REIHE INFORMATIK
TR-2008-004

A Detailed Comparison of UML and OWL

Kilian Kiko und Colin Atkinson
University of Mannheim
– Fakultät für Mathematik und Informatik –
Lehrstuhl für Softwaretechnik
A5, 6
D-68159 Mannheim, Germany

A Detailed Comparison of UML and OWL

Kilian Kiko

University of Mannheim

Colin Atkinson

University of Mannheim

Abstract

As models and ontologies assume an increasingly central role in software and information systems engineering, the question of how exactly they compare and how they can sensibly be used together assumes growing importance. However, no study to date has systematically and comprehensively compared the two technology spaces, and a large variety of different bridging and integration ideas have been proposed in recent years without any detailed analysis of whether they are sound or useful. In this paper, we address this problem by providing a detailed and comprehensive comparison of the two technology spaces in terms of their flagship languages – UML and OWL – each a de facto and de jure standard in its respective space. To fully analyze the end user experience, we perform the comparison at two levels – one considering the underlying boundary assumptions and philosophy adopted by each language and the other considering their detailed features. We also consider all relevant auxiliary languages such as OCL. The resulting comparison clarifies the relationship between the two technologies and provides a solid foundation for deciding how to use them together or integrate them.

Index Terms – Analysis, language comparison, representation languages, ontology languages, software modeling languages, syntax, semantics, interpretation assumptions.

1 Introduction

A major trend in IT over the last decade has been the move towards greater inter-connectivity of systems and the creation of enterprise-wide computing solutions. Whereas in the past, applications were typically written to solve a single, localized problem using their own private data format, today they are expected to fit flexibly into large multi-purpose systems that span whole enterprises or even planets (in the case of the Internet). This trend has shifted the critical activity of systems development from the traditional “programming” of imperative code to the creation of rich, interchangeable representations of information and knowledge.

Two major “paradigms” have emerged in recent years to support this activity, each with its own terminology, standards bodies, research communities and flagship language. One is the so-called “modeling” paradigm that places “models” at the center of the development process. Model-driven development evolved primarily in the software engineering community and has the OMG’s Unified Modeling Language (UML) [17] as its flagship language. The other is the so-called “ontology engineering” paradigm that places “ontologies” at the center of the development process. Ontology engineering primarily evolved from the artificial intelligence community and has the W3C’s Web Ontology Language (OWL) [1] as its flagship language.

Since the two paradigms were developed with different roles in mind, the two approaches have until recently remained relatively separate, with each community tailoring its language(s) to its own particular needs. For a long time, therefore, there was little interest in comparing the two approaches. However, with the growing importance of knowledge representation in mainstream software development and the primea facea ability of both paradigms to serve this role, there is a growing interest in both academia and industry in understanding how the two paradigms relate to one another, which offers the best capabilities under which circumstances, and how they can be used together.

Recent literature identifies a variety of possible ways of bringing model-driven development and ontology-driven development together, ranging from the provision of "bridges" that allow one approach to be used within the technology framework of the other (e.g. [16], [11], [12]) to the wholesale integration of the paradigms at a fundamental level (e.g. [13], [15]). Unfortunately, no study to date has provided a detailed and comprehensive comparison of the two technology spaces based on the capabilities offered to end users. Existing proposals either compare the technologies at a very superficial level of abstraction or neglect one of their key elements. The most extensive comparison to date is that of Hart et. al. [2], but while providing a valuable overview of their relationship, this work does not take the capabilities of the Object Constraint Language (OCL) [22] or the underlying assumptions of the two domains into account. A detailed summary of the previous work in this regard is provided in [44].

The basic rationale for this paper is that it is not possible to make good decisions about which technology to use for which purpose, how to effectively use the two technology spaces together and how ultimately to integrate them without a detailed and comprehensive understanding of how they compare and differ. The goal of this paper is thus to investigate the technology spaces in terms of their flagship languages – UML and OWL respectively. Although these are by no means the only languages used to create software models or ontologies, they are highly representative of the two technology spaces. The resulting understanding of the relationship provides the required solid foundation for making practical usage and integration decision. However, it is beyond the scope of this paper to provide guidelines for this.

The rest of the paper is structured as follows. In the next section, we describe our comparison approach and discuss the background to each “language”. In Section 3, we compare the goals and objectives of each language (i.e. semantic domain) together with their concrete representation (concrete syntax). Then in Section 4 we discuss the major underlying (and largely implicit) assumptions that characterize each language’s approach to knowledge representation and influence the interpretation of language constructs (semantic mapping). This is followed in Section 5 by a detailed, feature-by-feature comparison of the languages (i.e. their abstract syntax). Section 6 then discusses the conclusions that can be drawn and makes various suggestions and recommendations.

2 Comparison Objects and Method

Unqualified uses of the names “UML” and “OWL” can cause great confusion because they refer to languages with complex evolution histories, multiple variants and rich infrastructures. In order to compare the two technology spaces it is therefore necessary to first define precisely which parts of which languages are being considered.

As a modeling language, the UML is conceptually embedded within the four-level modeling framework depicted in Fig. 1. The UML itself (i.e. the definition of the UML language) is regarded as being an instance of the Meta Object Facility (MOF) [18] at the M2 level of the framework. It is therefore commonly referred to as a “metamodel”. The OMG envisages many other metamodels at the M2 level as instances of the MOF tailored to the needs of different domains. An example of another standard metamodel that accompanies the UML is the Common Warehouse Metamodel (CWM) [19].

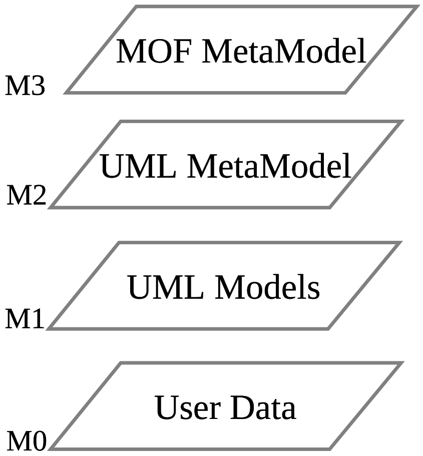


Fig. 1 Four layer metamodeling architecture

The relationship between the UML and the MOF is more complex than it appears in Fig. 1. UML2.0, the recent major revision of the UML standard, is not only defined as an instance of the MOF, but it is also partly a specialization of it. More precisely, the full MOF and the UML (UML 2.0 Superstructure) are both specializations of a common core set of language constructs defined in a shared package known as the InfrastructureLibrary (UML 2.0 Infrastructure) [20]. This actually defines the core language constructs that characterize the basic object-oriented, intensional information representation approach for which the UML is most widely known. These constructs focus on the representation of static structural information that forms the foundation of UML modeling – the so-called "class diagram". This is the part of the UML that we are considering in this paper. Since this part of the UML is also the foundation of the MOF, our comparison applies just as much to the MOF as it does to the UML.

An important complement to the static modeling features of the UML is the Object Constraint Language (OCL) [22]. Strictly speaking, this is not part of the UML itself since it is defined in a separate specification. However, it is such an indispensable complement to the basic modeling features of UML that in practice the two languages should always be considered together. The OCL is essential in this comparison since it provides the means to add precision to UML models. In the remainder of this paper, we use the term "UML Full" to denote the combination of the UML/MOF core and the OCL and the term "UML Lite" to reference to the UML/MOF core alone (without OCL). If we use the unqualified term UML we mean UML Full (i.e. UML2.0 [17], [20] + OCL2.0 [22]).

As a standard, OWL [1] is conceptually embedded within the four-level modeling framework depicted in Fig. 2. This is intended to indicate that OWL is built on top of the schema definition language RDF Schema (RDFS) [45], which in turn is built on top of the basic metadata markup language Resource Description Framework (RDF) [36]. “Built on top of” does not mean the same thing as "instance of" and so although they are superficially similar, Fig. 1 and Fig. 2 show different things. “Built on top of” is more like specialization than instantiation, although the precise semantics are not clear. Just as the core part of the MOF is effectively embedded with the UML, therefore, the RDF and RDFS are embedded within OWL.

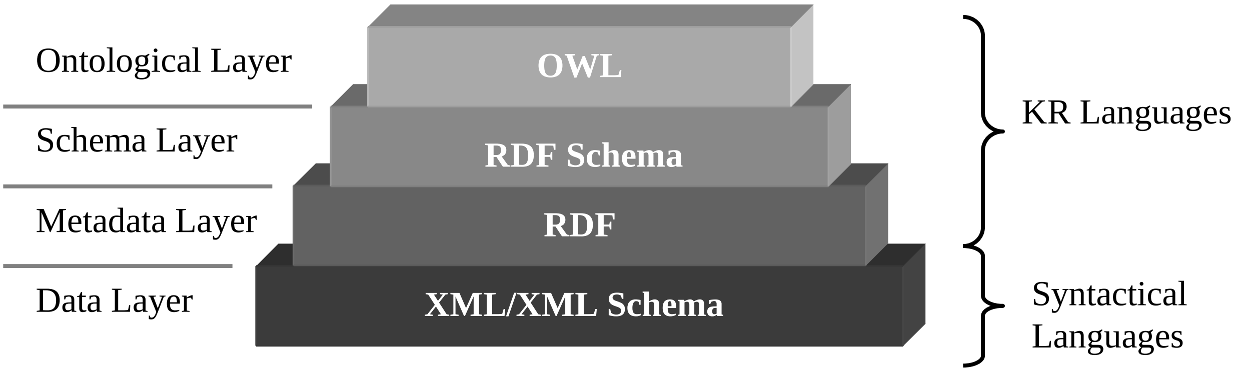


Fig. 2 Four level modeling framework

The relationship between the languages is a usage and specialization relationship. OWL uses and specializes RDFS, which uses RDF. As pointed out by Pan and Horrocks [25] the Semantic Web hierarchy is a “functional hierarchy”. This means that every language has a certain function in the architecture. However, the languages were not originally devised for this purpose. In fact, it is more of an evolutionary hierarchy that includes legacy languages.

The OWL standard actually defines three sub languages OWL-Lite, OWL-Full and OWL-DL. The first of these provides a minimalist language with a very restricted scope and focus.

It cannot reasonably be compared to the UML therefore. OWL-Full is the most general version of the language, which concentrates on maximizing expressiveness. OWL-DL is a subset of OWL-Full, which focuses on providing decidability. Since both tradeoffs are of interest in comparison to the UML, we consider both variants of OWL. Unless otherwise stated, the comparison is based on features in UML 2.0 [17], [20], [22] and OWL [1], [32], [38]. Having clarified what we aim to compare, we will now present our comparison method.

There have been several comparisons of (modeling) languages in the past (cf. [24], [27], [28], [29], [30], [31]). Our approach builds on these, but with a unique focus that is tailored to the objects of the comparison. In linguistics, a language is characterized in terms of its syntax and semantics. The syntax can be further subdivided into concrete syntax and abstract syntax and the semantics into the semantic domain and semantic mapping (cf. [24]). A language comparison should take all of these into consideration. We compare the concrete syntaxes and semantic domains in section 3, the semantic mappings in section 4 and the abstract syntaxes in section 5.

However, the most important issue in language comparison is the comparison of the language subject and expressiveness. The first is determined by the semantic domain and the latter by the abstract syntax and the semantic mapping.

The concrete syntax specifies the actual representation of language statements. Each concrete representation symbol or icon maps onto a language construct. The concrete syntax itself has no meaning. It is therefore possible for a language to have several discrete representations. Hence, the concrete syntax has no influence on the expressiveness of a language.

