Ontological Modeling: Part 8

Terry Halpin
LogicBlox and INTI International University

This is the eighth 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 seventh article [8] discussed the union, intersection, and complement operators in OWL 2. The current article explores support for ring constraints within OWL 2.

Ring Constraints

Recall that in OWL, all relationships are binary, so facts are expressed as subject-predicate-object sentences. The set of instances from which the subjects are drawn is the *domain* of the predicate, and the set of instances from which the objects are drawn is the *range* of the predicate. In OWL, binary predicates are called *properties*. *Object properties* relate entities to entities (e.g. :Einstein :wasBornIn :Germany), while *data properties* relate entities to literals (e.g. :Einstein :hasGivenName "Albert"). If the domain and range are the same or at least compatible (overlapping), it is meaningful to compare subject and object instances (e.g. may they be identical?). In Object-Role Modeling (ORM), such predicates are called *ring predicates* (picture a ring formed by navigating from an object type though the predicate and circling back to the object type), and constraints that apply specifically to these kinds of predicates (or more generally, pairs of compatible roles) are called *ring constraints* [1].

ORM includes graphical notation for several kinds of ring constraint, some of which are simply defined (e.g. irreflexive, asymmetric, antisymmetric, intransitive) while others are recursively defined (e.g. acyclic, strongly intransitive). OWL 2 supports only the following five kinds of ring constraint: reflexive, irreflexive, symmetric, asymmetric, and transitive. Of these, OWL 1 supported only symmetric and transitive. Unlike ORM, OWL allows ring constraints to be applied only to object property expressions.

Reflexive Predicates

In mathematics, a relation R on a set A is said to be *reflexive* over its domain if and only if xRx for each element x in A. For example, the relation $\leq$ on the set of real numbers is reflexive, since every real number is less than or equal to itself, and the subsethood relation $\subseteq$ is reflexive, since every set is a subset of itself. A relation R is *globally reflexive* if and only if every individual thing in the domain of discourse bears the relation R to itself (i.e., R is globally reflexive if and only if $\forall x\ xRx$). For example, in a world that includes numbers and sets, the identity relation = is globally reflexive (since everything equals itself), but $\leq$ and $\subseteq$ are not globally reflexive because $\leq$ does not apply to sets and $\subseteq$ does not apply to numbers.

OWL allows object property expressions to be declared globally reflexive by characterizing them as reflexive properties [12, p. 10]. For example, if we restrict the domain of individuals (owl:Thing) to persons, and we agree that each person knows himself/herself, then the "knows" predicate may be declared to be reflexive as shown in Table 1. In Manchester syntax, declare "Reflexive" as a characteristic of the object property. In Turtle syntax, declare the object property as an instance of owl:*ReflexiveProperty*.

Table 1 Declaring *knows* to be globally reflexive

Manchester Syntax	*Turtle Syntax*
ObjectProperty: knows Characteristics: Reflexive	:knows rdf:type owl:ReflexiveProperty.

In OWL, if a predicate is known to be reflexive, then any instance in its domain may be inferred to bear the relationship to itself [14, p. 86]. For example, given the above declarations and the following declarations (Manchester syntax on the left, Turtle syntax on the right)

Individual: Einstein Types: Person	:Einstein rdf:type :Person.

we may infer

Individual: Einstein Facts: knows Einstein	:Einstein :knows :Einstein.

Care is required in using this reflexive property construct, because it implies global reflexivity. For example, if our universe of discourse included not just people but other classes of things (e.g. cars), then the predicate "knows" is *not* reflexive in this global sense (e.g. a car does not know itself). In many object domains, the only predicate likely to be globally reflexive is the identity relation "=".

For such reasons, ORM defines reflexivity in the following local sense. A ring predicate R is *locally reflexive* if and only if $\forall x \forall y(xRy \rightarrow xRx)$. In other words, if any individual bears the relation to something, then it must also bear the relation to itself. Hence in ORM you may declare the predicate "knows" to be locally reflexive even when the universe of discourse includes objects like cars where the predicate does not apply at all.

