Ontological Modeling: Part 7

Terry Halpin
LogicBlox and INTI International University

This is the seventh in a series of articles on ontology-based approaches to modeling. The main focus is on popular ontology languages proposed for the Semantic Web, such as the Resource Description Framework (RDF), RDF Schema (RDFS), and the Web Ontology Language (OWL). OWL is based on description logic. A later series of articles will explore other logic-based languages such as datalog. The first article [2] introduced ontologies and the Semantic Web, and covered basic concepts in the Resource Description Framework (RDF), contrasting them with other data modeling approaches. The second article [3] discussed the N3 notation for RDF, and covered the basics of RDF Schema. The third article [4] provided further coverage of RDFS, and introduced different flavors of the Web Ontology language (OWL). The fourth article [5] discussed basic features of OWL, mainly using Manchester syntax. The fifth article [6] discussed OWL taxonomy, comparison operators for classes, data types and predicates, and examined inverses, functional roles and keys in more depth. The sixth article [7] covered cardinality restrictions in OWL 2. The current article discusses how the union, intersection, and complement operators are handled in OWL 2.

Union

In set theory, given any sets *A* and *B*, the *union* of *A* and *B* is the set of all elements in either *A* or *B* or both. That is, $A \cup B = \{x \mid x \in A \vee x \in B\}$, where "$\cup$" is the union operator and "$\vee$" is the inclusive-or operator. The Venn diagram in Figure 1(a) depicts the union of *A* and *B* by shading in the region covering both the circles that depict the sets. If we union $A \cup B$ with set *C*, we obtain $(A \cup B) \cup C$, as pictured by the shaded region in Figure 1(b). The union operation is associative, so the order in which we associate the operands by bracketing doesn't matter, i.e. $(A \cup B) \cup C = A \cup (B \cup C)$. This allows us to drop the brackets, rendering the union of *A*, *B* and *C* simply as $A \cup B \cup C$.

Although the union operator is usually written as a binary operator in infix position, it may also be written as an *n*-ary operator ($n \geq 2$) in prefix position with its operands listed in parentheses, e.g. $\cup(A, B, C)$, since this can be trivially rewritten in binary form. The union operation is also commutative, i.e. $A \cup B = B \cup A$. So the order in which we list the operands doesn't matter. This applies also to the *n*-ary prefix notation, e.g. $\cup(A, B, C, D) = \cup(D, A, C, B)$.

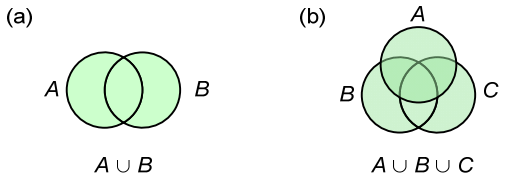

Figure 1 Venn diagrams for (a) the union of two sets, and (b) the union of three sets.

The union operator is supported in all versions of OWL, except for OWL Lite, and may be applied to classes or data ranges (e.g. datatypes or other collections of literals). The complex class expression or data range expression resulting from a union may be used anywhere a simple class or data range may be used. The union of two or more classes contains all the individuals that occur in at least one of those classes. In Manchester syntax, the union operator appears as "*or*", placed in infix position. Turtle syntax instead uses the *owl:unionOf* predicate, placed in prefix position with its *n* operands ($n \geq 2$) in parentheses.

Table 1 illustrates a case where an inclusive union is intended. Any student who is either bright or industrious (or both) is classified as a promising student.

Table 1 Defining PromisingStudent as the union of BrightStudent and IndustriousStudent

Manchester Syntax	*Turtle Syntax*
Class: PromisingStudent EquivalentTo: BrightStudent or IndustriousStudent	:PromisingStudent a owl:Class; owl:unionOf (:BrightStudent :IndustriousStudent).

Figure 2(a) shows an equivalent model for the Table 1 example in Object-Role Modeling (ORM) [1]. Inclusion relationships from subtype to supertype are shown as arrows, and the circled dot depicts the inclusive-or operator. In ORM, as in OWL, subtypes are overlapping by default, so no more needs to be said. Figure 1(b) shows an equivalent class diagram in the Unified Modeling Language (UML) [8]. The union constraint is depicted by the annotation "complete". In UML, subclasses are disjoint by default, so the annotation "overlapping" needs to be explicitly asserted.

