W3C Editor's Draft

New Features and Rationale

W3C Editor's Draft 28 November 2008

Authors:
> Christine Golbreich, University of Versailles Saint-Quentin
> Evan K. Wallace, NIST
Contributors:
> Vipul Kashyap, Partners Health System, Inc.
> Elisa Kendall, Sandpiper Software
> Deborah McGuinness, Rensselaer Polytechnic Institute
> Bijan Parsia, The University of Manchester
> *Note*: The complete list of contributors is being compiled and will be included in the next draft.

This document is also available in these non-normative formats: PDF version.

Abstract

OWL 2 extends the W3C OWL Web Ontology Language with a small but useful set of features that have been requested by users, for which effective reasoning algorithms are now available, and that OWL tool developers are willing to support. The new features include extra syntactic sugar, additional property and qualified cardinality constructors, extended datatype support, simple metamodeling, and extended annotations.
This document is a simple introduction to the new features of the OWL 2 Web Ontology Language, including an explanation of its differences with respect to OWL 1. It also presents the requirements that have motivated the design of the main new

features, and their rationale from a theoretical and implementation perspective. Thus it is intended to supplement the Use Cases and Requirements provided as part of the initial OWL Recommendation [*OWL Use Cases and Requirements*] and to be an overview of OWL 2 new features.

Status of this Document

May Be Superseded

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Set of Documents

This document is being published as one of a set of 11 documents:

1. Structural Specification and Functional-Style Syntax
2. Direct Semantics
3. RDF-Based Semantics
4. Conformance and Test Cases
5. Mapping to RDF Graphs
6. XML Serialization
7. Profiles
8. Quick Reference Guide
9. New Features and Rationale (this document)
10. Manchester Syntax
11. rdf:text: A Datatype for Internationalized Text

Please Comment By 2008-12-01

The OWL Working Group seeks public feedback on these Working Drafts. Please send your comments to public-owl-comments@w3.org (public archive). If possible, please offer specific changes to the text that would address your concern. You may also wish to check the Wiki Version of this document for internal-review comments and changes being drafted which may address your concerns.

No Endorsement

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

W3C Editor's Draft

Patents

This document was produced by a group operating under the [5 February 2004 W3C Patent Policy](). W3C maintains a [public list of any patent disclosures]() made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains [Essential Claim(s)]() must disclose the information in accordance with [section 6 of the W3C Patent Policy]().

Contents

W3C Editor's Draft

1 Overview

This document is an Overview of the main new features that have been added to OWL 2 and of their rationale. These features are based on real applications, user and tool developer experience, much of which was documented and discussed as part of the OWLED Workshop Series. For each new feature, the document informaly introduces its meaning, as specified in the [Direct Semantics] document, and also its syntax, as defined in the [Syntax] document. It provides an explanation of the fundamental reasons that have motivated the design of these features, from a theoretical and implementation perspective. Furthermore, each feature is illustrated by simple examples from various applications, which are referred in an

Appendix including some Use Cases (among others) that motivated these extensions. Finally, two synthetic tables resume the links between the use cases, requirements, and examples provided for illustration of the new features.

2 Features & Rationale

OWL 2 is an update to OWL adding several new features, including an increased expressive power - mainly w.r.t. properties, extended support for datatypes, simple metamodelling capabilities, extended annotation capabilities, database style keys. OWL 2 also defines several profiles, OWL 2 language subsets that may better meet certain performance requirements or may be easier to implement. Thus, OWL 2 new features are presented in this document according to the following categories:

1. extra syntactic sugar to make some common statements easier to say, e.g., the disjoint union of classes
2. new constructs that increase the expressivity for properties, e.g., qualified cardinality restrictions or property chain inclusion, database style keys
3. extended support for datatypes, e.g., data type restrictions and facets for restricting a datatype to a subset of its values
4. simple metamodeling capabilities to express metalogical information about the entities of an ontology
5. extended annotations capabilities to annotate entities, ontologies and also axioms
6. other major innovations: declarations, new language profiles (sublanguages).

Each of these features is described according to a common pattern as follows:

- a brief sentence explaining why the new feature is required
- a feature description including: a short informal sentence defining its meaning - with a link to the Semantics document, followed by its syntax - with a link to the Syntax document - and simple illustrative example(s) issued from the Use Cases.
- the rationale from a theoretical and implementation perspectives that led to the changes to OWL 1 seen in OWL 2.
- links to related use cases of the bibliography in the Appendix.

In addition, Section 2.7 discusses structural differences from OWL 1 that manifest themselves in the abstract syntax of OWL 2.

Depending on his profile (e.g., user, theoretician, implementor) the reader can focus on what he is more interested in this document thanks to different buttons that allow to *Hide and Show* the feature Examples, Theory or Implementation perspectives, respectively.