The semantic domain describes the area of application of a language, the so-called “universe of discourse” (UoD). Everything stated in a language is a part or element of that universe. Two languages have to have at least intersecting semantic domains to be worth comparing, i.e. they have to deal (at least in parts) with the same things.

The abstract syntax defines the range of language constructs that describe the set of allowed statements expressible in the language. The user of a language composes these constructs to create statements in the language that reflect situations in the universe of discourse. The number of common language constructs affects the degree of equivalence in expressiveness between two languages,

The semantic mapping or interpretation function determines the meaning of language expressions. As the name implies, a language statement or construct is mapped or related to its semantics, i.e. its denotation. The semantic mapping is the actual act of interpretation. It is the mental connection of terms to the conceptualizations they represent. The mapping acts on certain rules that can be more or less explicitly defined. The rules in turn underly a set of assumptions and constraints (e.g. human comprehension capabilities and characteristics). Formal languages use so-called model-theoretic semantics (also called Tarski-style semantics [46]) to provide a formal meaning to language expressions. The model-theoretic semantics explicitly defines the mapping of abstract syntax constructs into the semantic domain. The mapping is described as a mathematical (interpretation) function from the vocabulary of language constructs into the semantic domain (in terms of set theory). We present these interpretation definitions in section 5. Hart et. al. have already presented a feature-based (abstract syntax) comparison in [2] (later revised in [16]) though less complete and formally than the one in section 5.

3 Comparison of the language representation and purpose

A language is on the one hand represented by its concrete syntax (i.e. the perceivable form of its constructs and statements) and on the other by the constructs and statements which reflect objects and situations in a conceptual universe of discourse (i.e. the semantic domain). These two aspects of the languages under investigation are examined in this section.

3.1 The Concrete Syntax

The concrete syntax specifies the actual representation of language statements. The UML for example has a diagrammatic representation and an XML representation called XML Metadata Interchange (XMI) [47], and RDF also has an XML notation RDF/XML [48].

Clearly, the concrete syntaxes of UML and OWL are not the same, even when XML is used as the concrete notation, because the tag names are distinct and there are potential synonyms and homonyms. As mentioned in the last section, this has no influence on the expressiveness and thus on the compatibility of the two languages and is therefore not considered further in the comparison.

3.2 The Semantic Domain

The languages were devised to fulfill different purposes. While OWL supports the representation of knowledge about a system, UML was developed primarily to support the construction of a (software) system. Nevertheless, the underlying goal of both is the representation of a system. This is why a large number of studies (e.g. [14], [16], [13], [23]) agree that the basic subject matter of the two languages is essentially the same, since both are used to define object-centered, intension-based representations of knowledge about a system or universe of discourse. Object-centered means that the main component of a knowledge representation is an object and its relation to other objects (equivalent to the object-oriented approach in software engineering). A single relation is known as a property or association. “Intension-based” reflects the fact that the knowledge representations can represent an abstraction of the elements in the concrete system. Instead of explicitly representing every object in a system the common properties of a set of objects are represented (i.e. objects are classified according to their properties).

Both languages therefore have two basic layers of knowledge representation, namely concrete, instance-level information called extensional (or assertional) knowledge representation (AKR), and abstract, type-level information called terminological (or intensional) knowledge representation (TKR). For extensional knowledge representation, the semantic domain is the set of objects and objects relations (object states) under consideration at a given point in time. For terminological knowledge (i.e., ontologies and class models), the semantic domain is the set of valid sets of objects and object relations (i.e., the set of possible object models). The languages' semantic domain in both cases therefore consists of objects that can form tuples of related objects.

However, the two languages differ in the way that knowledge is understood and expressed. This leads to a difference in the role of terminological knowledge and extensional knowledge. Ontology engineering uses a terminological knowledge representation approach to classify extensional knowledge and to infer new knowledge from it. If the extensional knowledge contradicts the ontology, it is identified as not satisfying the ontology. System models (terminological knowledge representations) in software engineering are used to represent and constrain the allowed set of system states. A concrete system state (extensional knowledge representation) must satisfy the constraints laid down by the system model to be a legal instance of it. Both approaches start with extensional knowledge to define a terminology, but while ontology engineering reuses this ontology to apply it to other extensional knowledge to deduce further extensional knowledge, in software development it is used for the construction of a (single) system (i.e. extensional knowledge representation). The consequence is that extensional knowledge in the UML is intended to be dependent on a terminological knowledge representation (i.e., a class model).

The difference between the constructional motivation for UML and the representational motivation behind OWL is reflected in their different underlying assumptions. These, in turn, influence the set of available constructs as well as their interpretation, as will be observed in the following sections.

4 Comparison of Interpretation Assumptions

The knowledge representation and software engineering communities typically have different goals in mind when capturing information. They make corresponding but different assumptions regarding the interpretation of language expressions or statements. The set of assumptions influences the semantic mapping between language constructs and their denotations. While they are fixed or institutionalized in OWL, they are more variable in the UML. The UML allows different interpretations of a given language construct depending on the viewpoint. This is similar to situational factors in linguistics, where the pertinent assumptions are referred to as pragmatics. We will focus on one viewpoint and set of assumptions with respect to the UML which we call the representation-oriented set of assumptions or paradigm [10]. This viewpoint is the dominant view of object-oriented analysis. Its focus is problem domain modeling. This is also the focus of the conceptual design phase in database development and the viewpoint of the Entity-Relationship modeling paradigm [43]. Since problem domain modeling is similar to ontology definition, a representation-oriented interpretation of the UML has the largest overlap with the assumptions that usually underpin OWL.

The most fundamental assumption is the “world model”. Both languages adopt an object-centered world view which results in them having a large number of equal language constructs (e.g. class and object) and similar language constructs (e.g. property) as can be seen in the next section. Both languages differentiate between terminological (intensional) and extensional (assertional) knowledge. In OWL DL the extensional knowledge is represented by a set of names (individuals) that is disjoint from the set of names of the terminological knowledge (classes, properties and datatypes). OWL does not use an extra packaging construct to separate terminological knowledge from extensional knowledge. It therefore broadens the term “ontology” to include instance data [1]. The UML accommodates the concept of an object model but this is an implicit idea since no object model construct exists in the UML metamodel. Therefore, as in OWL, assertional knowledge in the UML can be included with the intensional knowledge in one model. A UML model that exclusively contains terminological knowledge is known as a class model, while a model that solely contains assertional knowledge is known as an object model.

4.1 Open-World versus Closed-World Interpretation Assumptions

A fundamental issue in knowledge representation is the underlying world assumption. Reiter [41] distinguishes two kinds of world assumptions: closed-world (CWA) and open-world (OWA). The UML is oriented towards data modeling and system construction so that even when used to create a conceptual model of a domain the represented knowledge is implicitly viewed as being complete. OWL, in contrast, interprets models as potentially representing partial knowledge. These two different assumptions have fundamental implications on modeling practice. In a CWA model, it is not necessary to close the definition of a classifier's intension or an object's specification because the closure is implicitly presumed. UML Lite therefore does not inherently support explicit closure axioms for classes or explicit restrictions on individuals. In an OWA model, on the other hand, explicit closure axioms are needed to specify what cannot apply to a concept. This necessitates value restrictions on class and object properties to represent complete knowledge. A closed-world language directly represents the system under study meaning that there is a functional relation between the language expressions and the modeled world. An open-world language on the other hand distinguishes the knowledge of the world from the world itself. An expression can therefore have multiple interpretations in the modeled world.

4.2 Unique Name Assumption & Synonyms

OWL's goal of supporting the semantic web imposes certain requirements on the language such as the need to support distributed and interoperable ontologies. OWL meets these requirements by allowing the definition of synonyms for classes, properties and individual descriptions. The language therefore grants the existence of synonym concept definitions and provides constructs to state explicitly that two classes, properties or individuals are equivalent. Identifiers for classes, properties and individuals are distinct and the same name always refers to the same element. This means that the definition of homonyms is not feasible. However, a given element (object, property or class) may be referred to by several different names. This means that the relation between concrete syntax expressions and their semantic domain counterparts is a non-injective function, i.e. names have no unique interpretation, whereas in a language like UML the function is injective. The assumption that every name has a unique interpretation (i.e. an interpretation that is not shared with any other name) is known as the unique name assumption (UNA).

Individuals in OWL can be viewed as either equal or different. Class and property extensions can be disjoint, overlapping or equal, meaning that the class or property has an equivalent interpretation. The extensions are indeed independent of the UNA as identifiers are bound to intensions and one extension can have different intensions with different names. Therefore, it is possible for differently named classes to relate to the same extent. If classes and properties are themselves treated as individuals, as in OWL Full, it is also possible to state that two classes are the same, meaning that their intensions are equal. UML Full does not support synonyms because it was designed to support "offline" usage in which a model is developed by one person or a team of designers that are committed to a fixed and clear terminology. The UNA in the UML impedes synonyms in the assertional knowledge space (i.e., the object model) and in the intensional view of classes and associations and it leads to a limited view of class and association extensions that ignores equivalence specification.

4.3 Global Scope and First-Class Status Properties

Like classes, properties can have first-class status. This is equivalent to the treatment of classes and relations as unary and binary predicates in first-order predicate calculus. First-class status recognizes properties as equal classification elements. Properties are then interpreted as sets of tuples of objects (i.e., sets of object relations). The intension of a property is based on relation-specific characteristics, (e.g. symmetric, transitive, functional, etc.) and the context specifications (i.e., the association end types). UML associations and OWL properties, like classes, can form taxonomies and set-based operations can be applied to them.

First-class status also entails that properties have a meaning of their own. In other words, their general semantics is independent of a specific domain and range context. This is known as decontextualization. In the case of open-world semantics, these relations have global scope, meaning that these properties apply to every object in the semantic domain unless they are constrained by domain and range specifications. OWL properties therefore have global scope - that is, by default they have the universal concept as their domain and range. An OWL property therefore always applies to any class and an individual is always allowed to have additional features that are not prohibited by the class definition.

The UML does not sanction global scope of attributes and associations because of the closed-world interpretation placed on these constructs. The intensions of associations are implicitly closed which implies that "context-free" associations by default apply to no object, i.e., their domain and range is the bottom concept, which is the complement of the universal concept.

It is not possible to state that two associations are equivalent, as this is not intended in UML Full. UML Lite diagram syntax is based on token-referentiality, which prohibits distributed specifications. To specify that one association is the abstraction of other associations, association specialization or redefinition is used to indicate refinements of the original association. This allows multi-contextualized, classifying associations to be specified (or respectively their refinements to be represented) in UML Lite. Otherwise, anonymous associations can be used as sub-associations of the property (association) to state association redefinition. Since the OCL is type-referential, UML Full handles associations as if they were OWL properties. UML Full attributes are mere design- and implementation-level constructs. In contrast to associations, attributes in UML are not first-class model elements. They depend on their classifiers, cannot form taxonomies and have no decontextualized interpretation.

4.4 Sufficient Conditions & Defined vs. Primitive Concepts

One of the purposes of OWL ontologies is to facilitate automatic classification – that is, the computer-driven classification of individuals based on the terminological knowledge provided in an ontology. Automatic classification of individuals is only feasible if classes offer sufficient conditions for membership. OWL supports the definition of essential concept properties through necessary and sufficient conditions and distinguishes between primitive and defined classes since its goal is to support class-based reasoning on assertional information. OWL implements this feature by applying two terminological axioms, distinguishing the defining statement from the defined expression, and using anonymous class descriptions. The UML does not provide native assistance in the definition of sufficient conditions or defined classes, since the UML is intended to define constraints that have to hold in a concrete version of a represented system. However, it can be shown that necessary and sufficient conditions can be emulated with OCL constraints (cf. Section 5.3).

