Formalizing the CRM

Carlo Meghini and Martin Doerr

1 Introduction

This document presents a formalization of the CIDOC CRM in first-order logic. The resulting first-order theory, that we call $\mathcal{CRM}$, captures all aspects of the CRM that are expressed in the specification.

For implementation purposes, the CRM is subsequently expressed as an OWL ontology that we call `CRM`. We use the term "OWL" as a synonym of "OWL 2".

As a notational convention,

- CRM terms are written in Sans Serif, *e.g.* E1 CRM Entity
- first-order symbols in italics, *e.g. E1 CRM Entity*
- OWL entities in teletype style, *e.g.* `E1 CRM Entity`

2 Translation of class specifications

Table 1 shows the translation in first-order logic of the specification of a CRM class. As the first row in Table 1 indicates, a class is modelled as a unary predicate symbol, given by the first part of the class name, that is *E2* represents class E2 Temporal Entity. In the first-order sentences in Table 1, as well as in the ones in the rest of this document, variables are universally quantified unless differently stated.

Translation of sub- or super-class statements is obvious. Class disjointness is discussed below. All other aspects of class specification involve properties, and are discussed in Section 3.

2.1 Class disjointness

Disjointness statements capture incompatibility between classes or properties, making inconsistent any KB in which such classes or properties share a common instance.

CRM Specification	*Translation into first-order logic*
Class E*n c-name*	Unary Predicate *En*
A Superclass of B	$B(x) \supset A(x)$
A Subclass of B	$A(x) \supset B(x)$

Table 1: Translation of class specification into first-order logic

In order to capture disjointness between classes A and B, we use the axiom:

$$A(x) \supset \neg B(x)$$

which is equivalent to:

$$B(x) \supset \neg A(x)$$

and is weaker than:

$$A(x) \equiv \neg B(x)$$

For every interpretation, the last axiom forces every individual in the domain of the interpretation to be either in the extension of A or in the extension of B. The weaker form, instead, allows individuals to be neither in the extension of A nor in the extension of B, while keeping these extensions disjoint. We note that the weaker form is also the interpretation of class disjointness in OWL [1].

The CRM makes the following class disjointness statements[1]:

- E2 Temporal Entity is disjoint from E77 Persistent Item.
- E18 Physical Thing is disjoint from E28 Conceptual Object.

So we have the following two axioms in $\mathcal{CRM}$:

$$E2(x) \supset \neg E77(x)$$
$$E18(x) \supset \neg E28(x)$$

Since disjointness axioms propagate down the IsA hierarchy, these two axioms sanction the disjointness of many classes. In particular, due to the fact that E2 Temporal Entity and E77 Persistent Item are very high in the CRM class hierarchy, the first axiom creates a wide dichotomy in the class set of the CRM.

3 Translation of property specifications

Table 2 shows the translation in first-order logic of the CRM property specification. As it can be seen from Table 2:

- Property are represented as binary predicate symbols, given by the first part of the corresponding property names; that is, *P16* represents P16 used specific object (was used for).
- Domain, range, sub- and super-property statements are translated in the obvious way.
- The translation of all other aspects is illustrated in a separate section below.

3.1 Meta-Properties

A meta-property is a property whose domain is a property (hence the name). That is, any instance of a property associates an instance of the domain property with a value in the range of the meta-property.

Meta-properties are named by adding a progressive number to their domain properties. For instance[2], "The P3.1 has type property of P3 has note allows differentiation of specific notes, *e.g.* "construction", "decoration" etc. An item may have many notes, but a note is attached to a specific item".