W3C Editor's Draft

Hide Theoretical Perspective   Hide Implementation Perspective

Hide Examples

2.1 Syntactic sugar

"Syntactic sugar" is a term for syntactic extensions that make a language friendlier to users yet do not extend the expressivity of the language. OWL 2 adds syntactic sugar to make some common patterns easier to write. Among these are two new shorthands that provide more concise ways to state disjointness among classes. These shorthands are: DisjointUnion and DisjointClasses.

2.1.1 F1: DisjointUnion

While OWL 1 provides means to define a set of subclasses as a disjoint and complete covering of a superclass by using several axioms, this cannot be done concisely. This feature has been required as a shortcoming given the common use of such satements in domain models.

Feature

DisjointUnion defines a class as the union of other classes, all of which are pair-wise disjoint. It is a shorthand for owl:disjointWith statements used in combination with owl:unionOf to define a complete superclass from a set of mutually disjoint subclasses. [Syntax] [Semantics]

```
DisjointUnion := 'DisjointUnion' '(' { Annotation } Class
ClassExpression ClassExpression { ClassExpression } ')'
```

Example:
- HCLS

`DisjointUnion(BrainHemisphere LeftHemisphere RightHemisphere ) (UC#2)`	A *BrainHemisphere* is exclusively either a *LeftHemisphere* or a *RightHemisphere* and cannot be both a *BrainHemisphere* and a *LeftHemisphere*.
`DisjointUnion(Lobe FrontalLobe ParietalLobe TemporalLobe OccipitalLobe LimbicLobe) (UC#1)`	A *Lobe* is exclusively either a *FrontalLobe* , a *ParietalLobe*, a *TemporalLobe*, a

```
DisjointUnion(AmineGroup
PrimaryAmineGroup
SecondaryAmineGroup
TertiaryAmineGroup ) (UC#3)
```

OccipitalLobe or a *LimbicLobe* and cannot be both of them.

An *AmineGroup* is exclusively either a *PrimaryAmineGroup*, a *SecondaryAmineGroup* or a *TertiaryAmineGroup* and cannot be both of them.

- Automotive industry

```
DisjointUnion(CarDoor FrontDoor
RearDoor TrunkDoor) (UC#4)
```

A *CarDoor* is exclusively either a *FrontDoor*, a *RearDoor* or a *TrunkDoor* and not both of them.

Theoretical Perspective

Since *DisjointUnion* is simply a shorthand for several *disjointWith* statements in combination with unionOf, it does not change the expressiveness, semantics, or complexity of the language.

Implementation Perspective

Being syntactic sugar, it's possible to take an ontology that is OWL 1 except for DisjointUnion and preprocess it into an equivalent OWL 1 ontology without DisjointUnion. Implementations, however, may prefer to take special notices of DisjointUnion for more efficient loading reasons.

Use cases

Use Case #1 Use Case #2 Use Case #3 Use Case #4

2.1.2 F2: DisjointClasses

While OWL 1 provides means to state that two subclasses are disjoint, stating that several subclasses are pair-wise disjoint cannot be done concisely. This shortand has been required given the common use of such disjointness statements by many applications.

Feature

DisjointClasses states that all classes from the set are pair-wise disjoint. It is a shorthand for several owl:disjointWith statements to define a set of mutually disjoint subclasses. [Syntax] [Semantics]

W3C Editor's Draft

```
DisjointClasses := 'DisjointClasses' '(' { Annotation }
ClassExpression ClassExpression { ClassExpression } ')'
```

Example:

- HCLS

DisjointClasses(*LeftLung RightLung*) (*UC#2*)	Nothing can be both a *Leftlung* and a *RightLung*.
DisjointClasses(*UpperLobeOfLung MiddleLobeOfLung LowerLobeOfLung*) (*UC#2*)	*UpperLobeOfLung MiddleLobeOfLung LowerLobeOfLung* are pairwise exclusive.

Note: The FMA exhibit a huge number of such classes [*FMA* C in Appendix]: 3736 classes of template *Left X vs Right X* (e.g. Left lung vs Right lung) 13989 classes of template *X left Y vs X right Y* (e.g. Skin of right breast vs Skin of left breast) 25 classes with template *Male X* vs Female X *(e.g. Male breast vs Female breast)* 75 classes X *male Y vs X female Y* (e.g. Right side of male chest vs Right side of female chest)

Theoretical Perspective

Since *DisjointClasses* is simply a shorthand for several *disjointWith* statements it does not change the expressiveness, semantics, or complexity of the language.

Implementation Perspective

Being syntactic sugar, it's possible to take an ontology that is OWL 1 except for DisjointClasses and preprocess it into an equivalent OWL 1 ontology without DisjointClasses. Implementations, however, may prefer to take special notices of DisjointClasses since the terseness and "groupiness" make for more efficient loading.