4.5 Metaclasses

The OWL DL sublanguage of OWL restricts instances of the OWL Class construct to having a mere extensional interpretation. OWL Full relaxes this constraint and enables information elements to exhibit an object and type duality. This in turn allows classes to have types of their own. These second-order classes are called metaclasses. UML does not support metaclasses directly. It has two mechanisms to define classes that have classes as their instances. The profile mechanism and stereotype construct can define metaclasses as subclasses of the UML Class construct. However, these classes are in fact regarded as an extension of the modeling language and not part of the modeled system [4]. Power types, a further UML construct, allow the definition of metaclasses that serve as power sets for other (regular) classes (i.e., their instances are subclasses of a class). Power types therefore rely on a class that is partitioned by them, which is the normal semantics of a metaclass. However, an OWL metaclass is the power type of the universal class *owl:Thing*, which is not available in the UML. Furthermore, the power type construct does not fit naturally into the strict four layer modeling hierarchy of the UML [7].

	UML Lite	UML Full	OWL
Object-centered	✓	✓	✓
Open-world assumption (not CWA)	×	×	✓
Global scope properties	×	×	✓
First-class status of Properties	✓	✓	✓
Synonyms (not UNA)	×	×	✓
Metaclasses	✓*	✓*	✓
Sufficient conditions	×	✓	✓
Universal concept	×	✓*	✓

Table 1. Interpretation assumptions overview

Table 1 provides a summary of the interpretation assumptions that underpin OWL and UML in their normal usage modes. ✓ means the assumption applies in the language, × means the assumption does not apply and ✓* means the assumption applies in certain limited cases.

5 Comparison of Language Constructs

The abstract syntax of a language defines the set of constructs that are used to make statements in the language. Ideally, we would like to compare the abstract syntaxes of the two languages under study on a construct-by-construct basis but this is made difficult by the different assumptions that the two languages apply to the interpretation of their constructs and expressions. Every time an assumption affects the interpretation of a construct, we will explicitly consider and discuss the different possible interpretations.

In the following subsections, every OWL language construct used for conceptual modeling is compared to its UML Lite and/or UML Full counterpart (where one exists) and the respective interpretation assumptions will be considered. Then, following this, the UML constructs that have no direct counterpart in OWL Full are discussed accordingly.

The comparison is based on the model-theoretic semantics given for OWL in [38] and the definition of the UML 2.0 and OCL 2.0 abstract syntaxes in [17] and [22].

5.1 Individual & Object

UML and OWL are fundamentally object-centered. Objects are the core medium for assertional (extensional) knowledge representation (AKR). Terminological knowledge, in contrast, abstracts away from individuals to capture knowledge common to sets of objects. The constructs *owl:individual* in OWL and *InstanceSpecification* in the UML are used to represent individuals in a representation of assertional knowledge.

The semantics of each construct is as follows. Let Δ be the semantic domain and IE the interpretation function (semantic mapping). Every name *x* in the AKR is mapped to an object in the semantic domain:

$$IE(x) \in \Delta$$

Since OWL takes an open-world assumption about the semantic domain, an OWL AKR has several satisfying interpretations. In other words, a given OWL AKR can be compatible with (i.e. represent one of) many different object populations in the real world. The closed world assumption of UML, in contrast, admits only one interpretation for a UML AKR. In other words, a UML AKR is compatible with only one real-world population of objects – exactly the set that is represented in the AKR.

UML and OWL allow individuals to be defined with facts like class membership and property values. OWL defines class membership with the RDF *type* relation or by using the name of the classifier. UML uses the *instanceOf* dependency and the colon-based naming convention. Both languages allow objects to be defined without a type definition (see [17]). However, the purpose and interpretations differ in the two languages. While OWL allows objects to be unspecified and therefore instances of the universal type, UML objects are only instances of the InstanceSpecification classifier and used as examples for possible object states without identity. An untyped UML Lite object resembles an illustration of an arbitrary object, which is not a legal member of an AKR. Individuals (AKR members) represented in UML Lite object models always have to have a concrete classification, which in practice means a user-defined classifier since there is no universal classifier.

Property values are defined in OWL by using the name of the property and relating it to an object or data value. In the UML, slots are used for the definition of datatype property values and links for the specification of object property values. Both languages support an unspecific property value, i.e. a value that is present but its type is unknown. In UML Full, the value "null" is used [22] as the unspecified property value.

The unique name assumption in UML requires every object specification to be distinct, while OWL allows individual descriptions to represent the same individual. OWL therefore further provides three constructs for stating facts about the identity of individuals. These are *owl:sameAs*, *owl:differentFrom*, and *owl:AllDifferent*. As synonyms are not supported in the UML, these constructs are not available in the language.

Instance specifications in UML do not need to have a closure axiom because of the CWA. Every instance specification in the UML is implicitly closed. This makes it impossible to state that an object specification is not closed.

In summary, both constructs are essentially equivalent apart from the differing interpretations yielded by their respective world and name assumptions and the fact that UML Lite does not allow individuals with no explicit type (i.e., having only the universal classifier as their classifier).

5.2 Classes

A class is interpreted as a set of objects. Classes are elements of a terminological knowledge representation (TKR), which is known as a class model in UML and an ontology in OWL.

Each class primitive *c* in a TKR is interpreted as a set of objects in the semantic domain Δ:

$$IE(c) \subseteq \Delta, IE(c) \in 2^{\Delta}$$

The set of classes (Class) in a TKR is interpreted accordingly as the set of class extensions:

$$IE(Class) = 2^{\Delta}$$

An instance of a class can explicitly be related to its classifier with the type property (*rdf:type* or *instanceOf*). The type property is interpreted as a set of pairs, where the first element is an object in the AKR and the second element is a class from the TKR:

$$IE(type) \subseteq IE(Thing) \times IE(Class)$$

A class can be viewed as an intension and an extension. The class extension is the set of objects to which the class applies, while the intension is the collection of constraints and conditions that apply to instances of that class. The set of class features therefore determines the set of objects in the semantic domain and their representations in an AKR that are instances of the class. The major difference between each languages' interpretation of the class concept is that UML perceives classes as being implicitly closed with respect to their features while an OWL class comprises all members of the respective UML class and all objects that exceed the UML class's intension in terms of their features. A UML class' extension is therefore a subset of the respective OWL class' extension and the size of the extension of a class in UML's interpretation is less than or equal to the size of the same class's extent in OWL's interpretation.

$$IE_{UML}(C) \subseteq IE_{OWL}(C)$$

Since neither language allows the world assumption to be changed locally for a single classifier, UML and OWL class expressions have different interpretations (i.e. can be representations of completely different sets of UoD objects). To facilitate a feature-by feature comparison, we abstract from the implicit completeness assumption of UML models without any impact on the semantics of single UML features.

$$IE_{OWL} = IE$$

To support data types the semantic domain is divided into two disjoint sets, the object domain OD and the data type domain DD (cf. [26]). The formal semantics of the universal class is the set of all objects in the object domain:

$$IE(Thing) = OD$$

The bottom concept is interpreted as the empty set:

$$IE(Nothing) = \varnothing$$

And the foundational class Literal is interpreted as the set of data values in the data type domain:

$$IE(Literal) = DD$$

OWL contains a standard foundation ontology that is implicitly available in every user-modeled ontology. The foundation ontology holds the universal concept *owl:Thing* as well as the bottom concept *owl:Nothing* and the class of all data values *rdfs:Literal*. The UML does not directly feature these classes, but many object-oriented systems offer the universal classes of objects and data values. The OCL in fact possesses the universal concept of objects and data values in one class *OclAny* and the bottom concept *OclVoid* [22]. The OCL defines

OclAny as the supertype of all types in the UML model and the primitive types in the OCL [21] (or as behaving like it [22]). This includes the primitive data types, the enumerations and UML user model types. To be able to make statements about the set of all individuals in the domain we interpret the extension of *OclAny* (i.e., *OclAny.allInstances()*) to be equal to the extension of the universal concept (*owl:Thing* in OWL). The operation *allInstances()* is applicable and correct as it returns only objects of user defined classes [22] and *OclAny* comprises all user defined classes.

The class intension in OWL is specified by class axioms that relate a class name to a class description. This can either be a superclass of the specified class, an equivalent class or a class that is disjoint with the specified class. Class descriptions are the class name, a set of property restrictions, an exhaustive enumeration of the extension's members, or a logical combination of other class descriptions. OWL does not support variables at all. All axioms are represented by sets (classes). Property restrictions are interpreted as anonymous subclasses of the universal concepts by placing constraints on the class extension. The same is true for all other unnamed class descriptions (intersection, union, complement, enumeration). The UML in principle supports anonymous class definitions by unnamed classes [17], although this is rarely used in practice. The OCL supports variables and set-based representation of axioms.

5.2.1 Class Axioms

Classes in OWL are defined by class axioms having a defined expression (the class identifier) and a defining statement (an intersection of class descriptions). The UML does not distinguish between the defining statement and the defined expression. Classes in OWL can be defined as subclasses of other classes, as equivalent to another class, or as disjoint from another class. These are the OWL class axioms. Class equivalence and class inclusion have a special purpose in OWL. While OWL can use the taxonomy construct (*rdfs:subClassOf*) for the definition of a class hierarchy in the same way as the UML generalization relationship, it uses this construct in combination with anonymous class definitions as a defining statement to describe *necessary conditions* for the class membership of an individual. To express *sufficient conditions* OWL uses the equivalent-class relationship. OWL uses anonymous class descriptions because it does not support variables. The UML, in contrast, admits only necessary conditions as discussed in Section 3.3. It therefore does not need a means to distinguish between primitive and defined classes (necessary and sufficient conditions). This implies that the UML Lite has only one axiom. Nevertheless, UML Full offers a means to handle sufficient conditions with OCL constraints. We therefore have to present a UML counterpart for every OWL class constructor considering the two axioms (i.e., class equivalence and class inclusion).

As explained in 5.2, we abstract from the general impact of the world assumption with respect to the classes' extensions in an enclosing ontology, as it is not possible to revise the foundational assumption locally. We first list the OWL class axioms in their basic interpretation (5.2.1.1 to 5.2.1.4) and then the class constructors for primitive and defined classes (5.2.2.1 to 5.2.2.6).

To facilitate a clear and concise comparison of language constructs we summarize their properties using the tabular template illustrated in Fig. 3. In the top left hand cell of the table, we identify the OWL abstract syntax element that is the subject of the comparison. Below this, in the bottom left hand cell we give an example of how this construct is normally used in OWL ontologies, and on the right, in the top right cell we give a textual description of the construct. The bottom right cell is organized into three parts: one containing the semantics of the construct in first order logic, one containing the equivalent UML Lite feature and associated description, and one containing appropriate OCL statements. The OCL statement is always based on an appropriate UML Lite class model whose elements (classes, associations) are referenced by the OCL expression. Together the UML Lite feature and the OCL expression form the UML Full representation of the OWL construct. This implies that there could be two UML Full representations of the OWL construct, one using a single UML Lite feature and one using a combination of UML Lite elements and OCL statements.

