

Sydney OWL Syntax (SOS)

Authors: Anne Cregan, Rolf Schwitter and Thomas Meyer

Version: 0.1

Date: 2007-04-27

Status: Working Draft

The following notation is used for Sydney OWL Syntax throughout this document:

- $c[N]$ denotes a (potentially complex) nominal category in SOS, for example $c[\text{man}]$ or $c[\text{big, man}]$
- $op[N]$ denotes a (potentially complex) nominal category in SOS that is part of an object property, for example $\text{has} \dots op[\text{child}]$ results in *hasChild* or $\text{has} \dots op[\text{big, child}]$ results in *hasBigChild*.
- $op[V]$ denotes a verbal category in SOS, for example *likes, married to* iID denotes a name in SOS, for example *Rona, Aaron*
- x, y, z are variables in SOS.

Update (Meeting: 2007-03-22)

- OWL classes are always verbalised by a noun phrase with an explicit quantifier, for example instead of

Person is equivalent to ...

we use

Any person is equivalent to ...

or

Any Person is equivalent to ...

since this allows us to properly distinguish between class names which occur in lowercase (*person*) or uppercase (*Person*) in the OWL ontology.

- Instead of

Likes and ... and admires are equivalent

we use the explicit verbalisation:

The relation likes and ... and the relation admires ...

since this makes it possible to speak about properties in an uniform way and allows us additionally to distinguish between property names which occur in lowercase (*likes*) or uppercase (*Likes*) in the OWL ontology. We decided to use the noun “relation” and

not the noun “property” in SOS since we assume that the first noun is easier to understand for non-specialists.

- Some constructions are very explicit in SOS. We use, for example:

The relation likes and ... and the relation admires ...

instead of

The relation likes and ... and admires ...

We will investigate later which constructions can be expressed in a more compact form and which elements can be elided.

Update (Meeting: 2007-04-24)

- We decided to minimize the use of variables:

Instead of

If X a man then X is a person.
Each X has itself as an admirer.
Each X does not like itself.

we use

Every man is a person.
Everything has itself as an admirer.
Nothing has itself as an admirer.

- We decided to allow the use of *something* and *something else* as synonyms for the variables x and y .
- We decided to introduce classes in a similar way as relations, e.g.:

The class aunt and the class auntie are equivalent.

whereas the default will be

The classes aunt and auntie are equivalent.

that means *class* needs to be mentioned only once but results in plural form. Similar for relations:

The relations likes and admires are equivalent.