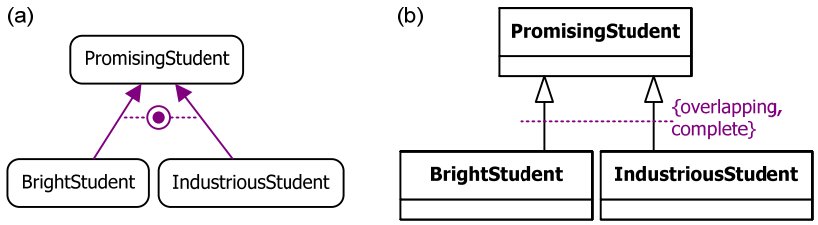

Figure 2 Inclusive-or subtyping model for Table 1 in (a) ORM and (b) UML.

Given the OWL declarations in Table 1, inferences may be drawn about membership in the superclass. For example, given the following statements (Manchester syntax on the left, Turtle syntax on the right))

Individual: Harry Types: BrightSudent	:Harry a :BrightStudent
Individual: Ron Types: IndustriousSudent	:Ron a :IndustriousStudent
Individual: Hermione Types: BrightStudent, IndustriousSudent	:Hermione a :BrightStudent, IndustriousSudent

we may infer

Individual: Harry Types: PromisingSudent	:Harry a :PromisingStudent
Individual: Ron Types: PromisingSudent	:Ron a :PromisingStudent
Individual: Hermione Types: PromisingSudent	:Hermione a :PromisingStudent

A class *partition* may be declared in OWL by adding a *disjoint classes* declaration to the union operation. In Manchester syntax, to declare that two or more classes that are disjoint (i.e. mutually exclusive) simply prepend "*DisjointClasses:* " to the class list. In Turtle, use the *owl:disjointWith* predicate. For example, in Table 2, a Party is either a person or an organization, but not both.

Table 2 Defining Party as a disjoint union of Person and Organization

Manchester Syntax	*Turtle Syntax*
DisjointClasses: Person, Organization Class: Party EquivalentTo: Person or Organization	:Person owl:disjointWith :Organization. :Party a owl:Class; owl:unionOf(:Person :Organization).

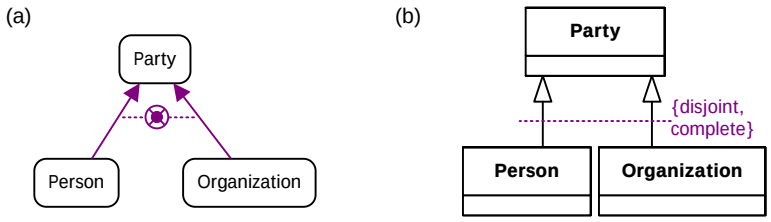

Figure 3 Exclusive-or subtyping model for Table 2 in (a) ORM and (b) UML.

Figure 3(a) models the Table 2 example in ORM, using an exclusive-or constraint (depicted by a crossed dot), to partition Party into Person and Organization. Figure 3(b) models the same example in UML, using the annotations "disjoint" and "complete".

Table 3 extends the example by defining the class Party as a disjoint union of three classes. The Manchester syntax extends seamlessly to lists of three or more disjoint classes, but the Turtle syntax is much more complex, as shown. The corresponding ORM and UML models are shown in Figure 4.

Table 3 Defining Party as a disjoint union of Person, PrivateCompany, and PublicOrganization

Manchester Syntax	*Turtle Syntax*
DisjointClasses: Person, PrivateCompany, PublicOrganization Class: Party EquivalentTo: Person or PrivateCompany or Organization	[] a owl:AllDisjointClasses; owl:members (:Person :PrivateCompany :PublicOrganization). :Party a owl:Class; owl:unionOf (:Person :PrivateCompany :PublicOrganization).

As a shorter alternative for declaring partitions, OWL supports the *disjoint union* operator. In Manchester syntax, this is rendered by using "*DisjointUnionOf: "*. For instance, the partition just discussed may be declared more concisely as shown below.

Class: Party
DisjointUnionOf: Person, PrivateCompany, PublicOrganization

In Turtle, the same partition may be rendered by using the *owl:disjointUnionOf* predicate to denote the disjoint union operator, as shown below. This is much simpler than the alternative Turtle syntax used previously.

:Party owl:disjointUnionOf (:Person :PrivateCompany :PublicOrganization).