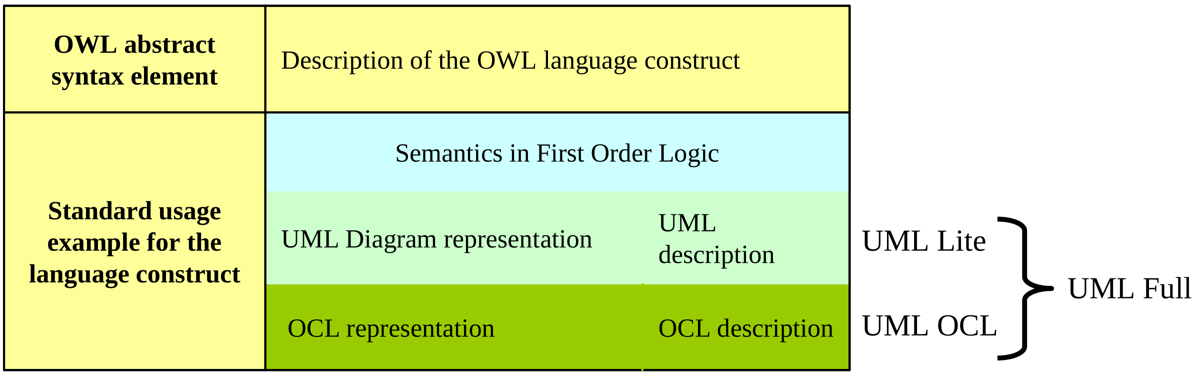
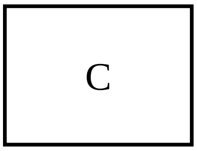
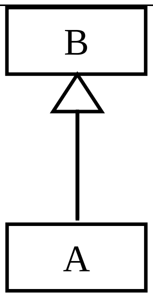

OWL abstract syntax element
Description of the OWL language construct
Standard usage example for the language construct
Semantics in First Order Logic
UML Diagram representation
UML description
OCL representation
OCL description
UML Lite
UML OCL
UML Full

Fig. 3. Comparison template

5.2.1.1 Atomic Classes

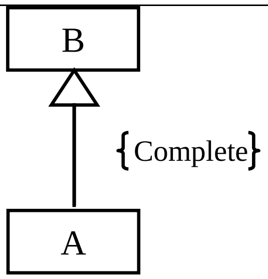
<table>
<tr><td><b>owl:Class</b></td><td colspan="2">The atomic class has no intension. It can only be interpreted by its name that has a meaning in the world outside the ontology. The atomic class is a class description that is simultaneously a class axiom.</td></tr>
<tr><td rowspan="2"><b>Class C</b></td><td colspan="2"><i>C</i></td></tr>
<tr><td> </td><td>UML Lite represents atomic classes as named elements of type <i>Class</i> without further features.</td></tr>
</table>

5.2.1.2 Class Inclusion and Class Taxonomies

rdfs:subClassOf	The subclass of one class has an extension that is the subset of the extension of the subsetted class.
A subClassOf B	$A \subseteq B$ $\forall x(A(x) \rightarrow B(x))$
	 Fig. 4 UML Lite offers the generalization relation to express class inclusion. Fig. 4 shows a class B being the subclass of a class A.

	`Context A inv:`
	`self.oclIsKindOf(B)`

5.2.1.3 Class Equivalence

owl:equivalentClass	The meaning of this class axiom is that the two class descriptions involved have the same class extension (i.e., both class extensions contain exactly the same set of individuals) [32].
A equivalentClass B	$A = B$ $\forall x(A(x) \rightarrow B(x) \wedge B(x) \rightarrow A(x))$
	 Fig. 5 A correct and legal way to state that two UML Lite classes are equivalent is using a generalization-set and defining it to be covering as demonstrated in Fig. 5

	`Context A inv:`
	`self.oclIsKindOf(B)`
	`Context B inv:`
	`self.oclIsKindOf(A)`

5.2.1.4 Class Disjointness

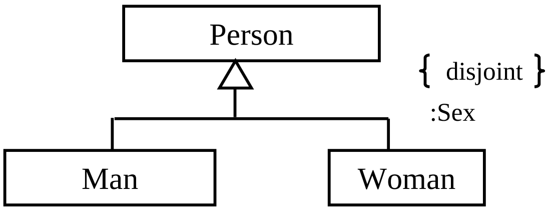
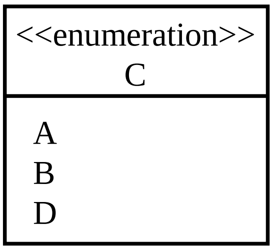
<table>
<tr><td><b>owl:disjointWith</b></td><td colspan="2">The meaning of a owl:disjointWith statement is that the class extensions of the two class descriptions involved have no individuals in common.</td></tr>
<tr><td rowspan="3"><b>A disjointWith B</b></td><td colspan="2">$A \cap B = \varnothing$
$\forall x(A(x) \rightarrow \neg(B(x)) = \neg(\exists x(A(x) \wedge B(x)))$</td></tr>
<tr><td colspan="2">

Fig. 6
Class disjointness in a UML Lite class diagram is only definable between sibling classes in a partitioning, that is, between classes of the same generalization-set. Fig. 6 shows a partitioning of a class C (Person) into the classes A (Man) and B (Woman), which form one generalization-set under a common discriminator (Sex). The generalization-set has the property “disjoint”, meaning that the two subclasses have no common instances.</td></tr>
<tr><td>Context A inv:
not self.oclIsKindOf(B)
Context B inv:
not self.oclIsKindOf(A)</td><td>In OCL, the definition of class disjointness is not limited to sibling classes in a generalization-set.</td></tr>
</table>

5.2.2 Class Descriptions

Class axioms describe classes using class descriptions (also known as class constructors). The class description as the defined expression of a class axiom is always a simple class name. The defining statement of a class axiom is a class description, which specifies the class extension of an unnamed anonymous class. In the following, we list the different forms of class constructors (for the defining statement) in the languages.

5.2.2.1 Enumeration

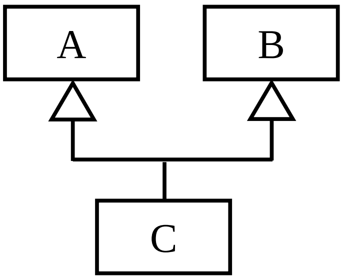
owl:oneOf	An enumeration lists a collection of individuals that constitute the class's extension. The value of this built-in OWL property must be a list of individuals that are the instances of the class [32].	
C oneOf A, B, D	$IE(oneOf) = IE(Class) \times 2^{\Delta}$	
	 Fig. 7	UML Lite supports the concept of enumeration as a data type whose values are enumerated in the model as enumeration literals [17] (cf. Fig. 7). The enumeration is not a class but a data type. This is different from OWL's definition of *owl:oneOf*, as it is only an enumeration of data values. OWL also supports enumerated data types with the same construct.

	`Context X inv:`
	`self.G = C::A`
		OCL has no means to define enumeration but it can use the values (members) of an enumeration in an OCL statement. The syntax for this is the enumeration name followed by two colons and the identifier of the enumeration value.

5.2.2.2 Class Conjunction

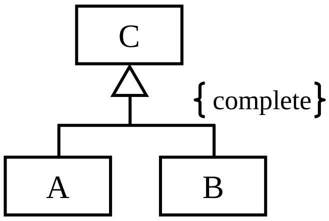
owl:intersectionOf	The owl:intersectionOf property links a class to a list of class descriptions. An owl:intersectionOf statement describes a class for which the class extension contains precisely those individuals that are members of the class extension of all class descriptions in the list [32].
C intersectionOf **A and B**	$C = A \cap B$ $\forall x((C(x) \to A(x) \wedge B(x)) \wedge (A(x) \wedge B(x) \to C(x)))$
	UML Lite does not support a visual construct to define a class as an intersection of several other classes.

	`Context A, B, C inv:`
	`C.allInstances() = A.allInstances()->`
	`intersection(B.allInstances())`

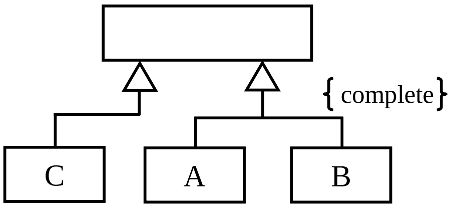
C subClassOf **intersectionOf** **A and B**	$C \subseteq A \cap B$ $\forall x(C(x) \to A(x) \wedge B(x))$
	 Fig. 8 UML Lite allows multiple inheritance, which indicates that a class is an immediate subclass of several other classes at the same time. An instance of that class (C in Fig. 8) is therefore an indirect instance of the superclasses (A and B in Fig. 8). The element has to obey the necessary conditions of all superclasses and is therefore an element of the intersection of all superclasses.

	`Context C inv:`
	`self.oclIsKindOf(A) and self.oclIsKindOf(B)`

5.2.2.3 Class Disjunction

owl:unionOf	The owl:unionOf property links a class to a list of class descriptions. An owl:unionOf statement describes an anonymous class for which the class extension contains those individuals that occur in at least one of the class extensions of the class descriptions in the list [32].
C unionOf A and B	$C = A \cup B$ $\forall x((C(x) \rightarrow A(x) \vee B(x)) \wedge (A(x) \vee B(x) \rightarrow C(x)))$
	 Fig. 9 The generalization-set concept in UML Lite is capable of defining a class as the union of a set of subclasses. The generalization-set attribute "isCovering" is used to describe an exhaustive partitioning of a class by its subclasses. Fig. 9 depicts the situation in a UML class diagram.

	`Context A, B, C inv:`
	`C.allInstances() = A.allInstances()->`
	`union(B.allInstances())`

C subClassOf unionOf A and B	$C \subseteq A \cup B$ $\forall x(C(x) \rightarrow A(x) \vee B(x))$
	 Fig. 10 A direct mapping of this situation is not feasible in a UML Lite class diagram. However, it can be simulated by the use of an anonymous class that represents the union as in Fig. 10.

	`Context A,B,C inv:`
	`A.allInstances()->union(B.allInstances())->`
	`includesAll(C.allInstances())`

5.2.2.4 Class Negation

owl:complementOf	An owl:complementOf property links a class to precisely one class description. An owl:complementOf statement describes a class for which the class extension contains exactly those individuals that do not belong to the class extension of the class description that is the object of the statement [32].
A complementOf B	$A = \neg B$ $IE(A) = \Delta - IE(B)$
	UML Lite is not capable of specifying a class as the complement of another class.

	`Context A, B, OclAny inv:`
	`A.allInstances() = (OclAny.allInstances() -`
	`B.allInstances())`

A subClassOf complementOf B	$A \subseteq \neg B$ $IE(A) \subseteq \Delta - IE(B)$
	The definition of a class as a primitive class that is a subclass of an anonymous class, which is a complement of another class, is not possible in UML Lite.

	`Context A, B, OclAny inv:`
	`(OclAny.allInstances() - B.allInstances())->`
	`includesAll(A.allInstances())`

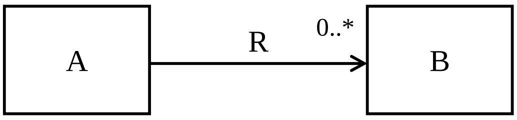
5.2.2.5 Property Restriction – Value Constraints

It is possible to constrain the range of a property in specific contexts in a variety of ways using property restrictions. A property restriction is a special kind of class description. It describes an anonymous class, namely a class of all individuals that satisfy the restriction [32]. This anonymous class is used to define other classes by inclusion or equivalence assertion. This is necessary in OWL, as variables are not supported. The UML interprets association and attribute definitions as necessary conditions. Sufficient conditions and explicit closure axioms can be defined with OCL invariants.

