping of OWL Cardinality Restrictions

OWL Construct	`Cardinality Restrictions`
Map51 [27, 38]	Case FunctionalProperty: `UML Property with [0..1]`
Map52 [27, 38]	Case InverseFunctionalProperty: `UML Association with [0..1]`
Map53 [27, 38]	Case ObjectMinCardinality: `UML Property with lower bound`
Map54 [27, 38]	Case ObjectMaxCardinality: `UML Property with upper bound`
Map55 [27, 38]	Case ObjectExactCardinality: `UML Property with lower bound qual to upper bound`
Notes	**Mappings 53-55** are the same for DataMinCardinality, DataMaxCardinality and DataExactCardinality

Table A.33: Mapping of OWL HasValue

OWL Construct	`HasValue`
Map56 [38]	`UML Class Attribute with default value`

Table A.34: Mapping of OWL InverseObjectProperties

OWL Construct	`InverseObjectProperties`
Map57 [38]	`UML Association`

Table A.35: Mapping of OWL HasKey

OWL Construct	`HasKey`
Map58 [38]	`UML Key`

Table A.36: Mapping of OWL Primitive types

OWL Construct	`Primitive Types`
Map59 [38]	Map to established UML libaries for XML datatypes

Appendix B

Added and changed mappings

We present an overview of our suggested mappings and changes to existing mappings. The same format is used as in Appendix A. The count of the mappings is continued from A.

B.1 UML to OWL

Table B.1: Mapping of UML Dependency

UML Construct	`Dependency`
Map60	`AnnotationAssertion(http://purl.org/dc/terms/requires :Subject : Value)`

Table B.2: Mapping of UML Association Navigability

UML Construct	`Association Navigability`
Map61	Case unidirectional: **Map4** or **Map5**
Map62	Case bidirectional: **Map6** or **Map7**
Map63	Case unspecified: **Map6** or **Map7**

Table B.3: Mapping of UML Comment (on property)

UML Construct	`Comment (on property)`
Map64	`AnnotationAssertion(rdfs:comment :Property "comment")`

Table B.4: Mapping of UML Cardinality Restrictions

UML Construct	`Cardinality Restrictions`
Notes	In **Mappings 23-27**, we suggest not mapping cardinalities with a lower bound of 0.

B.2 OWL to UML

Table B.5: Mapping of OWL Properties

OWL Construct	`Properties`
Map65	Case DataProperty: `UML Class Attribute`
Map66	Case ObjectProperty: `UML Association`

Table B.6: Mapping of OWL Annotation Assertion

OWL Construct	`Annotation Assertion`
Map67	`UML Comment`

Table B.7: Mapping of OWL Range

OWL Construct	`Range`
Map68	Case no range declared and ObjectProperty: `UML Class` named ”thing”
Map69	Case no range declared and DataProperty: Map to `xsd:anyType`

Appendix C

OWL construct usage (percentages)

For each metamodel construct that occurs in the OWL test set, the percentage of ontologies that uses the construct is shown.