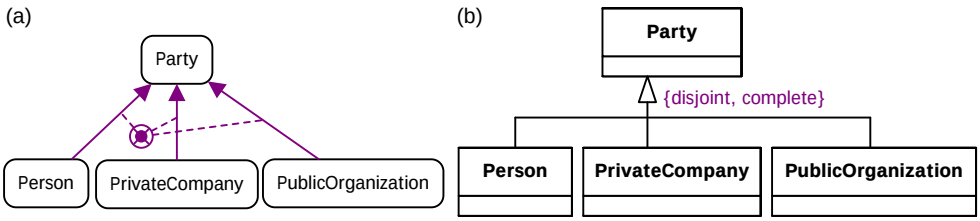

Figure 4 Exclusive-or subtyping model for Table 3 in (a) ORM and (b) UML.

Intersection

The *intersection* of sets *A* and *B* is the set of all elements that occur in both *A* and *B*. Using "∩" for the intersection operator and "&" for the logical conjunction operator "and", this definition may be written thus: $A \cap B = \{x \mid x \in A \ \& \ x \in B\}$. The Venn diagrams in Figure 5 depict the intersection of sets by shading the region where the circles that depict the sets overlap. Like union, the intersection operator is both associative and commutative, and an *n*-ary prefix notation may be used to abbreviate repeated applications of the binary infix operator. For example, the intersection of *A*, *B* and *C* may be rendered either as $A \cap B \cap C$ or as $\cap(A, B, C)$.

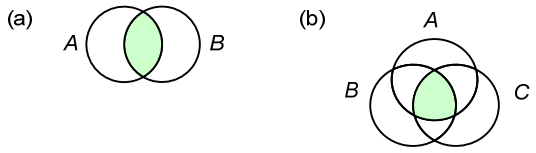

Figure 5 Venn diagrams for (a) the intersection of two sets, and (b) the intersection of three sets.

In OWL, the intersection operator may be applied to classes or data ranges, and the complex class expression or data range expression resulting from an intersection may be used anywhere a simple class or data range may be used. The intersection of two or more classes contains each individual that occurs in all those classes. In Manchester syntax, the intersection operator appears as "*and*", placed in infix position. Turtle syntax instead uses the *owl:intersectionOf* predicate, placed in prefix position with its *n* operands ($n \geq 2$) in parentheses.

Table 4 provides a simple example where the class StudentEmployee is defined as the intersection of Student and Employee, which are themselves subclasses of Person.

Table 4 Defining StudentEmployee as the intersection of Student and Employee

Manchester Syntax	*Turtle Syntax*

Class: Person
Class: Student
  SubClassOf: Person
Class: Employee
  SubClassOf: Person
Class StudentEmployee
  EquivalentTo: Student and Employee
	:Person a owl:Class.
	:Student rdfs:subClassOf :Person.
	:Employee rdfs:subClassOf :Person.
	:StudentEmployee a owl:Class;
	  owl:intersectionOf (:Student :Employee).

Figure 6 models this example in ORM and UML. This is a case of multiple inheritance, since StudentEmployee is a subtype of both Student and Employee.

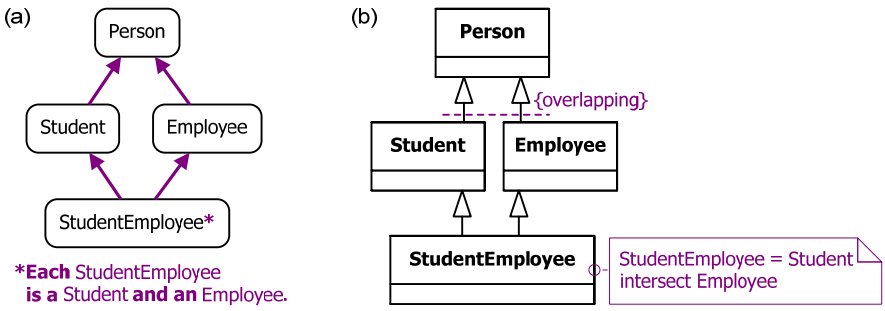

Figure 6 Modeling student employees in (a) ORM and (b) UML.

The asterisk in Figure 6(a) indicates that the subtype StudentEmployee is derived, rather than simply asserted. In this case, ORM requires a subtype definition. Although the definition may be written as a quantified conditional, as shown in Figure 6(a), it is understood as a quantified biconditional, so that each object that is both a student and an employee must be a student employee. An alternative syntax renders the subtype definition thus: Define StudentEmployee as Student who is an Employee. UML has no high level language for expressing subclass definitions, so this has been added as an informal note in Figure 6(b). If desired, a constraint for this could be formally specified in UML using the Object Constraint Language (OCL) [9], but OCL expressions are often too cryptic for validation by nontechnical business users.