5.2.2.5.1 Value Restriction

owl:allValuesFrom	The value constraint owl:allValuesFrom is a built-in OWL property that links a restriction class to either a class description or a data range [32]. The constraint is analogous to the universal quantifier of predicate calculus. All instances of the described class that have the property must be related to an instance of the class description or a value in the data range.
A equivalentClass Restriction on R allValuesFrom B	$\forall o,y(B(y) \wedge R(o,y) \to A(o)) \wedge \forall x,y(A(x) \wedge R(x,y) \to B(y))$
	UML Lite is not able to specify a sufficient condition on a class feature in a class diagram.

	OCL uses the universal class to state that all objects that have only property values from B for the property R are members of class A. This is the sufficient condition of the OWL axiom. The necessary part is equal to the constraint described below, so that the complete invariant is the set of both assertions.
	`Context o:OclAny inv:`
	`o.R->forAll(y|y.oclIsKindOf(B)) implies`
	`o.oclIsKindOf(A)`
	`Context x:A inv:`
	`x.R->forAll(y|y.oclIsKindOf(B))`

A subClassOf Restriction on R allValuesFrom B	$\forall x,y(A(x) \to (R(x,y) \to B(y))) \Leftrightarrow \forall x,y(A(x) \wedge R(x,y) \to B(y))$
	 Fig. 11 Under the CWA, every class description is assumed to be complete. The counterpart to the OWL statement above is therefore the definition of a typical UML Lite association with 0..* multiplicity as depicted in Fig. 11. Due to the CWA, the UML Lite diagram raises further implicit constraints that are disregarded.

	`Context x:A inv:`
	`x.R->forAll(y|y.oclIsKindOf(B))`

5.2.2.5.2 Existential Quantification

owl:someValuesFrom	The value constraint owl:someValuesFrom is a built-in OWL property that links a restriction class to a class description or a data range. A restriction containing an owl:someValuesFrom constraint describes a class of all individuals for which at least one value of the property concerned is an instance of the class description or a data value in the data range [32].
A equivalentClass Restriction on R someValuesFrom B	$\forall o(\exists y(B(y) \wedge R(o,y) \rightarrow A(o))) \wedge \forall x(A(x) \rightarrow \exists y(R(x,y) \wedge B(y)))$
	UML Lite is not able to specify a sufficient condition on a class feature in a class diagram.

	OCL uses the universal class to state that all objects that have some property values from B for the property R are members of class A. This is the sufficient condition of the OWL axiom. The necessary part is equal to the constraint described below, so that the complete invariant is the set of both assertions.
	`Context o:Thing inv:`
	`o.R->exists(y|y.oclIsKindOf(B)) implies o.oclIsKindOf(A)`
	`Context x:A inv:`
	`x.R->exists(y|y.oclIsKindOf(B))`

A subClassOf Restriction on R someValuesFrom B	$\forall x(A(x) \rightarrow \exists y(R(x,y) \wedge B(y)))$
	Because of the CWA, every property is by default restricted by an implicit closure axiom. This renders a mere existential quantification in the UML Lite impossible.

	In UML Full, an OCL constraint can refine an inherited association R:
	`Context x:A inv:`
	`x.R->exists(y|y.oclIsKindOf(B))`

5.2.2.5.3 Role Filler

owl:hasValue	The value constraint owl:hasValue is a built-in OWL property that links a restriction class to a value V which can be either an individual or a data value. A restriction containing an owl:hasValue constraint describes the class of all individuals for which the property concerned has at least one value semantically equal to V (it may have other values as well) [32].
A equivalentClass Restriction on R hasValue V	$(\forall x(\exists y(R(x,y) \wedge y = V) \rightarrow A(x))) \wedge (\forall x(A(x) \rightarrow \exists y(R(x,y) \wedge y = V)))$
	UML Lite is not able to specify a sufficient condition on a class feature in a class diagram.

	Since OCL cannot directly access object model elements, UML Full is not capable of supporting the role filler construct completely. Nevertheless, OCL can state the value restriction on a data value (V) if A is a class and R is an association of A:
	`Context x:A inv:`
	`x.R->includes(V)`
	`Context o:OclAny inv:`
	`o.R->includes(V) implies o.oclIsKindOf(A)`

A subClassOf Restriction on R hasValue V	$\forall x(A(x) \rightarrow \exists y(R(x,y) \wedge y = V))$
	UML Lite is not able to specify such a value constraint in a class diagram.

	OCL cannot directly access object model elements so UML Full is not capable of supporting the role filler construct completely. Nevertheless, OCL can state the value restriction on a data value (V) if A is a class and R is an association of A:
	`Context x:A inv:`
	`x.R->includes(V)`

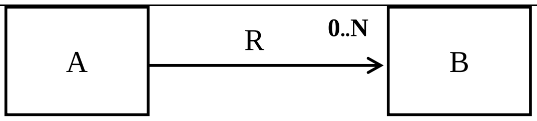
5.2.2.6 Property Restriction – Cardinality Constraints

In general, it is assumed that any instance of a class may have an arbitrary number of values for a particular property. The existential quantification of the last section states that a property is required. It is similar to a qualified number restriction with minimum cardinality of one. Other cardinality constraints can restrain the maximum number of values for a property. Cardinality constraints are analogous to the unqualified number of restrictions of predicate calculus.

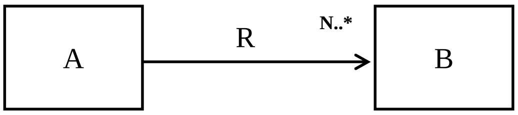
5.2.2.6.1 Maximum Cardinality

owl:maxCardinality	The cardinality constraint owl:maxCardinality is a built-in OWL property that links a restriction class to a data value belonging to the value space of the XML Schema datatype nonNegativeInteger. A restriction containing an owl:maxCardinality constraint describes a class of all individuals that have at most N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint [32].
A equivalentClass Restriction on R maxCardinality N	$\forall x(A(x) \rightarrow \lvert\{\mathrm{y} \mid \mathrm{R(x,y)}\}\rvert \le N) \wedge \forall x(\lvert\{\mathrm{y} \mid \mathrm{R(x,y)}\}\rvert \le N \rightarrow A(x))$
	UML Lite is not able to specify a sufficient condition on a class feature in a class diagram.

	`Context o:OclAny inv:`
	`o.R->size() <= N implies o.oclIsKindOf(A)`
	`Context x:A inv: x.R->size() <= N`

A subClassOf Restriction on R maxCardinality N	$\forall x(A(x) \rightarrow \lvert\{\mathrm{y} \mid \mathrm{R(x,y)}\}\rvert \le N)$
	 Fig. 12 UML Lite specifies cardinality constraints with multiplicity information on the association ends. To emulate an OWL maximum cardinality constraint, the navigable end of the association is given a value for the upper multiplicity attribute that is equal to the maxCardinality value of the OWL statement. Fig. 12 shows an example of an association from class A to class B with maximum cardinality of N.
	`Context x:A inv: x.R->size() <= N`

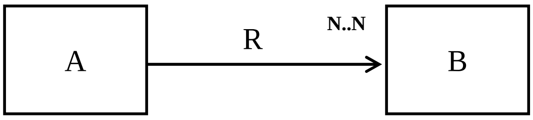
5.2.2.6.2 Minimum Cardinality

<table>
<tr><td><b>owl:minCardinality</b></td><td colspan="2">The cardinality constraint owl:minCardinality is a built-in OWL property that links a restriction class to a data value belonging to the value space of the XML Schema datatype nonNegativeInteger. A restriction containing an owl:minCardinality constraint describes a class of all individuals that have at least N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint [32].</td></tr>
<tr><td rowspan="3"><b>A equivalentClass Restriction on R minCardinality N</b></td><td colspan="2">$\forall x(A(x) \rightarrow \lvert\{\mathrm{y} \mid \mathrm{R(x,y)}\}\rvert \geq N) \wedge \forall x(\lvert\{\mathrm{y} \mid \mathrm{R(x,y)}\}\rvert \geq N \rightarrow A(x))$</td></tr>
<tr><td colspan="2">UML Lite is not able to specify a sufficient condition on a class feature in a class diagram.</td></tr>
<tr><td colspan="2"><code>Context o:OclAny inv:</code>
<code>o.R->size() >= N implies o.oclIsKindOf(A)</code>
<code>Context x:A inv: x.R->size() >= N</code></td></tr>
<tr><td rowspan="3"><b>A subClassOf Restriction on R minCardinality N</b></td><td colspan="2">$\forall x(A(x) \rightarrow \lvert\{\mathrm{y} \mid \mathrm{R(x,y)}\}\rvert \geq N)$</td></tr>
<tr><td>
Fig. 13</td><td>Fig. 13 shows an example of an association from class A to class B with minimum cardinality of N.</td></tr>
<tr><td colspan="2"><code>Context x:A inv: x.R->size() >= N</code></td></tr>
</table>

5.2.2.6.3 Exact Cardinality

owl:cardinality	The cardinality constraint owl:cardinality is a built-in OWL property that links a restriction class to a data value belonging to the range of the XML Schema datatype nonNegativeInteger. A restriction containing an owl:cardinality constraint describes a class of all individuals that have exactly N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint [32].	
A equivalentClass Restriction on R cardinality N	$\forall x(A(x) \rightarrow \lvert\{\mathrm{y} \mid \mathrm{R(x,y)}\}\rvert = N) \wedge \forall x(\lvert\{\mathrm{y} \mid \mathrm{R(x,y)}\}\rvert = N \rightarrow A(x))$	
	UML Lite is not able to specify a sufficient condition on a class feature in a class diagram.	

	`Context o:OclAny inv:`
	`o.R->size() = N implies o.oclIsKindOf(A)`
	`Context x:A inv: x.R->size() = N`	

A subClassOf Restriction on R cardinality N	$\forall x(A(x) \rightarrow \lvert\{\mathrm{y} \mid \mathrm{R(x,y)}\}\rvert = N)$	
	 Fig. 14	Fig. 14 shows an example of an association from class A to class B with cardinality of N.
	`Context x:A inv: x.R->size() = N`	

5.3 OWL Properties and UML Associations and Attributes

An OWL property is a binary relation that asserts general facts about classes of objects and specific facts about an individual. Two types of properties are distinguished (in OWL DL): datatype properties and object properties. The former relate individuals and data values. Object properties link instances of two classes. UML associations are n-ary relations between classifiers. Their instances are called links. Links relate single objects in the object model. A UML attribute or property construct is a structural feature of an object that is contained in every instance of a class and takes an individual value in each instance. The individual value of an attribute is related to the object in a slot. In the UML, a class is usually related to datatypes via attributes. However, both UML associations and attributes can relate a class with another class or a datatype. The difference between the two primitives is on the one hand that only associations can relate multiple classifiers, and on the other hand, that attributes (UML properties) are not first-class modeling primitives. The distinction between attributes and associations in UML is based on the implementation-oriented foundation of the language. For the representation-oriented viewpoint, the UML Property construct is redundant. UML attributes are contingent on the classifier they describe. They cannot form taxonomies and have no decontextualized interpretation (see Section 4.3). UML attributes can therefore not represent OWL properties. The OWL Property construct therefore has to be mapped to a unidirectional binary association in the UML. OWL properties are defined and characterized by property axioms. These axioms are the property name (atomic property) and type (datatype or object), the domain and range, the property inclusion (taxonomy), the property equivalence, the inverse, global cardinality features, and logical property characteristics.