[1] pag. xv
[2] pag. 98

CRM Specification	*Translation into first-order logic*
Property Pn *p-name*	Binary Predicate Pn
P has Domain C	$P(x,y) \supset C(x)$
P has Range D	$P(x,y) \supset D(y)$
P is a Superproperty of Q	$Q(x,y) \supset P(x,y)$
P is a Subproperty of Q	$P(x,y) \supset Q(x,y)$
Quantification	see Section 3.5
P has Meta-Property P.n: C	$P.n(x,y,z) \supset [P(x,y) \wedge C(z)]$
P is Symmetric	$P(x,y) \supset P(y,x)$
P has Asymmetric Meta-Property P.n: C	$P.n(x,y,z) \supset [P(x,y) \wedge \neg P.n(y,x,z) \wedge C(z)]$
P is Transitive	$[P(x,y) \wedge P(y,z)] \supset P(x,z)$
Weak Shortcut P$_1$ ... P$_n$	$[P_1(x,z_1) \wedge P_2(z_1,z_2) \wedge \ldots \wedge P_{n-1}(z_{n-1},y)] \supset P(x,y)$
Strong Shortcut P$_1$ ... P$_n$	$P(x,y) \equiv \exists z_1 \ldots z_{n-1}[P_1(x,z_1) \wedge P_2(z_1,z_2) \wedge \ldots \wedge P_{n-1}(z_{n-1},y)]$

Table 2: Translation into first-order logic of the specification of CRM property P.

In the CRM specification, meta-properties are specified in the context of their domain properties. The specification gives the name of the meta-property and its range.

As shown in Table 2, meta-properties are modelled as 3-place predicate symbols: the first two places are given to the terms in the domain property, the last place is used for the type. The corresponding axiom includes in the consequent the assertion of the domain property, thus making it possible to omit it whenever a typing statment is present.

3.2 Shortcuts

Some properties realize *shortcuts.* For instance[3], "Type assignment events allow a more detailed path from E1 CRM Entity through P41 classified (was classified), E17 Type Assignment, P42 assigned (was assigned by) to E55 Type for assigning types to objects compared to the shortcut offered by P2 has type (is type of)." We call the property offering the shortcut (*e.g.*, *P2* in the previous example), the *shortcut property,* and the corresponding path we call *shortcut path.*

Moreover, shortcuts come in two sorts:

- *strong* shortcuts, in which there is perfect equivalence between the shortcut property and the shortcut path; and
- *weak* shortcuts, in which the shortcut path implies the shortcut property, but not viceversa.

  An example of weak shortcut is given by[4] "P51 has former or current owner (is former or current owner of) is a shortcut for the more detailed path from E18 Physical Thing through P24 transferred title of (changed ownership through), E8 Acquisition, P23 transferred title from (surrendered title through), or P22 transferred title to (acquired title through) to E39 Actor". Not everytime someone is the current owner of something there has been a change of ownership.

A strong shortcut is interpreted as an abbreviation, therefore translated as an equivalence axiom between the shortcut property and the specified path. In the first example above, the equivalence axiom is given by:

$$P2(x,y) \equiv \exists z_1[P41(x,z_1) \wedge P42(z_1,y)]$$

[3]pag. 47-48
[4]pag. 50

Note that, unlike the specification, we omit to state the type of the intermediate nodes because that type has already been declared as the range of the corresponding property. Also note that the usage of a shortcut requires the introduction of $n-1$ unknown individual constants or existentially quantified variables, one for each node on the shortcut's path.

Weak shortcut are instead interpreted as implications from the path to the shortcut predicate. In the last example, the axiom is given by:

$$[P24(x,y) \wedge (P23(y,z) \vee P22(y,z))] \supset P51(x,z)$$

The general forms of the translation of shortcuts is given in Table 2. Notice that weak shortcuts do not require existential variables, due to the semantics of conditionals.

3.3 Symmetry

Symmetry is translated as customary.

Some asymmetry statements are coupled with the declaration of meta-properties, for this reason they are indicated as "asymmetric meta-property" in Table 2. The meaning of such statement is that the property in whose declaration they occur is symmetric, yet their declared meta-property is asymmetric. This is the case of properties *P69, P139* and *P130.* The translation of these statements is simply a combination of asymmetry and meta-property declaration.

3.4 Transitivity

Transitivity is translated as customary.