Contrast this example with the one shown in Table 5, which is based on an example from the official OWL 2 Primer [10, pp. 28-29]. Here the class Grandfather is simply asserted, not defined. Instead it is simply constrained to be included in (rather than being identical to) the intersection of the Man and Parent classes. Not all men who are parents have to be grandparents.

Table 5 Constraining Grandfather to be included in the intersection of Man and Parent

Manchester Syntax	*Turtle Syntax*
Class: Grandfather SubClassOf: Man and Parent	:Grandfather rdfs:subClassOf [rdf:type owl:Class ; owl:intersectionOf (:Man :Parent)] .

Figure 7 models this example in ORM and UML. In the ORM model, the absence of an asterisk indicates that the subtypes are simply asserted. Hence no definitions are provided for them. In practice, it would typically be much better to model Grandparent as a derived subtype, basing its definition on fact types such as Person is of Gender and Person is a grandparent of Person (which itself may typically be derived from Person is a parent of Person).

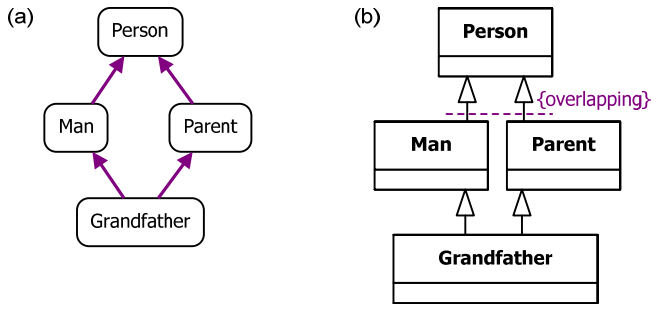

Figure 7 Modeling subtyping constraints on Grandfather in (a) ORM and (b) UML.

Complement

In set theory, A', the *complement* of set A, is the set of all elements that are in the universal set U but not in the set A. Using "~" for the logical negation operator "not", we have $A' = \{x \mid \sim x \in A\}$. In OWL, the class of all individuals is called owl:Thing. We may visualize complements as the shaded regions in Figure 8.

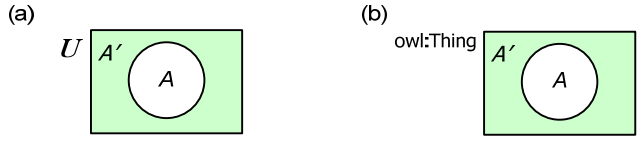

Figure 8 Picturing A' as (a) the complement of a set A, and (b) the complement of an OWL class A.

In OWL, the complement operator may be applied to classes or data ranges, and the complex class expression or data range expression resulting may be used anywhere a simple class or data range may be used. The complement of a class contains each individual that is not a member of that class. In Manchester syntax, the complement operator appears as "*not*". Turtle syntax instead uses the *owl:complementOf* predicate. Table 7 provides a simple example where the class NonHuman is defined as the complement of the class Human.

Table 6 Declaring in OWL that the class NonHuman is the complement of Human

Manchester Syntax	*Turtle Syntax*

Class: Human
Class: NonHuman
  EquivalentTo: not Human
	:Human a owl:Class.
	:NonHuman a owl:Class;
	  Owl:complementOf :Human.

It is rare in data modeling to introduce Thing as an entity type or class. However, if we do this, we can model the Table 6 example in ORM and UML as shown in Figure 9.

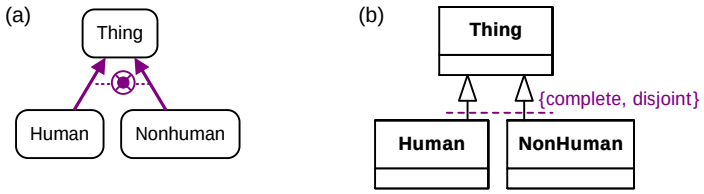

Figure 9 One way to model the Table 6 example in (a) ORM and (b) UML.

As a more typical example, Table 7 shows how to declare non-drivers as persons who are not drivers. The class Driver is also defined using a cardinality restriction, as discussed in the previous article [7]. Notice how much simpler the Manchester syntax is compared with Turtle.

Table 7 Defining Driver and NonDriver in OWL

Manchester Syntax	*Turtle Syntax*

Class: Person
Class: Driver
  EquivalentTo: Person and drives min 1 Car
Class: NonDriver
  EquivalentTo: Person and not Driver
	:Person a owl:Class.
	:Driver owl:equivalentClass
	  [a owl:Restriction;
	  owl:onProperty :drives;
	  owl:minQualifiedCardinality 1;
	  owl:onClass :Car].
	:NonDriver a owl:Class;
	  owl:intersectionOf (:Person
	  [owl:complementOf :Driver]).