5.3.1 Atomic Properties

<owl:ObjectProperty rdf:ID="R"/>	$R \subseteq OD\ x\ OD$	Datatype and object properties are mapped to binary, unidirectional UML associations. This association has one navigable association-end, which is an attribute of the class that owns it. However, without a universal class in UML Lite an unspecified property cannot be represented. As described in Section 4.3, an unspecified association, because of the CWA, by default applies to no object, i.e. the domain and range is the bottom concept.
	This defines a property R with the restriction that its values should be individuals.	
<owl:DatatypeProperty rdf:ID="R"/>	$R \subseteq OD\ x\ DD$	
	This defines a property R with the restriction that its values should be data values.	

5.3.2 Domain & Range

rdfs:domain	For a property one can define (multiple) rdfs:domain axioms [32]. Syntactically, rdfs:domain is a built-in property that links a property to a class description, which could be any of the class descriptions described in 5.2 The domain of a property specifies the set of objects that can be related to other values with the property. It asserts that the subjects of such property statements must belong to the class extension of the indicated class description.	
R domain C	$\forall x \exists y (R(x,y) \rightarrow C(x))$	
	UML Lite has no universal classifier and because of the CWA, cannot define an association domain separately.	

	`Context x:OclAny inv:`
	`x.R->notEmpty() implies`
	`x.oclIsKindOf(C)`
		UML Full uses OclAny to relate to all objects in the object space. Therefore, the variable x is interpreted as an arbitrary object.

rdfs:range	For a property one can define (multiple) rdfs:range axioms [32]. Syntactically, rdfs:range is a built-in property that links a property to either a data range or a class description, which could be any of the class descriptions described in 5.2 The range of a property specifies the set of objects or data values that can be related to from other objects with the property. It asserts that the values of this property must belong to the class extension of the class description or to data values in the specified data range.	
R range C	$\forall x, y (R(x,y) \rightarrow C(y))$	
	UML Lite has no universal classifier and because of the CWA, cannot define an association range separately.	

	`Context x,y:OclAny inv:`
	`x.R->notEmpty() implies`
	`R->forAll(y|y.oclIsKindOf(C))`
		UML Full uses OclAny to relate to all objects in the object space. Therefore, the variable x is interpreted as an arbitrary object.

5.3.3 Property Inclusion and Property Taxonomies

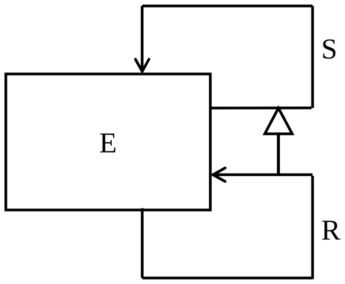
<table>
<tr><td><b>rdfs:subPropertyOf</b></td><td colspan="2">A property can be defined to be a specialization (subproperty) of an existing property. The subproperty of a property has an extension that is the subset of the extension of the subsetted property. The contexts and type of the properties or associations must be compatible.</td></tr>
<tr><td rowspan="3"><b>R subPropertyOf S</b></td><td colspan="2">$R \subseteq S$
$\forall x, y(R(x, y) \rightarrow S(x, y))$</td></tr>
<tr><td>

Fig. 15</td><td>UML Lite offers the generalization relation to express association taxonomies. However, UML Lite has no universal classifier and can therefore not define an association or sub-association with unspecified domain and range values. If these are given elsewhere, a sub-association can be defined as in Fig. 15 with "E" being an arbitrary class.</td></tr>
<tr><td colspan="2">OCL is not equipped to support constraints in the context of associations and thus there are no sub-association definitions.</td></tr>
</table>

5.3.4 Property Equivalence

<table>
<tr><td><b>owl:equivalentProperty</b></td><td>The owl:equivalentProperty construct can be used to state that two properties have the same property extension [32] - that is both property extensions contain the same set of pairs or tuples.</td></tr>
<tr><td rowspan="3"><b>R equivalentProperty S</b></td><td>$R = S$
$\forall x, y(R(x, y) \rightarrow S(x, y) \wedge S(x, y) \rightarrow R(x, y))$</td></tr>
<tr><td>UML Lite class diagram has no feature to express association equivalence.</td></tr>
<tr><td>OCL is not equipped to support constraints in the context of associations and thus no association equivalence definitions.</td></tr>
</table>

5.3.5 Inverse Property Relation

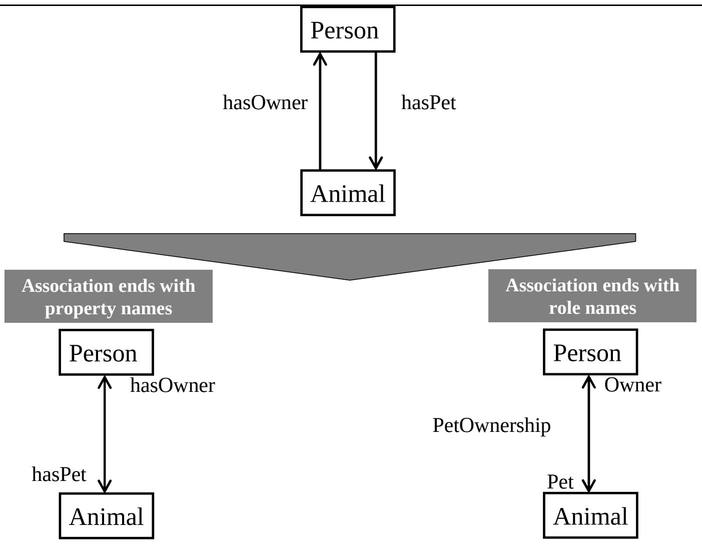
<table>
<tr><td><b>owl:inverseOf</b></td><td>The owl:inverseOf construct can be used to define an inverse relation between object properties [32]. Properties are unidirectional, that is, their direction goes from domain to range. To state that a certain property is an inverse of another property the inverseOf relation can be used.</td></tr>
<tr><td rowspan="4"><b>R inverseOf S</b></td><td>$\forall x, y, a, b(R(x, y) \to S(y, x) \wedge S(a, b) \to R(b, a))$</td></tr>
<tr><td>

Fig. 16</td></tr>
<tr><td>UML Lite has no feature to express that two unidirectional associations are inverses of each other. UML Lite instead uses bidirectional associations to combine two inverse, binary and unidirectional associations. The association then has two navigable ends, where one is the opposite of the other <b>Fig. 16</b>. To be able to distinguish the semantics of the relation it is mandatory to define role names for the association ends or to add a direction marker on the association name. <b>Fig. 16</b> shows how two (binary, unidirectional,) inverse OWL properties are mapped to one bidirectional UML association by applying the property names to the association ends.</td></tr>
<tr><td>OCL can navigate along associations to manage cycles in a model. Two inverse associations form a cycle that can be characterized by the following constraint:
<code>Context x,a:OclAny inv:</code>
<code>x.R->forAll(y| y.S = x) and</code>
<code>a.S->forAll(b| b.R = a)</code></td></tr>
</table>

5.3.6 Functional Property

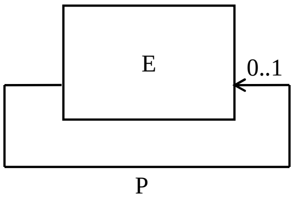
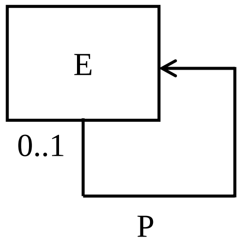
<table>
<tr><td rowspan="1"><b>owl:FunctionalProperty</b></td><td colspan="2">The owl:FunctionalProperty construct can be used to define a global cardinality constraint on a property. A functional property P(x,y) is a property that can have only one (unique) value y for each instance x, i.e., there cannot be two distinct values y1 and y2 such that the pairs (x,y1) and (x,y2) are both instances of this property [32].</td></tr>
<tr><td rowspan="3"><b>FunctionalProperty P</b></td><td colspan="2">$\forall x, a, b(P(x,a) \wedge P(x,b) \rightarrow b = a)$</td></tr>
<tr><td>

Fig. 17</td><td>In UML Lite class diagrams, it is not possible to declare an association as being a functional relation. As the functional property construct is just a short form for a maximum cardinality constraint, it is possible to define an association as functional by specifying the upper multiplicity of the navigable end as being one (cf. Fig. 17).</td></tr>
<tr><td colspan="2">In UML Full an OCL constraint can be used to define an association functional for all objects:
<code>Context x: OclAny inv: x.P->size() <= 1</code></td></tr>
</table>

5.3.7 Inverse Functional Property

<table>
<tr><td>**owl:InverseFunctionalProperty**</td><td colspan="2">The owl:InverseFunctionalProperty construct can be used to define a global cardinality constraint on an object property. If a property is declared to be inverse-functional, then the object of a property statement uniquely determines the subject (some individual) [32].</td></tr>
<tr><td rowspan="3">**InverseFunctionalProperty P**</td><td colspan="2">$\forall x,a,b(P(a,x) \wedge P(b,x) \rightarrow a = b)$</td></tr>
<tr><td>
Fig. 18</td><td>In a UML Lite class diagram it is possible to define an association as inverse functional by specifying the upper multiplicity of the non-navigable end or the opposite end as being one (cf. Fig. 18).</td></tr>
<tr><td colspan="2">In UML Full an OCL constraint can be used to define an association functional for all objects:
`Context x,a,b: OclAny inv:`
`a.P->includes(x) and b.P->includes(x)`
`implies a=b`</td></tr>
</table>

5.3.8 Transitive Property

<table>
<tr><td>**owl:TransitiveProperty**</td><td>The owl:TransitiveProperty construct can be used to define the global logical characteristics of an object property. If a transitive property P has a pair (x,y) as an instance of P and a pair (y,z) is also an instance of P, then we can infer the pair (x,z) is also an instance of P [32].</td></tr>
<tr><td rowspan="3">**TransitiveProperty P**</td><td>$\forall x,y,z(P(x,y) \wedge P(y,z) \rightarrow P(x,z))$</td></tr>
<tr><td>In UML Lite, it is not possible to define that an association has transitive characteristics.</td></tr>
<tr><td>In UML Full an OCL constraint can be used to define an association transitive for all objects:
`Context x: OclAny inv:`
`x.P->includesAll(x.P.P)`</td></tr>
</table>

5.3.9 Symmetric Property

owl:SymmetricProperty	The owl:SymmetricProperty construct can be used to define the global logical characteristics of an object property. A symmetric property is a property for which it holds that if the pair (x,y) is an instance of P, then the pair (y,x) is also an instance of P [32].
SymmetricProperty P	$\forall x, y(P(x, y) \rightarrow P(y, x))$
	In UML Lite, it is not possible to define that an association has symmetric characteristics.
	In UML Full an OCL constraint can be used to define an association symmetric for all objects: `Context x: OclAny inv: x.P.P->includes(x)`

5.4 UML specific constructs

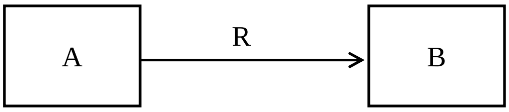
UML supports the specification of behavioral features. Some UML features that are often regarded as structural are in fact "behavioral-driven" such as abstract classes, interfaces, active classes, composite structure, ports, connectors and collaborations. These constructs either specify which messages an object understands or emits, or how objects are related by message communication. As explained in the introduction, behavioral features are not of interest for this comparison as OWL is a structure representation language.

Furthermore, UML includes several design or implementation related constructs that have no representational meaning in terms of the modeled domain. These constructs are templates, dependencies, derived elements, static attributes, ordered features, and association ends (UML Property). These constructs are based on the construction-oriented perspective of the UML, which means they are used in the design and implementation phases of software engineering and have a well-defined denotation in the models used in those development phases. However, the subject they describe is the software product that tries to represent the system described in the problem (or business) domain model. Therefore, they provide no additional use for knowledge representation.