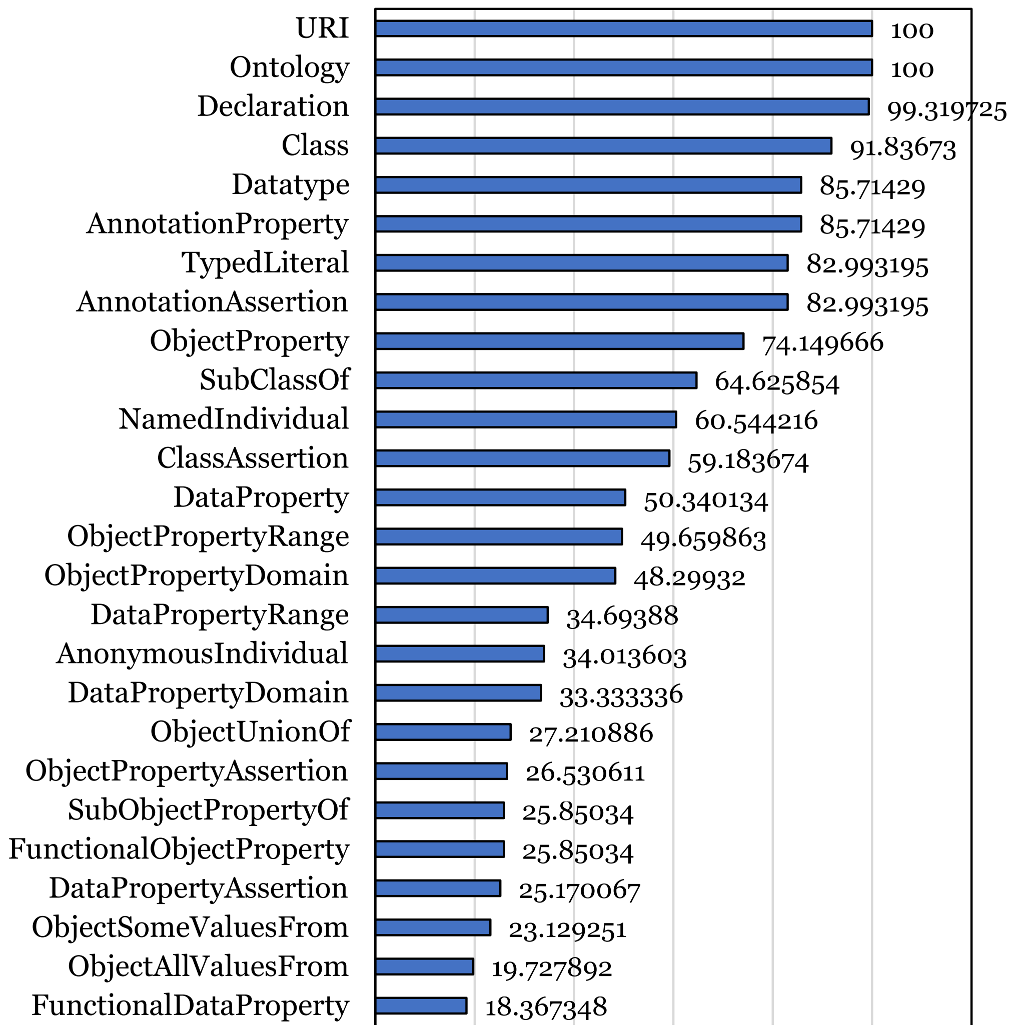


Figure C.1: OWL construct usage (continues next page)

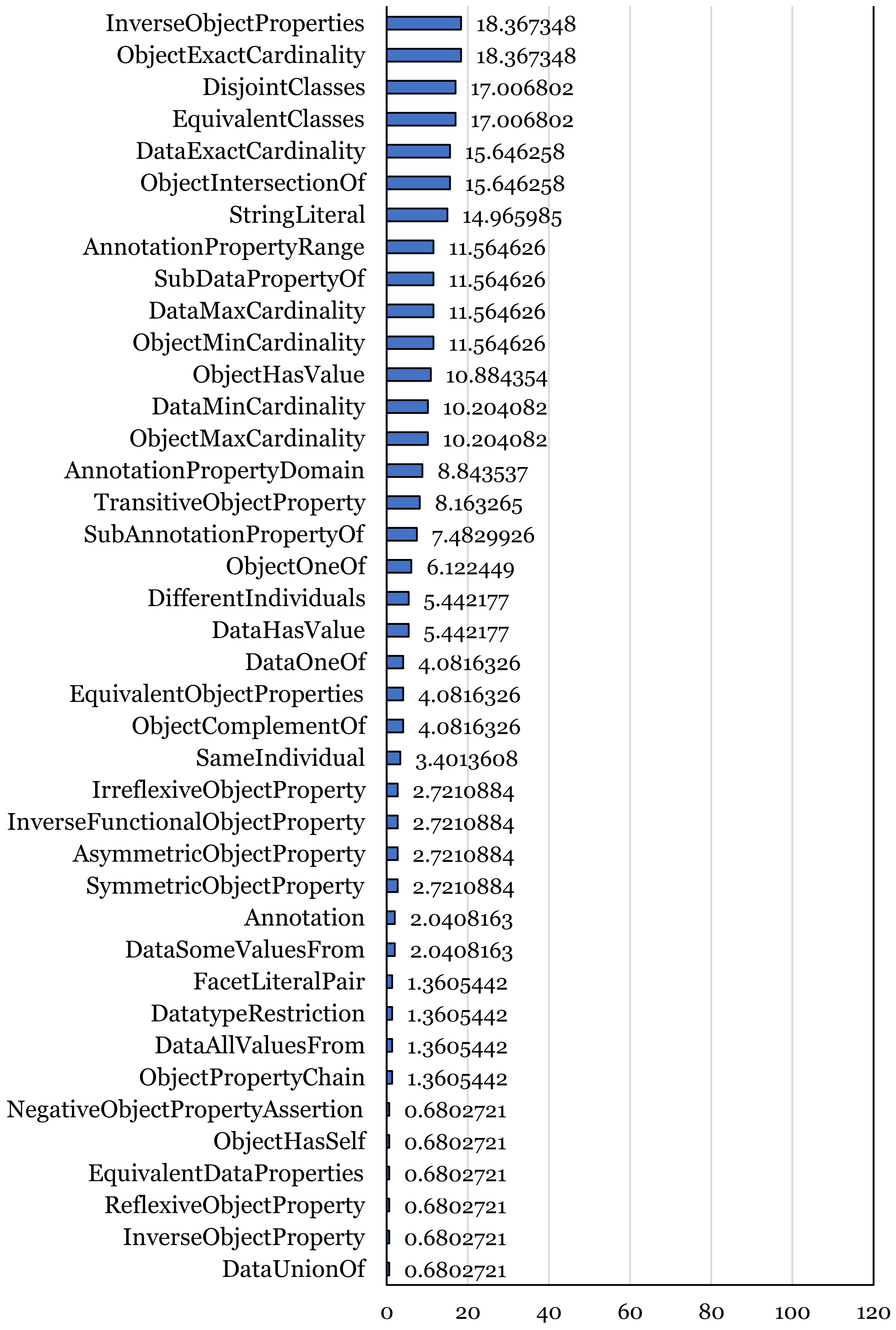


Figure C.2: OWL construct usage (continued)