Use cases

Use Case #1 Use Case #2

2.1.3 F3: NegativeObjectPropertyAssertion NegativeDataPropertyAssertion

While OWL 1 provides means to assert values of a property for an individual, asserting that a property has not some values is impossible. This requires the ability to assert facts about an individual stating property values that it does not have.

W3C Editor's Draft

Feature

OWL 2 provides the shortands *NegativeObjectPropertyAssertion* and *NegativeDataPropertyAssertion* for asserting negative facts. While an *ObjectPropertyAssertion* (resp. *DataPropertyAssertion*) axiom states that a given property holds for the given individuals, a *NegativeObjectPropertyAssertion* (resp. *NegativeDataPropertyAssertion*) axiom states that a given property does not hold for the given individuals. [Syntax] [Semantics]

NegativeObjectPropertyAssertion := 'NegativePropertyAssertion' '(' { **Annotation** } **objectPropertyExpression sourceIndividual targetIndividual** ')'

NegativeDataPropertyAssertion := 'NegativePropertyAssertion' '(' { **Annotation** } **DataPropertyExpression sourceIndividual targetValue** ')'

Example:
- HCLS

```
NegativePropertyAssertion(
livesIn ThisPatient
IledeFranceDistrict ) (UC#9)
NegativePropertyAssertion(
hasAge ThisPatient
5^^xsd:integer ) (UC#9)
```

ThisPatient does not live in the *IledeFranceDistrict* .

ThisPatient is not five years old.

Theoretical Perspective

Since *NegativePropertyAssertion* is simply a shorthand it does not change the expressiveness, semantics, or complexity of the language.

Implementation Perspective
Being syntactic sugar, it's possible to take an ontology that is OWL 1 except for NegativePropertyAssertion and preprocess it into an equivalent OWL 1 ontology without it.
Use cases

Use Case #9

2.2 New constructs for Properties

OWL 1 was mainly focused on constructs for expressing information about classes and individuals, but paid less attention to properties. OWL 1 exhibited some weakness regarding expressivenes for properties. OWL 2 addresses it by

complementing OWL 1 with new constructs that increase the expressivity of the language for properties, as requested by many users. OWL 2 offers new constructs for expressing additional restrictions on properties, new characteristics of properties, incompatibility of properties, properties chains and key properties.

2.2.1 F4: Self Restriction

OWL 1 does not allow to define subclasses of objects that are related to themselves by a given property, for example the subclass of processes that auto-regulate themselves. Expressing this requires local reflexivity.

Feature

OWL 2 allows to assert restrictions on object properties by means of the new construct *HasSelf*. The class expression **ObjectHasSelf** defined using an *HasSelf* restriction on an object property denotes the class of all objects that are related to themselves via the given object property. It can be viewed as a kind of *local reflexivity* quality of the object property. [Syntax] [Semantics]

```
ObjectHasSelf := 'HasSelf' '(' ObjectPropertyExpression ')'
```

Example:
 • HCLS

`SubClassOf( AutoRegulatingProcess HasSelf( regulate) )`	Auto-regulating processes regulate themselves.
`SubClassOf( RingMolecule HasSelf( connectedTo)) (UC#3)`	Biochemical ring molecules are *connectedTo* themselves.

Theoretical Perspective

The description logic underlying OWL-DL is SHOIN. OWL 2 is based on a more expressive description logic: SROIQ [*SROIQ*]. SROIQ extension of SHOIN was designed to provide all possible useful additions to OWL-DL that were requested by users, while not affecting its decidability and practicability. SROIQ logic extends SHOIN with reflexive, asymmetric, and irreflexive roles, disjoint roles, a universal role, and constructs ∃ R.Self. It also allows qualified number restrictions and negated role assertions in Aboxes.

Additionally, SROIQ offers complex role inclusion axioms of the form R ∘ S < R or S ∘ R < R to express propagation of one property along another one, which have proven to be very useful in particular for biomedical ontologies (see F8: Property chain inclusion).

W3C Editor's Draft

Implementation Perspective

Local reflexivity is already supported by existing tools, e.g., FACT++. According to developpers, local reflexivity was relatively easy to implement [*TOOLS*] – for any individual x that must have a relationship along a reflexive property, an appropriately labelled edge <x, x> has been added to the model.

Use cases

Use Case #5 Use Case #3

2.2.2 F5: Qualified cardinality

While OWL 1 allows for the definition of persons that have at least three children, specifying the subclass of persons that have at least three children who are Girls is impossible. Expressing the latter class requires the qualification of the target of the property *hasChildren* (*atleast 3 hasChildren Girl*).