Figure 10 shows ORM and UML models for this case. In the ORM model, the subtypes are derived, so formal subtype definitions are provided. In the UML model, the subclass definitions are captured informally in notes.

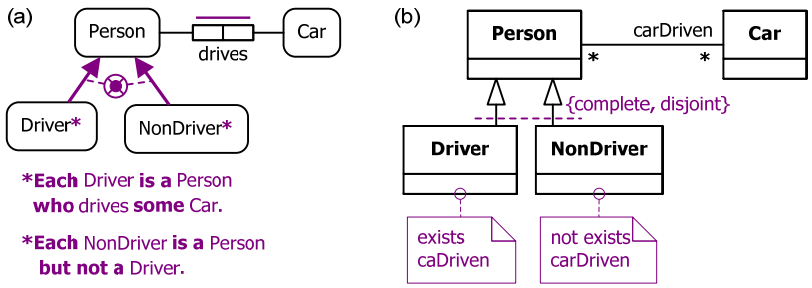

Figure 10 Modeling the Table 7 example in (a) ORM and (b) UML.

All the examples so far have used classes. In OWL 2, the union, intersection and complement operations may also be applied to data ranges. Table 8 provides some examples.

Table 8 Applying union, intersection and complement operators to data ranges

Manchester Syntax	*Turtle Syntax*
Datatype: StringOrInteger EquivalentTo: xsd:string or xsd:integer	:StringOrInteger a rdfs:Datatype; owl:unionOf (xsd:string xsd:integer).
Datatype: PassGradeNr EquivalentTo: GradeNr and not FailGradeNr	:PassGradeNr a rdfs:Datatype; owl:intersectionOf (:GradeNr [owl:datatypeComplementOf :FailGradeNr]).

Conclusion

The current article provided a detailed coverage of the union, intersection and complement operations in OWL 2. The next article will explore some other features of OWL 2, such as ring constraints and enumerated types.

References

1. Halpin, T. & Morgan, T. 2008, *Information Modeling and Relational Databases, 2nd edition*, Morgan Kaufmann, San Francisco.
2. Halpin, T. 2009, 'Ontological Modeling: Part 1', *Business Rules Journal*, Vol. 10, No. 9 (Sep. 2009), URL: http://www.BRCommunity.com/a2009/b496.html.
3. Halpin, T. 2009, 'Ontological Modeling: Part 2', *Business Rules Journal*, Vol. 10, No. 12 (Dec. 2009), URL: http://www.BRCommunity.com/a2009/b513.html.
4. Halpin, T. 2010, 'Ontological Modeling: Part 3', *Business Rules Journal*, Vol. 11, No. 3 (March 2010), URL: http://www.BRCommunity.com/a2010/b527.html.
5. Halpin, T. 2010, 'Ontological Modeling: Part 4', *Business Rules Journal*, Vol. 11, No. 6 (June 2010), URL: http://www.BRCommunity.com/a2010/b539.html.
6. Halpin, T. 2010, 'Ontological Modeling: Part 5', *Business Rules Journal*, Vol. 11, No. 12 (Dec. 2010), URL: http://www.BRCommunity.com/a2010/b570.html.
7. Halpin, T. 2011, 'Ontological Modeling: Part 6', *Business Rules Journal*, Vol. 12, No. 2 (Feb., 2011), URL: http://www.BRCommunity.com/a2011/b579.html.
8. Object Management Group 2003, *UML 2.0 Superstructure Specification*. Available online at: www.omg.org/uml.
9. Object Management Group 2005, *UML OCL 2.0 Specification*. Available online at: http://www.omg.org/docs/ptc/05-06-06.pdf.
10. W3C 2009, 'OWL 2 Web Ontology Language: Primer', URL: http://www.w3.org/TR/owl2-primer/.

11. W3C 2009, ‘OWL 2 Web Ontology Language: Direct Semantics’, URL: http://www.w3.org/TR/owl2-direct-semantics/.
12. W3C 2009, ‘OWL 2 Web Ontology Language Manchester Syntax’, URL: http://www.w3.org/TR/owl2-manchester-syntax/.
13. W3C 2009, ‘OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax’, URL: http://www.w3.org/TR/owl2-syntax/.