While OWL does not directly support local reflexivity in this sense, it does include an *ObjectHasSelf* restriction to declare a subset of a predicate's domain where each individual in that subset bears the relation to itself. For example, suppose we declare the predicate "likes" between people. Some but not all people like themselves, so "likes" is not reflexive. However, we may define SelfLiker as a subclass of Person, where each person in SelfLiker does like himself/herself. The relevant Manchester and Turtle syntax for this is shown in Table 2. As you can see, the Manchester syntax is much simpler.

Table 2 Constraining each member of SelfLiker to like itself

Manchester Syntax	*Turtle Syntax*

Class: Person
ObjectProperty: likes
    Domain: Person
    Range: Person
Class: SelfLiker
    EquivalentTo: likes Self
	:Person rdf:type owl:Class.
	:likes rdfs:domain :Person;
	    rdfs:range :Person.
	:SelfLiker owl:equivalentClass [
	    rdf:type owl:Restriction ;
	    owl:onProperty :likes;
	    owl:hasSelf "true"^^xsd:Boolean.].

Irreflexive Predicates

A ring predicate R is *irreflexive* if and only if $\forall x \sim xRx$ (i.e. for each individual x, it is *not* the case that x bears the relation R to itself). So nothing may be in an irreflexive relationship with itself. For example, < is irreflexive because nothing is less than itself, and the parenthood relation is irreflexive because nothing can be a parent of itself. Notice that there is no need to distinguish between global and local senses of irreflexivity because if the relation does not apply at all then the negation requirement is satisfied.

Figure 1(a) shows an ORM diagram of the parenthood fact type, together with a satisfying, sample population. For simplicity, we assume in this model that persons may be identified by their first given name. The spanning uniqueness constraint bar indicates that the predicate is many-to-many. The frequency constraint of "≤ 2" on the child role indicates that a person has at most two parents. The irreflexive nature of the ring predicate is depicted by a ring icon connected by a dotted line to the predicate, with a dot for an object and a stroke through the ring to indicate that the object cannot bear the connected relationship to itself. For example, Ann cannot be a parent of Ann.

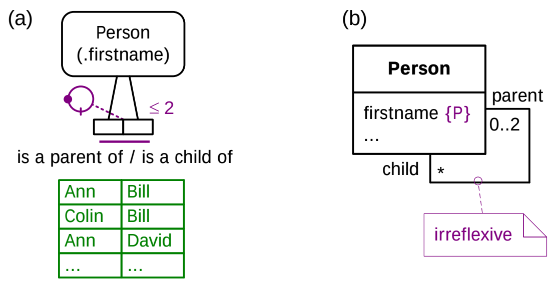

Figure 1 Depicting parenthood as an irreflexive predicate in (a) ORM, and (b) UML.

Figure 1(b) models the same example as a Unified Modeling Language (UML) class diagram [9], omitting the sample population. The ring association line includes elbows to enable the class to be connected at both ends, and has the relevant association role name at each end. The multiplicity constraints of "0..2" and "*" indicate that each person has at most two parents and each person has zero or more children. The "{P}" constraint is a non-standard notation to indicate that firstname provides the preferred identifier for persons. UML has no graphic notation for ring constraints, but the attached note indicates informally that the association is irreflexive. If desired, the irreflexive constraint could be formally specified as a formula in the Object Constraint Language (OCL) [10].

In OWL, the parenthood predicate may be declared to be irreflexive as shown in Table 3. In Manchester syntax, declare "Irreflexive" as a characteristic of the object property. In Turtle syntax, declare the object property as an instance of owl:IrreflexiveProperty.

Table 3 Constraining *isParentOf* to be irreflexive

Manchester Syntax	*Turtle Syntax*

Class: Person
ObjectProperty: isParentOf
    Domain: Person
    Range: Person
    Characteristics: Irreflexive
	:Person rdf:type owl:Class.
	:isParentOf rdfs:domain :Person;
	        rdfs:range :Person.
	:isParentOf rdf:type owl:IrreflexiveProperty.

The main point of declaring a predicate to be irreflexive to is to help prevent bad data being asserted to it. For example, once the above irreflexive constraint has been declared, any attempt to add a fact that somebody is his/her own parent (e.g. :Ann :isParentOf :Ann) will be rejected.