Feature

OWL 2 allows to assert minimum, maximum or exact qualified cardinality restrictions on object or data properties by means of the new constructs *MinCardinality MaxCardinality ExactCardinality*. A qualified cardinality restriction (QCR) defines a restriction on the number of instances of the property and on its class or data range. The addition that *qualified* cardinality restriction brings to the initial OWL 1 constructs for cardinality restrictions is to allow defining restrictions not only on the number of instances of the property but also on the *class or data range* of the instances. In OWL 2, *both* qualified or unqualified cardinality restrictions are possible. An unqualified cardinality restriction is simply equivalent to a qualified one where the restricting class is owl:Thing. [Syntax] [Semantics]

2.2.2.1 Object Property Cardinality Restrictions

The class expressions **ObjectMinCardinality**, **ObjectMaxCardinality**, and **ObjectExactCardinality** defined using a *MinCardinality*, *MaxCardinality* or *ExactCardinality* restriction on an object property denotes the set of objects that are connected via the given object property to at least, at most, or exactly the given number of individuals of the given class:

- **Minimum Cardinality**

```
ObjectMinCardinality := 'MinCardinality' '(' nonNegativeInteger
ObjectPropertyExpression [ ClassExpression ] ')'
```

- **Maximum Cardinality**

```
ObjectMaxCardinality := 'MaxCardinality' '(' nonNegativeInteger
ObjectPropertyExpression [ ClassExpression ] ')'
```

- **Exact Cardinality**

```
ObjectExactCardinality := 'ExactCardinality' '(' nonNegativeInteger
ObjectPropertyExpression [ ClassExpression ] ')'
```

Example:
- HCLS

The following examples are some examples of Object Property Cardinality
Restrictions from Use Cases among many in HCLS.

`ExactCardinality( 1 hasDirectPart FrontalLobe ) (UC#1)`	Class of objects having exactly one *direct part* of type *frontal lobe*.
`MinCardinality( 5 hasDirectPart owl:Thing )`	Class of objects having at least 5 *direct part*.

While in OWL 1 it was only possible to express that a Brain Hemisphere has at
least 5 direct part but not that it has exactly one *direct part* of each type: *frontal,
parietal, temporal, occipital, limbic lobe*, as needed in *UC#1*), both statements
are possible in OWL 2 (see above).

`MaxCardinality( 3 boundedTo Hydrogen) (UC#3)`	Class of objects *bounded to* at most three different *Hydrogen*

- Automotive industry

`MaxCardinality( 5 hasPart Door ) (UC#4)`	Class of objects having atmost 5 *Door*
`ExactCardinality( 2 hasPart RearDoor ) (UC#4)`	Class of objects having exactly 2 *RearDoor*

2.2.2.2 Data Property Cardinality Restrictions

- **Minimum Cardinality**

W3C Editor's Draft

```
DataMinCardinality := 'MinCardinality' '(' nonNegativeInteger
DataPropertyExpression [ DataRange ] ')'
```

- **Maximum Cardinality**

```
DataMaxCardinality := 'MaxCardinality' '(' nonNegativeInteger
DataPropertyExpression [ DataRange ] ')'
```

- **Exact Cardinality**

```
DataExactCardinality := 'ExactCardinality' '(' nonNegativeInteger
DataPropertyExpression [ DataRange ] ')'
```

Example:
- HCLS

```
MaxCardinality( 1 hasSSN )
```
Each individual has at most
one Social Security Number

Theoretical Perspective

As already said above, qualified cardinality restrictions are present in the SROIQ description logic underlying OWL 2 since they were required in various applications e.g.; [*Medical Req*] [*Little Web*] and did not pose theoretical or practical problems to be added [*SHOIQ*]. It was known from a long time that resulting logic is decidable and QCR was already supported by DAML+OIL, the predecessor of OWL 1.

Implementation Perspective

QCRs do not pose implementation problem either. It has been successfully implemented both in earlier editor, e.g.; OilED, and reasoner, e.g., FACT++, that already processed ontology with QCRs, before OWL 1 recommendation. Current versions of tools under development for OWL 2, e.g.; Protégé 4, FACT++, PELLET, RACER, KAON2 also deals with QCRs [*TOOLS*] [*OWL API*].

Use cases

Use Case #1 Use Case #2 Use Case #3, Use Case #4 Use Case #8

2.2.3 F6: Reflexive, Irreflexive, Asymmetric

While OWL 1 allows to assert that an object property is symmetric or transitive, it is impossible to assert that it is reflexive, irreflexive or assymetric. In mereology, the

partOf relation is defined to be transitive (if x is a part of y and y is a part of z, then x is a part of z), reflexive (every object is a part of itself), and antisymmetric (if an object has a part which in turn has part itself, then they are the same). Many applications, particularly those where it is necessary to describe complex structures such as life science applications, require extensive use of part-whole relations, axiomatized according to these principles. Similarly, other relations encountered in ontology modeling require similar axiomatizations, possibly with different sets of characteristics (see, e.g., [*OBO*] [*RO*]). Examples include proper part of and locative relations (typically transitive and irreflexive), causal relations (typically transitive and irreflexive) and membership relations (typically irreflexive). Thus as illustrated by many use cases new charactersitics of properties, Reflexivity, Irreflexivity, Asymmetry, is a strong requirement.