Properties that have identical domain and range and are not symmetric are transitive, the only exception to this rule being property *P114,* which is symmetric and transitive.

No other property is transitive.

3.5 Property quantification

The CRM specification includes constraints on the cardinality of properties, named "quantification" statements. For self-containedness, the definitions of these contraints is reported for self-containedness in appendix, in Table 7[5].

For the translation in logic of the property quantifiers, we follow a two step approach:

1. In the first step, we reduce the definition of each quantifier to an equivalent set of simpler statements. This is done in Table 8, also given in the appendix. As a result of this step, we have two simpler statements, *total property* and *functional property*, that can be applied to the property or to its inverse. Therefore a property or its inverse adheres to one of the following cases:
   (a) the property is only total, *i.e.*, defined on every element of its domain, but can take up more than one value;
   (b) the property is only functional, *i.e.*, at most one value is provided for any element of its domain;
   (c) the property is neither total, *i.e.*, some domain elements can miss it, nor functional, *i.e.*, more than one value can be provided for any element of its domain;

[5] pag. xii-xiii

(d) the property is both total and functional, *i.e.*all domain element must have one value for it, and no more than one.

2. In the second step, we translate each of the simpler statements obtained in the previous step in first-order logic. We have (as usual, variables are existentially quantified unless otherwise specified):

   - P is a functional property: $[P(x,y) \wedge P(x',y)] \supset (x = x')$
   - P is a total property (having domain A): $A(x) \supset \exists y P(x,y)$
   - the inverse of P is a functional property: $[P(x,y) \wedge P(x,y')] \supset (y = y')$
   - the inverse of P is a total property (having range A): $A(x) \supset \exists y P(y,x)$

The complete translation of each quantifier can then be obtained by conjoining the translation of the corresponding simpler statements. For instance:

- *many to many* (0,n:0,n): no axiom is required
- *one to one* (1,1:1,1) means that both P and its inverse are total and functional (here A and B are the domain and the range pf property P, respectively). We then have the translation:

  $$A(x) \supset \exists y P(x,y)$$
  $$[P(x,y) \wedge P(x',y)] \supset (x = x')$$
  $$B(x) \supset \exists y P(y,x)$$
  $$[P(x,y) \wedge P(x,y')] \supset (y = y')$$

  where the first two axioms capture totality and functionality of P, respectively, and the last two do the same for the inverse of P.

4 Validation

Every class and every property is consistent. The ontology is consistent

Then, we need to find some test that the formalization respects the intuition, some reasoning patterns.

Properties that we expect to be satisfied by any KB, can be proved to hold.

5 The OWL ontology `CRM`

Having expressed the CRM in the neutral language of first-order logic, we now proceed to encode the resulting first-order theory in OWL, with the objective of making a step towards the implementation of the CRM. This is clearly not the only way of implementing the CRM, but it presents at least two advantages: first, existing implementations of OWL can be used to manage the creation and the evolution of instances of the CRM; second, the SPARQL query language can be used to extract information from instances of the CRM at no cost.

Our OWL encoding of the CRM, which is called the `CRM` ontology, is given in the functional notation [2]. In order to derive it, we proceed in the obvious way by translating the logic formulae derived in Section 2 and 3 into OWL axioms. As it will be shown, the so obtained axioms are not the only needed ones.

We will use `crm:` as the prefix for the CRM namespace, that is the namespace containing the URIs of the OWL classes and properties corresponding to the CRM classes and properties.