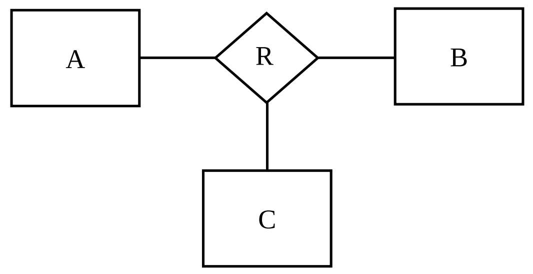
However, UML does possess some representation-oriented constructs that are not directly available in OWL. These shorthand constructs are n-ary associations, qualified associations, bidirectional associations, and association classes. The following subsections present these constructs and their OWL counterparts.

Two UML constructs – aggregation and composition – are not translatable to OWL, as it does not feature predefined mereological relationship constructs in the knowledge representation ontology. However, this does not preclude a general ontology like [39], [40] from providing the required constructs. The regular association construct in UML is slightly different from the OWL property construct as the latter can be unspecified, meaning that the domain or range or both are implicitly defined to be the universal concept. As all UML relationships are based on the association construct, we include the standard definition for the matter of completeness.

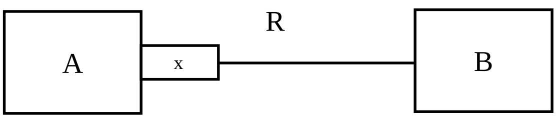
5.4.1 UML Associations

	**R domain A range B**	A regular UML association is equivalent to an OWL object property that has a specified domain and range.

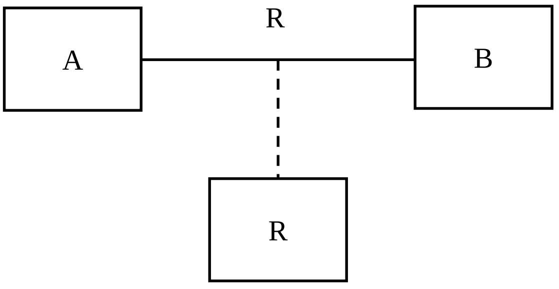
5.4.2 UML N-ary Associations

	A common solution to representing n-ary relations in OWL is reification, i.e., the creation of an individual, which stands for an instance of the relation and relates the things that are involved in that instance of the relation. However, the exact design of the relation class and the other related classes depends on the association characteristics. For further reading, see [34].

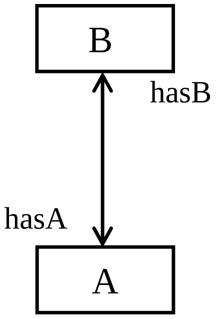
5.4.3 UML Qualified Associations

	A qualifier declares a partition of the set of associated instances with respect to an instance at the qualified end [17]. As the qualifier is a property of the qualified class, a qualified association is a kind of ternary relation and can be treated like n-ary associations (see 4.4.2).

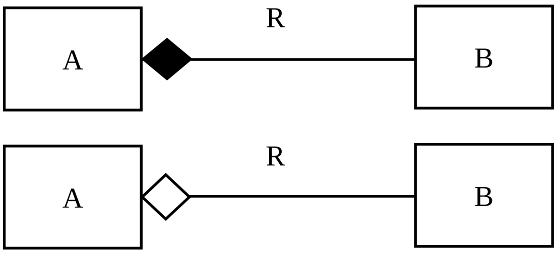
5.4.4 UML Association Classes

	An association may be refined to have its own set of features; that is, features that do not belong to any of the connected classifiers but rather to the association itself. Such an association is called an association class [17]. An association class is treated in a similar way to an n-ary association as a class reifying the relation in OWL.

5.4.5 UML Bidirectional Associations

	A bidirectional association can be translated into a pair of OWL properties, with one being the inverse of the other (cf. 4.3.5).

5.4.6 UML Aggregation and Composition Relationships

	The UML supports the mereological relationships of aggregation and composition. Both constructs have no counterpart in OWL.

5.5 Ontology & Model

Both languages use a packaging mechanism to gather all ontology or model information in one place. UML uses the *Model* construct, which is a subclass of the *Package* primitive to gather all model elements of one knowledge representation in one module. OWL uses the *Ontology* construct for the same purpose. However, neither language separates terminological knowledge from assertional knowledge (see Section 3). A model or an ontology can thus be interpreted as the set of all classes, properties, data types and individuals of a universe of discourse.

Informally:

Ontology = Class $\cup$ Property $\cup$ Datatype $\cup$ Object

IE is the interpretation function, Δ is the semantic domain, OD is the object domain, DD is the datatype domain, and Model is a UML model and Ontology is an OWL ontology:

$$IE(Model) = IE(Ontology) = 2^{\Delta} \cup 2^{\Delta\times\Delta} \cup \Delta = 2^{OD} \cup 2^{DD} \cup 2^{OD\times OD} \cup 2^{OD\times DD} \cup OD \cup DD$$

5.6 Data types

In both languages data types are distinguished from object types. While the object types represent the semantic domain of objects, data types represent the orthogonal semantic domain of data values (cf. Section 5.2). Data values have a kind of type-referential semantics as they have no identity for themselves. OWL makes use of the RDF datatyping scheme [36] and uses XML Schema data types (XSD) [35]. Data values are instances of the RDF Schema class rdfs:Literal. Literals can be either plain (no datatype) or typed. Datatypes are instances of the class rdfs:Datatype [32]. The simple built-in XML Schema data types that are recommended by the RDF Semantics document [37] for use in OWL are: the primitive datatype String, the primitive datatype Boolean, the primitive numerical datatypes Decimal, Float, Double, the primitive time-related datatypes DateTime, Time, Date, and the primitive datatypes HexBinary, Base64Binary, and AnyURI.

The UML supports the predefined primitive data types String, Integer, Boolean, and UnlimitedNatural that are defined in the PrimitiveTypes package of the AuxiliaryConstructs package [17]. The OCL additionally provides the type Real [22]. It is also open to any kind of user-defined data types.

The data types of UML correspond to equally named OWL data types. In other words, UML String is equal to XSD String, UML Boolean is equal to XSD Boolean and UML Integer is equal to XSD Integer. UnlimitedNatural contains the set of naturals and the infinity value ("*"). It is used as the type for the upper bounds of multiplicities in the metamodel. OWL does not need it because it is equal to omitting a value for owl:maxCardinality.

5.7 Version Information & Other Non-Ontology Constructs

OWL provides additional constructs to handle ontology merging and evolution. Version information can be included via the owl:versionInfo construct. Further constructs concerning ontology evolution like owl:deprecatedClass, owl:deprecatedProperty, owl:backwardCompatibleWith, owl:incompatibleWith, and owl:priorVersion are similar to special UML dependency relations and stereotypes which are widely used for UML model evolution. The UML uses the package merge mechanism and construct [17] to describe model evolution, which is similar to the import construct in OWL.

5.8 OWL Full Constructs & Axioms

Until now we have only considered OWL DL. OWL Full extends OWL DL by providing unconstrained use of RDF constructs. The consequence is that the construct *owl:Class* is equivalent to *rdfs:Class* and OWL Full allows classes to be treated as individuals because *rdfs:Class* is a subclass of *rdfs:Resource*. As *rdfs:Literal* is an instance of *rdfs:Class* this implies that data types and object types are not distinguished. The universe of individuals therefore consists of all resources (*owl:Thing* is equivalent to *rdfs:Resource*), which also means that object properties and datatype properties are not disjoint [32]. It is therefore legal in OWL Full to define a class "Dog" that is at the same time an individual as illustrated in Fig. 19 showing two instances of "Dog" namely "Fido" and "Lassie" and "Species" that is a classifier of "Dog".

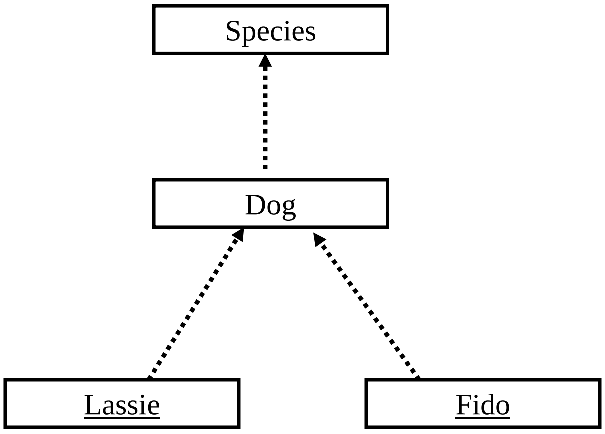


Fig. 19

In UML, this scenario can only be captured by using the power type concept. In this case, a further class would be needed (e.g. "Animal") that would be partitioned by the metaclass "Species" and the superclass of "Dog". A direct specification as in OWL Full is not possible in UML Full.

OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. The following OWL fragment show an example ontology that extends the OWL concept *owl:class* by defining a subclass *SingletonClass*. The problem is that the semantics of this class is only implicit, while the interpretation of all other language constructs is explicitly defined in the language semantics [38].

```
<owl:Class rdf:ID=”SingletonClass”>
 <owl:subClassOf rdf:resource=”#Class”/>
</owl:Class>
```

The UML supports language-modeling capabilities by its profile extension mechanism. A profile defines a set of stereotypes that are subclasses of specified classes in the UML metamodel. This means that the standard UML Full, without the profile feature, has no support for language modeling, just like OWL DL.

6 Conclusion

In this paper, we have presented a detailed and comprehensive comparison of the UML and OWL languages based on a systematic analysis of their respective features and their underlying assumptions. The results of this comparison can be interpreted in two ways depending on what one wishes to emphasize. They can either be viewed as highlighting the great disparity between the languages or the great similarity between them.

Since both languages were devised for the purpose of knowledge representation and have common roots in object-orientation, their abstract syntaxes show a remarkable degree of similarity. On the other hand, each language has a slightly different purpose that manifests itself in several ways. Firstly, it directly affects the set of language constructs offered by each language. OWL restricts itself to a set of inference-supporting constructs together with a few constructs that handle Web-related issues. The UML, in contrast, is not concerned with automatic deducibility of information and offers an unrestricted constraint language. It additionally focuses on describing implementation related issues that are irrelevant for knowledge representation in the abstract. Secondly, it influences the set of assumptions, – many of them implicit – used to interpret the elements of the language. The different assumptions used with OWL and UML often lead to different interpretations of common language constructs. In other words,, while they share a large set of apparently common language constructs, these constructs possess slightly different meanings with slightly different mappings in the semantic domain. As a result, it is impossible to translate OWL ontologies into UML models and vice versa without the loss or corruption of information.

Since they have common roots and share a large set of common concepts there is a lot of scope for aligning and possibly merging them. We have explained how the different interpretations of common concepts are based on implicit underlying assumptions, which could be made explicit and modified. Also, missing concepts can easily be added, so that models and ontologies cover the same semantic domains. Therefore, one could adapt one or both of the languages to include the necessary missing concepts and require modelers to specify the perspective (i.e. interpretation assumptions) they wish to use. With the use of perspective-dependent transformation rules (that have to be defined), one could then translate an OWL ontology into a UML model. There is no fundamental reason why OWL's features for explicit information representation could not be used under a closed-world assumption or why the features of UML could not be used under an open-world assumption. All that would be needed is an additional set of features to specify exactly what assumptions and closure conditions hold at a particular point in time. In other words, all implicit (default) information should be made explicit. In fact, even UML Full already has most of the capabilities needed due to the flexibility of OCL. For OWL, this can be achieved for example as suggested in [33], by providing a special “ontology-wide” property that allows the world assumption to be defined.