2.2.3.1 Reflexive Property

Feature

A reflexivity axiom asserts that a given object property is reflexive that is, the property holds for all the individuals, or in mathematical notation, this is:

$\forall x \; x \; R \; x$

[Syntax] [Semantics]

ReflexiveObjectProperty := 'ReflexiveProperty' '(' { **Annotation** } **ObjectPropertyExpression** ')'

Example:
  • HCLS

| ReflexiveProperty(*sameBloodGroup*) (*UC#9*) | Everybody has the same blood group as himself. |
| ReflexiveProperty(*part_of*) (*UC#2*) | Everything is part_of itself |

Note: there are different interpretations of the mereological relations. For example OBO (Use Case #5) states that *part_of* is reflexive while the mereological relation *anatomicalPartOf* between anatomical entities is asserted to be irreflexive in Use Case #1.

W3C Editor's Draft

2.2.3.2 Irreflexive Property

An irreflexivity axiom asserts that a given property is irreflexive, that is the property does not hold for any individual, or in mathematical notation, this is:

$\forall x \neg (x\ R\ x)$

[Syntax] [Semantics]

IrreflexiveObjectProperty := 'IrreflexiveProperty' '(' { **Annotation** } **ObjectPropertyExpression** ')'

Example:
  • HCLS

`IrreflexiveProperty(` *proper_part_of* `)` (*UC#5*)	Nothing can be proper_part of itself.
`IrreflexiveProperty(` *boundedBy* `)` (*UC#1*)	Nothing can be bounded by itself.

  • Earth and Space

`IrreflexiveProperty(` *flowsInto* `)`(*UC#6*)	Nothing can flow into itself.

Note: the given examples corresponds to the statements about mereological and topological properties *anatomicalPartOf boundedBy* in the given Use Cases, e.g.; Use Case #1. But other applications may use these terms for properties with different characteristics.

2.2.3.3 Asymmetric Property

An assymetry axiom asserts that a given property is assymetric that is, if the property holds between individuals x and y, then it cannot hold between y and x, or in mathematical notation, this is:

$\forall x \ \forall y \ (x\ R\ y) \Rightarrow \neg (y\ R\ x)$

[Syntax] [Semantics]

AsymmetricObjectProperty := 'AsymmetricProperty' '(' { **Annotation** } **ObjectPropertyExpression** ')'

W3C Editor's Draft

Example:

- HCLS

```
AsymmetricProperty(                    The property proper_part_of
proper_part_of )(UC#8)                 is asymmetric.
```

Theoretical Perspective

see F4 Theoretical Perspective above

Implementation Perspective

This was similar to the approach taken to support local reflexivity, and was relatively straight forward. Implementation of the algorithms for irreflexive and asymmetric properties is based on the previously implemented extensions for local reflexivity and disjoint properties, and essentially 'came for free' [*TOOLS*].

Use cases

Use Case #5 Use Case #6 Use Case #8

2.2.4 F7: Disjoint properties

While OWL 1 provides means to state the disjointness of classes, it is impossible to state that properties are disjoint, that is OWL 1 does not provide means to assert that if the same pair of individuals is related by two different properties among a given set of properties, then the ontology is inconsistent.

Feature

OWL 2 provides the new construct *DisjointProperties* for asserting that several properties are incompatible (exclusive).

A disjoint properties axiom takes a set of object properties and states that all properties from the set are pair-wise disjoint; that is, no pair of individuals can at the same time be connected by two different properties of the set. [Syntax] [Semantics]

```
DisjointObjectProperties := 'DisjointProperties' '(' { Annotation
} ObjectPropertyExpression ObjectPropertyExpression {
ObjectPropertyExpression } ')'
```

Example:

- HCLS

W3C Editor's Draft

```
DisjointProperties( connectedTo
contiguousTo ) (UC#1)
```
connectedTo and
contiguousTo properties are
exclusive

Note: According to the definition of these properties in Use Case #1, when two anatomical entities are linked via an actual third anatomical entity, they are stated to be connected. On the opposite, when they are not related by an actual entity but are *adjacent* via a *conventional* entity, they are said to be *contiguous*. Consequently, two parts cannot be at the same time connected and contiguous.

Theoretical Perspective

see F4 Theoretical Perspective above

Implementation Perspective

In FACT++, the processing of disjoint properties was split into static and dynamic parts. A dynamic analysis is applied to nodes that are merged during the reasoning process, and all other cases are handled by static analysis [*TOOLS*].

Use cases

Use Case #1 Use Case #2 Use Case #3

2.2.5 F8: Property chain inclusion

OWL 1 does not provide means to define properties as a composition of other properties. OWL 1 enables only the propagation of values via one property in asserting it transitive, but does not provide means to propagate a property (e.g.; *isLocatedIn*) along another property (*e.g.; partOf*), which is a very common requirement, specially in Life Sciences. Users cannot define properties as a chain of relations (as hasUncle).