CRM Specification	*Translation into first-order logic*	*OWL Specification*
Class E*n* *c-name*	Unary Predicate En	`Declaration(Class(crm:En))`
A Superclass of B	$B(x) \supset A(x)$	`SubClassOf(B A)`
A Subclass of B	$A(x) \supset B(x)$	`SubClassOf(A B)`

Table 3: Translation of class specification into OWL

```
SubClassOf(crm:E1 owl:Thing)
SubClassOf(crm:E41 rdf:Literals)
SubClassOf(crm:E59 rdf:Literals)
SubClassOf(owl:real crm:E60)
```

Table 4: The Class Mapping Axioms

5.1 Translation of class specifications

CRM classes find their natural correspondents in OWL classes, therefore it seems natural to establish a one-to-one correspondence between the former and the latter. This correspondence is reflected in the first row of Table **??** which establishes that each class be declared by means of an OWL declaration statement. The subsequent rows give the OWL encoding of sub- or super-class statements.

In the rest of this Section we will present two more sets of axioms required for the proper encoding of the CRM classes in OWL.

5.1.1 Mapping axioms

OWL comes with built-in classes whose intended meaning overlaps with the intended meaning of some CRM classes. We need therefore to capture this fact by declaring the proper relationship between the involved (OWL correspondents of) CRM classes and their related OWL built-in classes. These declarations take the form of OWL axioms, called *class mapping axioms.*

The list of OWL 2 reserved names is given in Appendix, in Table 9. These includes the built-in classes. Based on these, we have the following mapping axioms:

- E1 CRM Entity is translated into the OWL class `crm:E1` which is declared as a subclass of `owl:Thing`.
- E41 Appellation is translated into the OWL class `crm:E41` which is declared as a subclass of `rdf:Literals`.
- Analogously to appellations, E59 Primitive Value is translated into the OWL class `crm:E59` which is declared as a subclass of `rdf:Literals`.
- E60 Number is translated into the OWL class `crm:E60` which is declared to be a superclass of `owl:real` since it includes also complex numbers, vectors and tensors.

Table 4 gives the resulting class mapping axioms.

There are other classes in the CRM specification that model data values and can be translated into XML Schema datatypes. A notable example is class E50 Date which finds a natural correspondent in `xsd:date`. Since datatypes lie outside OWL proper, we do not specify these correspondences here, although it clearly seems a good practice to rely on the XML Schema type system for implementing the CRM.

CRM Specification	*Translation into OWL*
Property P*n* *p-name*	`Declaration(DataProperty(crm:Pn))` or `Declaration(ObjectProperty(crm:Pn))`
P has Domain C	`ObjectPropertyDomain(crm:Pn crm:C)`
P has Range D	`ObjectPropertyRange(crm:Pn crm:D)`
P is a Superproperty of Q	`SubObjectPropertyOf(crm:Q crm:Pn)`
P is a Subproperty of Q	`SubObjectPropertyOf(crm:Pn crm:Q)`
Quantification	see Section 5.2.3
P has Meta-Property P.*n*: C	see Section 5.2.2
P is Symmetric	`SymmetricObjectProperty(crm:Pn)` *
P has Asymmetric Meta-Property P.*n*: C	see Section 5.2.2
P is Transitive	`TransitiveObjectProperty(crm:Pn)`
Weak Shortcut $P_1 \dots P_n$	`SubObjectPropertyOf(ObjectPropertyChain(crm:`P_1 `...` `crm:`P_n`) crm:Pn)`
Strong Shortcut $P_1 \dots P_n$	see Section 5.2.4

Table 5: Translation into OWL of the specification of CRM property *P*.

5.1.2 Class disjointness axioms

The CRM disjointness statements are translated in the following *class disjointness axioms:*

- E2 Temporal Entity is disjoint from E77 Persistent Item: `DisjointClasses(crm:E2 crm:E77)`
- E18 Physical Thing is disjoint from E28 Conceptual Object: `DisjointClasses(crm:E18 crm:E28)`

5.2 Translation of property specifications

5.2.1 Data *vs.* object properties

Similarly to CRM classes, CRM properties find their natural correspondents in OWL properties. However, OWL distinguishes between two kinds of properties:

- *object properties,* connecting two individuals, and
- *data properties,* connecting an individual and a literal.