Symmetric Predicates

A ring predicate *R* is *symmetric* if and only if $\forall x \forall y(xRy \rightarrow yRx)$; that is, for each individual *x* and *y* (not necessarily distinct), if *xRy* then it is also the case that *yRx*. In other words, if the relationship applies then its converse also applies. For example, the siblinghood relation is symmetric because if one person is a sibling (brother or sister) of another, then the second person is a sibling of the first.

Figure 2(a) shows an ORM diagram of the siblinghood fact type, together with a satisfying, sample population. The symmetric nature of the ring predicate is depicted by a ring icon connected by a dotted line to the predicate, with dots for the objects, and the top arc representing the left-to-right relationship and the bottom arc representing the right-to-left relationship. For example, if Linda is a sibling of Paul then Paul is a sibling of Linda. Figure 2(b) shows the same example in UML, without the sample data. Here the symmetric constraint is captured informally in a note, but it could also be captured formally in OCL. Siblinghood is both symmetric and irreflexive (nothing is a sibling of itself). Figure 2(c) shows the combined constraint icon for this in ORM, as well as a composite note in UML.

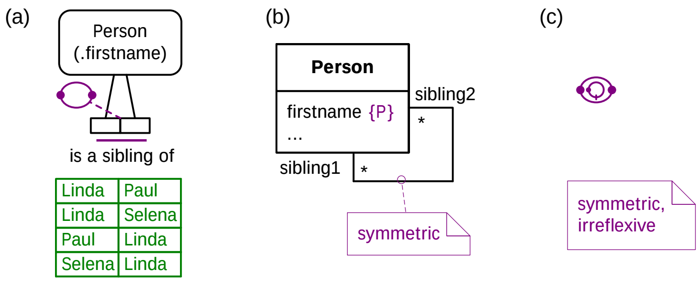

Figure 2 Depicting siblinghood as a symmetric predicate in (a) ORM and (b) UML, and (c) combining constraints.

The siblinghood predicate may be declared to be symmetric in OWL as shown in Table 4. In Manchester syntax, declare "Symmetric" as a characteristic of the object property. In Turtle syntax, declare the object property as an instance of owl:SymmetricProperty. For completeness, the irreflexive nature of the predicate is also declared.

Table 4 Constraining *isSiblingOf* to be symmetric (and irreflexive)

Manchester Syntax	*Turtle Syntax*
Class: Person ObjectProperty: isSiblingOf Domain: Person Range: Person Characteristics: Symmetric, Irreflexive	:Person rdf:type owl:Class. :isSiblingOf rdfs:domain :Person; rdfs:range :Person. :isSiblingOf rdf:type owl:SymmetricProperty, owl:IrreflexiveProperty.

In OWL, if a predicate is declared to be symmetric, and we assert a fact instance of it, then its converse may be inferred. For example, given the above declarations and the following declarations (Manchester syntax on the left, Turtle syntax on the right)

Individual: Romulus Facts: isSiblingOf Remus	:Romulus :isSiblingOf :Remus.

we may infer

Individual: Remus Facts: isSiblingOf Romulus	:Remus :isSiblingOf :Romulus.

There is a subtle difference between symmetric constraints in ORM and OWL. In ORM, constraining a predicate to be symmetric means that the asserted fact population must be symmetric. For example, if we assert that Romulus is a sibling of Remus then we must also assert that Remus is a sibling of Romulus (both facts are stored). In OWL however, you can assert just one of these facts and have the other fact inferred. To do that in ORM, you would provide a derivation rule for inferring the converse rather than constraining the fact type to be symmetric.

For example, in ORM you could declare the fact type Person is a sibling of Person to be *semiderived*, meaning that some of its instances may be asserted and some may be derived, and use the following rule to derive the converse facts: Person$_1$ is a sibling of Person$_2$ **if** Person$_2$ is a sibling of Person$_1$. In this case, you would constrain the siblinghood fact type to be irreflexive, but not symmetric.