Feature

OWL 2 allows to chain several object properties by means of the new construct *PropertyChain*.

A **propertyExpressionChain** expression defines a chain between several object properties by means of *PropertyChain*. [Syntax] [Semantics]

```
propertyExpressionChain := 'PropertyChain' '('
ObjectPropertyExpression  ObjectPropertyExpression {
ObjectPropertyExpression } ')'
```

W3C Editor's Draft

When used together with `SubPropertyOf`, an expression defined by `PropertyChain` provides a means to represent some types of rules. It allows in particular to express the propagation of one property along another property, e.g.; the propagation of a property from parts to whole, which is a very common requirement, specially in Life Sciences. However, the ontology set of axioms (involving SubObjectPropertyOf axioms with Property Chains) must satisfy some "Global Restrictions" in order to prevent cyclic definitions and to keep the language decidable.

In short, an axiom `SubPropertyOf( PropertyChain( p₁ ... pₙ ) p )` states that if an individual `x` is connected with an individual `y` by a chain of object properties p_1, ..., p_n, then `x` is also connected with `y` by the object property `p`. Such axioms are also known as *complex role inclusions* [*SROIQ*]. The full exact syntax of Property chain inclusion axioms is more precisely the following:

propertyExpressionChain := 'PropertyChain' '('
ObjectPropertyExpression ObjectPropertyExpression {
ObjectPropertyExpression } ')'
subObjectPropertyExpression := **ObjectPropertyExpression** |
propertyExpressionChain
SubObjectPropertyOf := 'SubPropertyOf' '(' { **Annotation** }
SubObjectPropertyExpression ObjectPropertyExpression ')'

Example:
  - HCLS

```
SubPropertyOf( PropertyChain(
locatedIn partOf ) locatedIn )
(UC#7)
```

If `x` is *locatedIn* `y`, and `y` is *partOf* `z`, then `x` is *locatedIn* `z`; for example a disease located in a part is located in the whole.

Theoretical Perspective

OWL 2 is based on SROIQ which offers complex role inclusion axioms that include axioms of the form R ∘ S < R or S ∘ R < R to express propagation of one property along another one, which have proven to be very useful in particular for biomedical ontologies. SROIQ allows also complex axioms of the form $S_1 \circ S_2 \circ, ..., \circ S_n$ < R under a certain condition of *Regularity* that prevents a role hierarchy from containing cyclic dependencies.

An OWL 2 `PropertyChain` expression used within a `SubPropertyOf` axiom provides a means to represent some types of rules while remaining decidable under special restrictions that is, provided that the Global Restrictions on Axioms

given Section 11 of the Syntax document are respected by the properties defined in the ontology.

Implementation Perspective

The most challenging aspect of implementing an OWL 2 reasoner, was the implementation of property chain inclusion axioms. Several optimisations were necessary to ensure acceptable reasoner performance [*TOOLS*].

To conclude: Current versions of tools under development for OWL 2, e.g.; Protégé 4, FACT++, PELLET, RACER, deals with all the features above [*TOOLS*] [*OWL API*]. In terms of extending existing OWL 1.0 reasoners and editing tools to cope with OWL 2, it has been found that the effort required to implement such extensions was perfectly acceptable, and minimal in comparison to the task of developing such tools from scratch [*TOOLS*].

Use Cases

Use Case #1 Use Case #5 Use Case #7 Use Case #8

2.2.6 F9: Key

OWL 1 does not provide means to define keys. However, keys, aka inverse functional datatype properties (but limited to named individuals), are clearly of vital importance to many applications in order to uniquely identify individuals of a given class by values of (a set of) key properties.

Feature

OWL 2 allows to define Database style keys for a given class by means of the new construct *HasKey*. A **HasKey** axiom states that each named instance of a class is uniquely identified by a (data or object) property or a set of properties that is, if two (named) instances of the class coincide on all the values of key properties, then these two individuals are the same. [Syntax] [Semantics]

```
HasKey := 'HasKey' '(' { Annotation } ClassExpression
ObjectPropertyExpression | DataPropertyExpression {
ObjectPropertyExpression | DataPropertyExpression } ')'
```

Example:
- HCLS

`HasKey( a:RegisteredPatient a:hasWaitingListN )`	Each registered patient on the ABM national organ waiting list, is uniquely

```
ClassAssertion(
a:RegisteredPatient
a:ThisPatient )
PropertyAssertion(
a:hasWaitingListN a:ThisPatient
"123-45-6789" )
```

identified by his waiting list number (*UC#9*)

a:ThisPatient is an instance of *a:RegisteredPatient*.

a:ThisPatient has the number "123-45-6789" on the waiting list.