Therefore it must be determined, for each CRM property, whether it is encoded as an OWL object or data property. Upon discussing mapping axioms, we have established that the correspondents of E41 Appellation and E59 Primitive Value are sub-classes of `rdf:Literals`. To be consistent with this determination, we must now settle that all properties having either `crm:E41`, or `crm:E59`, or a subclass of theirs, be translated in OWL as data properties, and all remaining properties be translated into OWL object properties. This is reflected in the first two rows of Table 5, which unlike Table 3, does not report the first-order translation for reasons of space. For the same reason, from the second row down, it is assumed that the involved CRM property *P* is mapped onto an object property. When the same construct is not available for data properties, the OWL translation is marked with a star.

P is functional	`FunctionalObjectProperty(crm:P)` or `FunctionalDataProperty(crm:P)`
P is total on A	`SubClassOf(crm:A ObjectSomeValuesFrom(crm:P owl:Thing))`
the inverse of P is functional	`InverseFunctionalObjectProperty(crm:P)`
the inverse of P is total on A	`SubClassOf(crm:A ObjectSomeValuesFrom(ObjectInverseOf(crm:P) owl:Thing))`

Table 6: Property quantification axioms in OWL

5.2.2 Meta-properties modeling in OWL

We recall from Section 3.1 that a meta-property $P.n$ of a property P is a property that associates an instance (a, b) of P with a value c in the meta-property range. As such it can be modeled in OWL via reification, that is by a class `CP.n` each instance of which stands for an instance of P and is connected to the individual c above via meta-property `P.n`. More technically:

1. for each object meta-property $P.n$ of a property P, the following declarations are required:

   ```
   Declaration(Class(crm:CP.n))
   Declaration(ObjectProperty(crm:P1.n))
   Declaration(ObjectProperty(crm:P2.n))
   Declaration(ObjectProperty(crm:P.n))
   ```

2. for each instance (a, b, c) of the meta-property $P.n$, the following assertions are created, where `I` is the anonymous individual that reifies the meta-property instance:

   ```
   ClassAssertion(crm:CP.n I)
   ObjectPropertyAssertion(crm:P1.n I a)
   ObjectPropertyAssertion(crm:P2.n I b)
   ObjectPropertyAssertion(crm:P.n I c)
   ObjectPropertyAssertion(crm:P a b)
   ```

3. if the meta-property is asymmetric, then the following assertion must be added:

   ```
   NegativeObjectPropertyAssertion(crm:P.n c I)
   ```

The assumption here is that the same anonymous individual `I` is used to reify the instance (a, b) of P. A proper user interface may facilitate the enforcement of this assumption.

All the above assertions cannot be obtained as implicit knowledge via axioms, thus their insertion in the ontology has to be performed procedurally.

5.2.3 Property quantification

We recall from Section 3.5 that the specification of the CRM describe properties, or their inverses, as functional or total. The corresponding OWL axioms are given in Table 5.2.3. We recall that the inverse of a data property is not expressible in OWL.

5.2.4 Strong shortcuts

Strong shortcuts are equivalence statements consisting of an *if* and an *only-if* part. For example, let us consider the strong shortcut:

for assigning types to objects. The *only-if* part of the equivalence is a weak shortcut and is captured by the axiom (cf. Table 5):

```
SubObjectPropertyOf(ObjectPropertyChain(crm:P41 crm:P42) crm:P2)
```

On the contrary, for each instance (a, b) of the property expressed via the assertion:

```
ObjectPropertyAssertion(crm:a P2 crm:b)
```

the *if* part of the equivalence implies the following assertions:

```
ObjectPropertyAssertion(crm:a P41 _:a1)
ObjectPropertyAssertion(_:a1 P42 crm:b)
```

where `_:a1` is an anonymous individual. While these assertions can be expressed on an individual basis, there is no way of obtaining them as implicit knowledge via some axioms.