Functional-Style Syntax	SOS Abstract Form	SOS Examples
InverseObjectProperty(op)	implicit	implicit
ObjectUnionOf($c_1 \dots c_n$)	a $c[N_1]$ or ... or a $c[N_n]$	a man or ... or an adult
ObjectIntersectionOf($c_1 \dots c_n$)	a $c[N_1]$ and ... and a $c[N_n]$	a man and ... and an adult
ObjectComplementOf(c)	is not a $c[N]$	is not a man
ObjectOneOf(iID \dots iID _n)	is one of iID ₁ or ... or iID _n	is one of Tom or ... or Harry
ObjectSomeValuesFrom(op c)	has some $c[N_1]$ as a $op[N_2]$ $op[V]$ some $c[N_1]$	has some animal as a pet likes some animal
ObjectAllValuesFrom(op c)	has only $c[N_1]$ as a $op[N_2]$ $op[V]$ only $c[N_1]$	has only animals as a pet likes only animals
ObjectExistsSelf(op)	$op[V]$ itself	likes itself
ObjectHasValue(op iID)	has iID as a $op[N]$	has John as a parent.
ObjectMinCardinality(n op c)	has at least n $c[N_1]$ as a $op[N_2]$	has at least 3 cats as an animal
ObjectMaxCardinality(n op c)	has at most n $c[N_1]$ as a $op[N_2]$	has at most 3 cats as an animal
ObjectExactCardinality(n op c)	has exactly n $c[N_1]$ as a $op[N_2]$	has exactly 3 cats as an animal
ObjectMinCardinality(n op)	has at least n $op[N]$	has at least 3 animals
ObjectMaxCardinality(n op)	has at most n $op[N]$	has at most 3 animals
ObjectExactCardinality(n op)	has exactly n $op[N]$	has exactly 3 animals
SubClassOf($c_1 \ c_2$)	Every $c[N_1]$ is a $c[N_2]$.	Every man is a person.
EquivalentClasses($c_1 \dots c_n$)	The classes $c[N_1]$ and ... and $c[N_n]$ are equivalent.	The classes aunt and ... and auntie are equivalent.
DisjointClasses($c_1 \dots c_n$)	The classes $c[N_1]$ and ... and $c[N_n]$ are mutually exclusive.	The classes female and ... and male are mutually exclusive.
DisjointUnion(cID $c_1 \dots c_n$)	The class cID $[N_1]$ is fully defined as $c[N_1]$ or ... or $c[N_n]$, and $c[N_1]$ and ... and $c[N_n]$ are mutually exclusive.	The class person is fully defined as female or ... or male, and female and ... and male are mutually exclusive.
SubObjectPropertyOf($op_1 \ op_2$)	If X has Y as a $op_1[N_1]$ then X has Y as a $op_2[N_2]$.	If X has Y as a parent then X has Y as an ancestor.
SubObjectPropertyOf(subObjectPropertyChain($op_1 \dots op_n$) op)	If X $op[V]$ Y and Y has Z as a $op_1[N_1]$ and ... then X $op[V]$ Z.	If X owns Y and Y has Z as a part and ... then X owns Z.
EquivalentObjectProperties($op_1 \dots$ op_n)	The relations $op_1[V_1]$ and ... and $op_n[V_n]$ are equivalent.	The relations likes and ... and admires are equivalent.
DisjointObjectProperties($op_1 \dots$ op_n)	The relations $op_1[V_1]$ and ... and $op_n[V_n]$ are mutually exclusive.	The relations married to and ... and has ancestor are mutually exclusive.
ObjectPropertyDomain(op c)	If X has Y as a $op[N_1]$ then X is a $c[N_2]$.	If X has Y as an ancestor then X is a person.
ObjectPropertyRange(op c)	If X has Y as a $op[N_1]$ then Y is a $c[N_2]$.	If X has Y as an ancestor then Y is a person.
InverseObjectProperties($op_1 \ op_2$)	If X has Y as a $op_1[N_1]$ then Y has X as a $op_2[N_2]$.	If X has Y as a parent then Y has X as a child.

TransitiveObjectProperty(op)	If X has Y as a op[N ₁] and Y has Z as a op[N ₁] then X has Z as a op[N ₁].	If X has Y as an ancestor and Y has Z as an ancestor then X has Z as an ancestor.
FunctionalObjectProperty(op)	If X has Y as a op[N ₁] then Y is the only op[N ₁] of X.	If X has Y as a father then Y is the only father of X.
InverseFunctionalObjectProperty(op)	If X has Y as a op[N ₁] then Y is the op[N ₁] of the only X.	If X has Y as a son then Y is the son of only X.
ReflexiveObjectProperty(op)	Everything has itself as a op[N].	Everything has itself as an admirer.
IrreflexiveObjectProperty(op)	Nothing op[V] itself.	Nothing likes itself.
SymmetricObjectProperty(op)	If X has Y as a op[N ₁] then Y has X as a op[N ₁].	If X has Y as a sibling then Y has X as a sibling.
AntisymmetricObjectProperty(op)	If X has Y as a op[N ₁] and Y has X as a op[N ₁] then X is identical to Y.	If X has Y as an ancestor and Y has X as an ancestor then X is identical to Y.
SameIndividual(iID ₁ ... iID _n)	iID ₁ and ... and iID _n are the same individual.	Ron and ... and Ronald are the same individual.
DifferentIndividuals(iID ₁ ... iID _n)	iID ₁ and ... and iID _n are different individuals.	Rona and ... and Aaron are different individuals.
ClassAssertion(iID c)	iID is a c[N].	Rona is a girl.
ObjectPropertyAssertion(op iID ₁ iID ₂)	iID ₁ op[V] iID ₂ .	Rona likes Aaron.
NegativeObjectPropertyAssertion(op iID ₁ iID ₂)	iID ₁ does not op[V] iID ₂ .	Rona does not like Bart.