Asymmetric Predicates

A ring predicate *R* is *asymmetric* if and only if, for each individual *x* and *y* (not necessarily distinct), if *xRy* then it is not the case that *yRx*. In other words, if the relationship applies then its converse cannot apply. For example, the parenthood relation is asymmetric because if one person is a parent of another, then the second person cannot be a parent of the first.

Figure 3(a) shows an ORM diagram of the parenthood fact type, together with a satisfying, sample population. The asymmetric nature of the ring predicate is depicted by a ring icon connected by a dotted line to the predicate, with dots for the objects, and a stroke through the bottom arc. For example, if Ann is a parent of Bill then Bill cannot be a parent of Ann. Figure 3(b) shows the same example in UML, without the sample data. Here the asymmetric constraint is captured informally in a note, but it could also be captured formally in OCL.

Note that if a predicate is asymmetric, it is automatically irreflexive as well (consider the definition of asymmetry for the case where $x = y$). Hence there is no need to add an irreflexive constraint to the models in Figure 3 because it is implied by the asymmetric constraint. Although asymmetry implies irreflexivity, the converse does not apply. For example, isSiblingOf is irreflexive but not asymmetric.

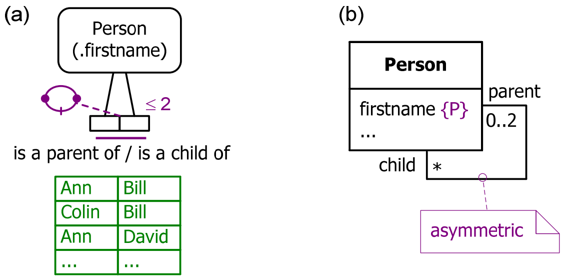

Figure 3 Depicting parenthood as an asymmetric predicate in (a) ORM, and (b) UML.

The parenthood predicate may be declared to be asymmetric in OWL as shown in Table 5. In Manchester syntax, declare "Asymmetric" as a characteristic of the object property. In Turtle syntax, declare the object property as an instance of owl:AsymmetricProperty.

Table 5 Constraining *isParentOf* to be asymmetric

Manchester Syntax	*Turtle Syntax*
Class: Person ObjectProperty: isParentOf Domain: Person Range: Person Characteristics: Asymmetric	:Person rdf:type owl:Class. :isParentOf rdfs:domain :Person; rdfs:range :Person. :isParentOf rdf:type owl:AsymmetricProperty.

The main point of declaring a predicate to be asymmetric is to help prevent bad data being asserted to it. For example, once the above asymmetric constraint has been declared and we assert that Ann is a parent of Bill, then any attempt to add the fact that Bill is a parent of Ann will be rejected.

Transitive Predicates

A ring predicate *R* is *transitive* if and only if, for each individual *x*, *y*, and *z* (not necessarily distinct), if *xRy* and *yRz* then then it is also the case that *xRz*. For example, the ancestorhood relation is transitive because if one person is an ancestor of another, and the second person is an ancestor of a third person, then the first person is an ancestor of the third person. Figure 4(a) shows an ORM diagram of the ancestorhood fact type, together with a sample population. Figure 4(b) shows the same example in UML, without the sample data.

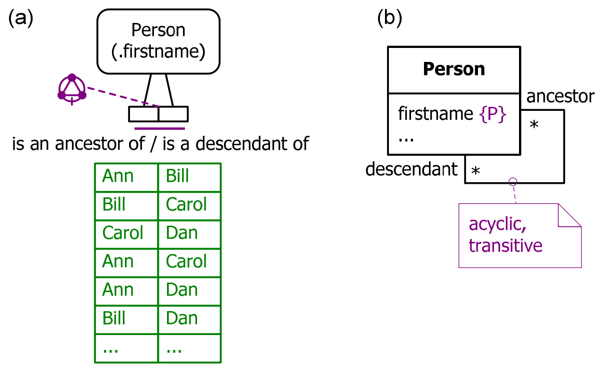

Figure 4 Depicting an asserted ancestorhood predicate that is acyclic and transitive in (a) ORM, and (b) UML.