5.3 Conclusions

In this section, we have presented `CRM`, an OWL ontology including the axioms that capture the semantics of the CRM vocabulary. In particular, `CRM` includes the following sets of axioms:

- *class axioms:* these are the axioms capturing the semantics of CRM classes, derived based on the rules given in Table 3;
- *class mapping axioms:* these are the axioms establishing the proper connection between the CRM classes and the OWL built-in classes, they are given in Table 4;
- *class disjointness axioms:* these are the axioms expressing the disjointness constraints between CRM classes, they are given in Section 5.1.2;
- *property axioms:* these are the axioms capturing the semantics of CRM properties, derived based on the rules given in Tables 5 and 6.

The full expression of the CRM is not possible in OWL, since neither strong shortcuts nor meta-properties are directly expressible.

The problem is not severe for meta-properties; in fact, meta-property instances never result as implicit knowledge, they always result from manual insertion, and it is a simple matter to extend each such manual insertion with the automatic insertion of the statements presented in Section 5.2.2 above.

For shortcuts, the problem is in principle much more severe than the previously discussed meta-property problem, because a shortcut property instance may result as implicit knowledge, therefore the insertion of the corresponding assertions is considerably more difficult.

many to many (0,n:0,n)	Unconstrained: An individual domain instance and range instance of this property can have zero, one or more instances of this property. In other words, this property is optional and repeatable for its domain and range.
one to many (0,n:0,1)	An individual domain instance of this property can have zero, one or more instances of this property, but an individual range instance cannot be referenced by more than one instance of this property. In other words, this property is optional for its domain and range, but repeatable for its domain only. In some contexts this situation is called a "fan-out".
many to one (0,1:0,n)	An individual domain instance of this property can have zero or one instance of this property, but an individual range instance can be referenced by zero, one or more instances of this property. In other words, this property is optional for its domain and range, but repeatable for its range only. In some contexts this situation is called a "fan-in".
many to many, necessary (1,n:0,n)	An individual domain instance of this property can have one or more instances of this property, but an individual range instance can have zero, one or more instances of this property. In other words, this property is necessary and repeatable for its domain, and optional and repeatable for its range.
one to many, necessary (1,n:0,1)	An individual domain instance of this property can have one or more instances of this property, but an individual range instance cannot be referenced by more than one instance of this property. In other words, this property is necessary and repeatable for its domain, and optional but not repeatable for its range. In some contexts this situation is called a "fan-out".
many to one, necessary (1,1:0,n)	An individual domain instance of this property must have exactly one instance of this property, but an individual range instance can be referenced by zero, one or more instances of this property. In other words, this property is necessary and not repeatable for its domain, and optional and repeatable for its range. In some contexts this situation is called a "fan-in".
one to many, dependent (0,n:1,1)	An individual domain instance of this property can have zero, one or more instances of this property, but an individual range instance must be referenced by exactly one instance of this property. In other words, this property is optional and repeatable for its domain, but necessary and not repeatable for its range. In some contexts this situation is called a "fan-out".
one to many, necessary, dependent (1,n:1,1)	An individual domain instance of this property can have one or more instances of this property, but an individual range instance must be referenced by exactly one instance of this property. In other words, this property is necessary and repeatable for its domain, and necessary but not repeatable for its range. In some contexts this situation is called a "fan-out".
many to one, necessary, dependent (1,1:1,n)	An individual domain instance of this property must have exactly one instance of this property, but an individual range instance can be referenced by one or more instances of this property. In other words, this property is necessary and not repeatable for its domain, and necessary and repeatable for its range. In some contexts this situation is called a "fan-in".
one to one (1,1:1,1)	An individual domain instance and range instance of this property must have exactly one instance of this property. In other words, this property is necessary and not repeatable for its domain and for its range.