In this example, since *a:hasWaitingListN* is the key for the class *a:RegisteredPatient*, the number "123-45-6789" uniquely identifies *a:ThisPatient*. The axiom HasKey(*a:RegisteredPatient a:hasWaitingListN*) does not state that each registered patient has at least one value of *a:hasWaitingListN*, but only that two different patients who have got a number assigned cannot have the same number on the waiting list: if the values of *a:hasWaitingListN* were the same for two instances of RegisteredPatient, these two individuals would be equal.

```
HasKey( a:Transplantation
a:donorId a:recepientId a:
ofOrgan )
```

Each Transplantation is uniquely identified by a donor, a recipient, and an organ (*UC#9*)

A set of several properties is needed to identify a transplantation: a donor may provide several organs, e.g., a kidney and a liver, to a single person, or the same organ, e.g., a kidney, to two recipients, or different organs to different recipients.

Theoretical Perspective

Keys in general have the following properties [Easy Keys]:

1. Missing key values raise an error (optional)
2. Functionality constraints on keys (optional)
3. If X and Y have the same key values, then X=Y.

The first feature is not expressible directly in first order logic. The second feature, *Functionality*, can be expressed in OWL (both for data and object properties). The third feature, in its general form, can lead to unfeasibly difficult reasoning in OWL (given what is currently known and anticipated). However, a more restricted form limited to named individuals, can be expressed in first order logics as a DL Safe rule e.g., the following DL safe rule expresses that *keyProperty* (whatever data or object property) is a key property:

keyProperty(X, Z), keyProperty(Y, Z) implies X = Y.

(for more details see Semantics for Key)

W3C Editor's Draft

Implementation Perspective

DL-safe rules can be added to description logic reasoners without major problems. This, for example, has been done in KAON2, and KAON2 has been successfully applied to a number of practical problems. Furthermore, DL-safe rules can be easily added to the hypertableau reasoning approach implemented in the HermiT reasoner. Since keys can be expressed as DL-safe rules, they can, as a consequence, be easily implemented in the main-stream OWL reasoners.

Use cases

Use Case #2 Use Case #7 Use Case #9

2.3 Extended datatypes capabilities

2.3.1 F10: Unary Datatype

- OWL 1 provides very limited expressive power for concrete values, such as integers and strings. The datatypes are modeled after XML Schema, which provides a rich set of datatypes; however, only xsd:string and xsd:integer are normative (in OWL DL), which is often not sufficient for applications.
- Furthermore, in OWL 1 it is possible to express restrictions on datatype properties qualified by a unary datatype. For example, one could state that every French citizen has a Passeport Number which is an xsd:string. But it is not possible to restrain the range of a datatype. For example, one could not say that adults have an age greater than 18 years, that pressure is in the range of 1030mb to 1035mb. As illustrated by the use cases below, datatype restrictions have been required by many users to allow such statements.

Feature

OWL 2 provides new capabilities for unary datatypes, supporting a richer set of datatypes and also facets for restricting the range of built-in datatypes:

- OWL 2 datatypes allow representing a) various kinds of numbers, e.g., integer, real, double, float, decimal, both adding support of a wider range of XML Schema Datatypes (double, float, decimal, positiveInteger etc.) and providing special datatypes e.g., owl:real b) strings with (or without) a Language Tag (using the rdf:text datatype) c) Boolean values, Binary Data, URIs, Time Instants, etc. Syntax
- It is possible to specify restrictions on datatypes by means of *facets* that restrain the range of values allowed for a given unary datatype. Constraining facets used in a *DatatypeRestriction* are one of the allowed facets defined in the Datatype Maps section, e.g., length, minLength, maxLength, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive. They are denoted by an URI. The restriction value used in a *DatatypeRestriction* is a literal listed in the Datatype Maps section. [Syntax] [Semantics]

W3C Editor's Draft

```
DatatypeRestriction := 'DatatypeRestriction' '(' Datatype
constrainingFacet restrictionValue { constrainingFacet restrictionValue }
')'
```

Example:
  • HCLS

`DatatypeRestriction(xsd:integer minInclusive 18) (UC#9)`	new datatype with a lower bound of 18 on the XML Schema datatype xsd:integer

This datatype is needed for example to define patients under 18 (child) who depend on pediatric services at hospital while over 18 (adult) depend on adult services.

Theoretical Perspective

See OWL Datatypes: Design and Implementation [*Datatype*].

Implementation Perspective

See Unary Datatypes Implementation Report and implementation matrix for status on built-ins listed in OWL 1.0 docs and facets listed in member submissions.

Use Cases

Use Case #9 Use Case #11 Use Case #12 Use Case #18 Use Case #19

Editor's Note: The current specification does not define data ranges of arity more than one; however as the syntax allows data ranges of arity n, it provides a "hook" allowing implementations to introduce extensions such as comparisons and arithmetic. - should be updated

2.3.2 F11: N-ary datatype

In OWL 1 it is not possible to represent relationships between values for one object, e.g., a square is a rectangle whose length equals width (nor relationships between values for different objects e.g., people who are older than their boss); N-

ary datatypes have been required by users to allow such statements, for example in the use cases listed below.