As noted above, the (explicit) abstract syntaxes of both languages already have a large overlap in terms of common constructs. However, there are a few concepts in one that are missing in the other due to the different requirements in each domain. The missing features in OWL (cf. Section 5.4) can be compensated for by defining specific higher ontologies (e.g. [40], [39]), but it is not possible to emulate the missing OWL constructs in the UML using additional libraries as these are more fundamental concepts. The features of OWL which are either not supported or only partly supported in UML are object enumeration, role fillers, atomic properties, property inclusion, property equivalence, synonym and antonym object specification, classes as instances and the universal class "Thing". The role of the latter, however, can be assumed by "OclAny" in our comparison as explained in Section 5.2. Although this is a legal choice based on extensional equivalence as both concepts represent the set of all objects in the UoD, it is not correct based on intentional equality since both concepts have different purposes and intensional definitions.

In many ways, the situation facing the “information” representation communities today is similar to that which faced the different object-oriented modeling communities a decade and a half ago when the UML was first defined. At that time users were faced with a large array of different languages possessing trivial abstract syntax and semantic differences that greatly increased learning overheads, fragmented the OOAD tool market, and raised interoperability barriers while providing little if any benefit. Much of the initial enthusiasm around the UML came from its removal of these arbitrary idiosyncrasies. A somewhat similar situation exists today with the division between the "ontology" and "modeling" worlds. There are a lot of arbitrary syntactic and semantic differences which create unnecessary learning and interoperability barriers between the two communities. Given the large shared common core of UML and OWL there is much to be gained from unifying the two, as the UML did with OOAD languages. Other modeling paradigms which could also be brought into such a unification effort include Entity-Relationship modeling and the Common Warehouse Model. Discussing the pros and cons of unification was not a goal of this comparison, however. Rather, the goal was to clarify at a technical level where the two languages overlap and where they differ. We hope this comparison will provide a useful foundation for ensuing discussions.

REFERENCES

[1] M.K. Smith, C. Welty, and D.L. McGuinness, eds., “OWL Web Ontology Language Guide”, *World Wide Web Consortium (W3C) recommendation*, Feb. 2004,

http://www.w3.org/TR/2004/REC-owl-guide-20040210/.

[2] L. Hart et al., “OWL Full and UML 2.0 Compared”, *OMG TFC Report*, 2004.

[3] A. Borgida and R.J. Brachman, "Conceptual Modelling with Description Logics," *The Description Logic Handbook*, F. Baader et al., eds., Cambridge University Press, 2002, pp 349-372.

[4] C. Atkinson and T. Kühne, "Rearchitecting the UML infrastructure," *ACM Transactions on Modeling and Computer Simulation*, vol. 12, 2002, pp. 290-321.

[5] J. Álvarez, A. Evans and P. Sammut, "Mapping between Levels in the Metamodel Architecture," *Proc. 4th Int'l Conf. Unified Modeling Language (UML2001),* M. Gogolla, C. Kobryn, eds., Springer-Verlag, 2001, pp. 34-46.

[6] C. Atkinson and T. Kühne, "Model-Driven Development: A Metamodeling Foundation," *IEEE Software,* vol. 20, 2003, pp. 36-41.

[7] Colin Atkinson and Thomas Kühne, "The Essence of Multilevel Metamodeling," *Proc. 4th Int'l Conf. Unified Modeling Language (UML2001),* M. Gogolla, C. Kobryn, eds., Springer-Verlag, 2001, pp. 19-33.

[8] C. Atkinson and T. Kühne, "Meta-Level Independent Modeling," *Int'l Workshop on Model Eng. at the 14th European Conf. on Object-Oriented Programming (ECOOP 2000),* 2000.

[9] C. Atkinson, "Supporting and Applying the UML Conceptual Framework," *Proc. 1st Int'l Conf. Unified Modeling Language 1998 (UML'98) - Beyond the Notation. First Int'l Workshop,* J. Bézivin and P.-A. Muller, eds., 1998, pp. 1-11.

[10] K. Kiko, "Towards a Unified Knowledge Representation Framework," master’s thesis, Dept. Software Technology, Univ. of Mannheim, Germany, 2005.

[11] K. Baclawski et al., "Extending the Unified Modeling Language for ontology development," *Software and System Modeling,* vol. 1, 2002, pp. 142-156.

[12] D. Djuric, "MDA-based Ontology Infrastructure," *Computer Science Information Systems (ComSIS)*, vol. 1, no. 1, 2004.

[13] S. Brockmans, R. Volz, A. Eberhart, and P. Löffler, "Visual Modeling of OWL DL Ontologies Using UML," *Int'l Semantic Web Conference*, 2004, pp. 198-213.

[14] S. Cranefield and M. Purvis, "UML as an Ontology Modelling Language," *Proc. 16th Int'l Joint Conf. Artificial Intelligence (IJCAI-99) Workshop on Intelligent Information Integration*, 1999.

[15] S. Cranefield, "UML and the Semantic Web," *Proc. 1st Int'l Semantic Web Working Symposium (SWWS'01), July 2001*, pages 113-130.

[16] Object Management Group, DSTC, IBM, Sandpiper Software Inc., "Ontology Definition Metamodel Second Revised Submission," *OMG Specification*, May 2005.

[17] Object Management Group, "UML 2.0 Superstructure Revised Final Adopted Specification," *OMG Specification*, Oct. 2004.

[18] Object Management Group, "Meta Object Facility 2.0 Core Final Adopted Specification," *OMG Specification*, Oct. 2003.

[19] Object Management Group, "Common Warehouse Metamodel Specification," *OMG Specification*, Feb. 2001.

[20] Object Management Group, "UML 2.0 Infrastructure Final Adopted Specification," *OMG Specification*, Dec. 2003.

[21] Object Management Group, "OCL 2.0 Adopted Specification," *OMG Specification*, Oct. 2003.

[22] Object Management Group, "OCL 2.0 Specification," *OMG Specification*, June 2005.

[23] A. Cali et al., "A Formal Framework for Reasoning on UML Class Diagrams," *Proc. 13th Int'l Symposium Foundations of Intelligent Systems (ISMIS 2002)*, M.-S. Hacid et al., eds., LNCS 2366, Springer-Verlag, 2002, pp. 503-512.

[24] D. Harel and B. Rumpe, “Modeling Languages: Syntax, Semantics and All That Stuff - Part I: The Basic Stuff”, *tech. report MCS00-16*, Faculty of Mathematics and Computer Science, The Weizmann Inst. of Science, Israel, 2000.

[25] J.Z. Pan and I. Horrocks, "Metamodeling Architectures of Web Ontology Languages," *Proc. 1st Int'l Semantic Web Working Symposium (SWWS 2001)*, July 2001.

[26] J.Z. Pan and I. Horrocks, "RDFS(FA) and RDF MT: Two Semantics for RDFS," *Proc. 2nd International Semantic Web Conference (ISWC2003).* D. Fensel and K. Sycara and J. Mylopoulos, eds., 2003, pp. 30-46.

[27] J.F. Sowa, “Knowledge Representation - Logical, Philosophical, and Computational Foundations”, Pacific Grove, Brooks/Cole, 2000.

[28] K. Orsvaern, "The REVISE Project: A Purpose Driven Method for Language Comparison," *Proc. 8th European Knowledge Acquisition Workshop (EKAW'96),* N. Shadbolt, K. O'Hara and A.T. Schreiber, eds., LNAI 1076, Springer-Verlag, 1996, pp. 66-81.

[29] E. Söderström et al., "Towards a Framework for Comparing Process Modelling Languages," *Proc. 14th Int'l Conf. Advanced Information Systems Engineering, (CAiSE2002),* A. Banks Pidduck et al., eds., LNCS 2348, Springer-Verlag, 2002, pp. 600-611.

[30] P. van Emde Boas et al., eds., "Formalizing UML: Mission Impossible?," *Proc. OOPSLA'98 Workshop on Formalizing UML*, 1998.

[31] T.A. Halpin and J.L. Han Oei, “A Framework for Comparing Conceptual Modelling Languages,” *tech. report 92-29*, Dept. Information Systems, Univ. of Nijmegen, The Netherlands, 1992.

[32] M. Dean and G. Schreiber, eds., “OWL Web Ontology Language Reference,” *World Wide Web Consortium (W3C) recommendation*, Feb. 2004; http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[33] J. Heflin, ed., “OWL Web Ontology Language Use Cases and Requirements,” *World Wide*

Web Consortium (W3C) recommendation, Feb. 2004; http://www.w3.org/TR/2004/ RECwebont-req-20040210/.

[34] N. Noy and A. Rector, “Defining N-ary Relations on the Semantic Web: Use With Individuals,” *World Wide Web Consortium (W3C) working draft,* July 2004; http://www.w3.org/ TR/swbp-n-aryRelations/.

[35] P.V. Biron and A. Malhotra, eds., “XML Schema Part 2: Datatypes,” *World Wide Web Consortium (W3C) recommendation*, May 2001.

[36] G. Klyne and J.J. Carroll, eds., “Resource Description Framework (RDF): Concepts and Abstract Syntax,” *World Wide Web Consortium (W3C) recommendation,* Feb. 2004; http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[37] P. Hayes, ed., “RDF Semantics,” *World Wide Web Consortium (W3C) recommendation*, Feb. 2004; http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.

[38] P.F. Patel-Schneider, P. Hayes and I. Horrocks, eds., “OWL Web Ontology Language Semantics and Abstract Syntax,” *World Wide Web Consortium (W3C) recommendation*, Feb. 2004; http://www.w3.org/TR/2004/REC-owl-semantics-20040210/.

[39] K. Falkovych, M. Sabou and H. Stuckenschmidt, "UML for the Semantic Web: Transformation-Based Approaches," B. Omelayenko and M.C.A. Klein, eds., *Knowledge Transformation for the Semantic Web*, vol. 95, IOS Press, 2003, pp. 92-106.

[40] W.N. Borst, “Construction of Engineering Ontologies,” *tech. report*, Center for Telematica and Information Technology, Univ. of Tweenty, Enschede, The Netherlands.

[41] Raymond Reiter, "A logic for default reasoning", *Artificial Intelligence,* Apr. 1980, vol. 13 no.1-2 pp. 81-132.

[42] M.K. Smith, ed., “Web Ontology Issue Status,” *World Wide Web Consortium (W3C) working draft*, Nov. 2003; http://www.w3.org/2001/sw/WebOnt/webont-issues.html.

[43] P.P. Chen, " The Entity-Relationship Model - Toward a Unified View of Data," ACM Trans. Database Systems, Mar. 1976, vol. 1, no. 1, pp. 9-36.

[44] K. Kiko and C. Atkinson, " Integrating Enterprise Information Representation Languages," *Proc. of Int'l EDOC Workshop on Vocabularies, Ontologies and Rules for the Enterprise*, G. Guizzardi and G. Wagner, eds., CTIT, Sep. 2005, pp. 41-50.

[45] D. Brickley and R. V. Guha eds., "RDF Vocabulary Description Language 1.0: RDF Schema," *World Wide Web Consortium (W3C) recommendation*, 10 February 2004; http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[46] A. Tarski, "Logic, Semantics, Mathematics: Papers from 1923 to 1938," Oxford University Press, 1956.

[47] Object Management Group, "XML Metadata Interchange, v.2.1," *OMG Specification*, Sep. 2005.

[48] D. Beckett, ed., "RDF/XML Syntax Specification (Revised)," *World Wide Web Consortium (W3C) recommendation*, 10 February 2004; http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.