Ignoring reincarnation, the ancestorhood predicate is actually acyclic (an object can never cycle back to itself by applying one or more ancestorhood facts). Ancestorhood is also asymmetric, but this is implied by acyclicity so there is no need to add that. ORM depicts the acyclic constraint graphically by a ring with three dots and a stroke, and the transitivity constraint by a triangle with dots at each node. These two constraint shapes are orthogonally combined by overlaying into a single icon as shown. UML has no graphical notation for these constraints, so an informal note for them is used in the UML class diagram.

In this model, ancestorhood facts are simply asserted. If instead, ancestorhood facts are always derivable from parenthood facts, then ORM can express this formally by declaring the ancestorhood fact type to be derived from this recursive derivation rule: $Person_1$ is an ancestor of $Person_2$ **if and only if** $Person_1$ is a parent of $Person_2$ **or** $Person_1$ is an ancestor of **some** $Person_3$ **who** is a parent of $Person_2$. Transitivity is now implied by the derivation rule. One reason for deriving transitive predicates rather than asserting them is that their population (or "transitive closure") can quickly become very large. For example, asserting the top three facts in the sample data for Figure 4(a) requires the following four facts shown to be included. As the number of asserted facts increases, the number of facts that are transitively implied increases dramatically.

Although OWL does not support acyclicity constraints, it does allow ring predicates to be declared to be transitive. In OWL the ancestorhood predicate may be declared to be transitive as shown in Table 6. In Manchester syntax, declare "Transitive" as a characteristic of the object property. In Turtle syntax, declare the object property as an instance of owl:TransitiveProperty. For completeness, the asymmetric constraint is also declared.

Table 6 Constraining *isAncestorOf* to be transitive (and asymmetric)

Manchester Syntax	*Turtle Syntax*
Class: Person ObjectProperty: isAncestorOf Domain: Person Range: Person Characteristics: Transitive, Asymmetric	:Person rdf:type owl:Class. :isAncestorOf rdfs:domain :Person; rdfs:range :Person. :isAncestorOf rdf:type owl:TransitiveProperty, owl:AsymmetricProperty.

In OWL, declaring the isAncestorOf predicate to be transitive does not require that all its instances must be asserted. For example, if you declare the predicate to be transitive, and assert that Ann is an ancestor of Bill and that Bill is an ancestor of Chris, then the OWL engine can infer that Ann is an ancestor of Chris. You can also arrange things this way in ORM by declaring the ancestorhood fact type to be semiderived, and supplying the following derivation rule: $Person_1$ is an ancestor of $Person_2$ **if** $Person_1$ is an ancestor of **some** $Person_3$ **who** is an ancestor of $Person_2$. However, if parenthood facts are available to derive ancestry, then the earlier derivation rule discussed is clearly preferable.

Conclusion

The current article briefly discussed the notion of ring constraints on predicates, and then provided a detailed coverage of the five ring constraints that are supported in OWL 2 (reflexive, irreflexive, symmetric, asymmetric, and transitive). The next article will discuss enumerated types and value restrictions on properties in OWL 2.

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. Halpin, T. 2011, 'Ontological Modeling: Part 7', *Business Rules Journal*, Vol. 12, No. ? (?., 2011), URL: http://www.BRCommunity.com/a2011/b???.html. ***Keri please supply
9. Object Management Group 2003, *UML 2.0 Superstructure Specification*. Available online at: www.omg.org/uml.
10. Object Management Group 2005, *UML OCL 2.0 Specification*. Available online at: http://www.omg.org/docs/ptc/05-06-06.pdf.
11. W3C 2009, 'OWL 2 Web Ontology Language: Primer', URL: http://www.w3.org/TR/owl2-primer/.
12. W3C 2009, 'OWL 2 Web Ontology Language: Direct Semantics', URL: http://www.w3.org/TR/owl2-direct-semantics/.
13. W3C 2009, 'OWL 2 Web Ontology Language Manchester Syntax', URL: http://www.w3.org/TR/owl2-manchester-syntax/.
14. W3C 2009, 'OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax', URL: http://www.w3.org/TR/owl2-syntax/.