Table 7: Property Quantifiers in the CRM specification

many to many (0,n:0,n)	Unconstrained: An individual domain instance and range instance of this property can have zero, one or more instances of this property. $\longrightarrow$ *No axiom is required* (*Nax* for short)
one to many (0,n:0,1)	An individual domain instance of this property can have zero, one or more instances of this property, $\longrightarrow$ *Nax* but an individual range instance cannot be referenced by more than one instance of this property. $\longrightarrow$ *Inverse functional*
many to one (0,1:0,n)	An individual domain instance of this property can have zero or one instance of this property, $\longrightarrow$ *Functional* but an individual range instance can be referenced by zero, one or more instances of this property. $\longrightarrow$ *Nax*
many to many, necessary (1,n:0,n)	An individual domain instance of this property can have one or more (*must have at least one?*) instances of this property, $\longrightarrow$ *Total* but an individual range instance can have zero, one or more instances of this property. $\longrightarrow$ *Nax*
one to many, necessary (1,n:0,1)	An individual domain instance of this property can have one or more (*must have at least one?*) instances of this property, $\longrightarrow$ *Total* but an individual range instance cannot be referenced by more than one instance of this property. $\longrightarrow$ *Inverse functional*
many to one, necessary (1,1:0,n)	An individual domain instance of this property must have exactly one instance of this property, $\longrightarrow$ *Total Functional* but an individual range instance can be referenced by zero, one or more instances of this property. $\longrightarrow$ *Nax*
one to many, dependent (0,n:1,1)	An individual domain instance of this property can have zero, one or more instances of this property, $\longrightarrow$ *Nax* but an individual range instance must be referenced by exactly one instance of this property. $\longrightarrow$ *Inverse Total Functional*
one to many, necessary, dependent (1,n:1,1)	An individual domain instance of this property can have one or more instances of this property, $\longrightarrow$ *Total* but an individual range instance must be referenced by exactly one instance of this property. $\longrightarrow$ *Inverse Total Functional*
many to one, necessary, dependent (1,1:1,n)	An individual domain instance of this property must have exactly one instance of this property, $\longrightarrow$ *Total Functional* but an individual range instance can be referenced by one or more instances of this property. $\longrightarrow$ *Inverse Total*
one to one (1,1:1,1)	An individual domain instance and range instance of this property must have exactly one instance of this property. $\longrightarrow$ *Total Functional* $\longrightarrow$ *Inverse Total Functional*

Table 8: Break-down of the Property Quantifiers definitions

owl:backwardCompatibleWith	owl:bottomDataProperty	owl:bottomObjectProperty	owl:deprecated	owl:incompatibleWith
owl:Nothing	owl:priorVersion	owl:rational	owl:real	owl:versionInfo
owl:Thing	owl:topDataProperty	owl:topObjectProperty	rdf:langRange	rdf:PlainLiteral
rdf:XMLLiteral	rdfs:comment	rdfs:isDefinedBy	rdfs:label	rdfs:Literal
rdfs:seeAlso	xsd:anyURI	xsd:base64Binary	xsd:boolean	xsd:byte
xsd:dateTime	xsd:dateTimeStamp	xsd:decimal	xsd:double	xsd:float
xsd:hexBinary	xsd:int	xsd:integer	xsd:language	xsd:length
xsd:long	xsd:maxExclusive	xsd:maxInclusive	xsd:maxLength	xsd:minExclusive
xsd:minInclusive	xsd:minLength	xsd:Name	xsd:NCName	xsd:negativeInteger
xsd:NMTOKEN	xsd:nonNegativeInteger	xsd:nonPositiveInteger	xsd:normalizedString	xsd:pattern
xsd:positiveInteger	xsd:short	xsd:string	xsd:token	xsd:unsignedByte
xsd:unsignedInt	xsd:unsignedLong	xsd:unsignedShort		

Table 9: Reserved Vocabulary of OWL 2 with Special Treatment

A Tables

References

[1] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau. OWL 2 Web Ontology Language direct semantics (second edition). Technical report, W3C Recommendation, 11 December 2012. `http://www.w3.org/TR/owl2-direct-semantics/`.

[2] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language. structural specification and functional-style syntax (second edition). Technical report, W3C Recommendation, 11 December 2012. `http://www.w3.org/TR/owl2-syntax/`.