Feature

OWL 2 extends the unary datatypes of OWL 1 to n-ary datatypes, allowing to compare values of data properties for a given object, for example the admisssion temperature of a patient to his current temperature. It allows more generally to use n-ary datatypes such as those built from linear expressions.

Example:
  • HCLS

```
AllValuesFrom(                          individuals whose
admissionTemperature                    admissionTemperature is
currentTemperature inferior)            inferior to
(UC#11)                                 currentTemperature.
```

Theoretical Perspective

N-ary can be used to compare values of data properties for a *single* given object e.g., in any patient, the systolic blood pressure is always greater or equal than the diastolic blood pressure. They cannot be used to compare values of data properties for *different* objects (e.g., defining the class of individuals whose body weight is superior than their mother's) which leads to undecidability.

Implementation Perspective

These features are already supported by popular existing tools, e.g. facets were already supported by Protégé editor supported before, and e.g., Pellet reasoner supports reasoning with all the built-in datatypes defined in XML Schema plus any user-defined data ranges that extend numeric or date/time derived types.

Use Cases

Use Case #10 Use Case #11

2.4 Simple metamodeling capabilities

2.4.1 F12: Punning

For a class Eagle of individuals it might be wanted to represent the two following different statements:

  • Metadata (metalogical information) about the class Eagle that would say that class Eagle was created by "C. Welty" in 1994, ID. This was already possible in OWL 1 and is still possible in OWL 2 by means of annotation assertions that state that the URI identifying Eagle is annotated with the

annotation value "C. Welty" by the annotation property *creator*, and with the value 1994 by the annotation property *year*).
- Statement about the Eagle specy as a whole, expressing that "eagles are listed in the IUCN Red List that is, Eagle denoting an individual of the RedListSpecies class. RedListSpecies is then a metaclass. Modeling with metaclasses is commonly called metamodelling.

In some applications, one would simply like to use the same term (name) for a class (e.g., Eagle viewed as a class) and for an instance (e.g., Eagle viewed as an individual of the Red List of species), whithout needing to have whole the inferences power of metamodelling. For example, a biomedical application may simply like to have a colummn in a database which data are names of gene classes, another one which data are molecular functions, e.g.; imported from the Gene Ontology. Such simple metamodelling capabilities is a common requirement in applications.

Feature

OWL 2 provides a simple form of metamodelling based on punning, that is the same name can be used for different types of entities, with certain restrictions, more percisely:

- the name used for an individual can also be used for a class, datatype, object property, data property or annotation property
- the name used for a class or a datatype can also be used for an individual, object property, data property or annotation property
- the name used for an object property or data property or annotation property can also be used for an individual, class or datatype

The same name cannot be used for an ObjectProperty and an DatatypeProperty or for a Class and a Datatype.

Example:
- Telecom

```
Declaration( Class( a:Person ) )     a:Person is declared to be a
(UC#13) (1)                          class
ClassAssertion( a:Service a:s1 )     a:s1 is an individual of
(2)                                  a:Service.
                                     the individual a:s1 is
PropertyAssertion( a:hasInput        connected by a:hasInput to
a:s1 a:Person )(3)                   the individual a:Person.
```

The same term 'Person' denotes both a class in (1) and an individual in (3). This is possible in OWL 2 thanks to *punning* (Class ↔ Individual).

- Collaborative environment (Wiki)

```
Declaration( Class(
a:Deprecated_Properties ) )
(UC#14)(1)
Declaration( ObjectProperty(
a:is_located_in ) ) (2)
ClassAssertion(
a:Deprecated_Properties
a:is_located_in ) (3)
```

a:Deprecated_Properties is declared to be a *Class*

a:is_located_in is declared to be an *ObjectProperty*

a:is_located_in is an individual of a:Deprecated_Properties.

The same term 'is_located_in' denotes both a property (2) and an individual (3). This is possible in OWL 2 thanks to *punning* (Property ↔ Individual).

Use Case #14 should also have been represented without metamodelling, with an annotation *deprecated property* on the property *a:is_located_in*.

- UML Design

```
Declaration( Class( a:Person ) )
Declaration( Class( a:Company )
) (UC#15) (1)
```

a:Person and a:Company are declared to be classes.

```
SubClassOf ( a:PersonCompany a:
Association) )(2)
```

a:PersonCompany denotes a subclass of an a: Association associating Person and Company.

```
PropertyDomain( a:PersonCompany
a:Person )(3)
```

The domain of the property a:PersonCompany is a:Person.

```
PropertyRange( a:PersonCompany
a:Company )(4)
```

The range of the property a:PersonCompany is a:Company.

```
The same term a:PersonCompany denotes both a class (2)
and an ObjectProperty(3 ; 4). This is possible in OWL
2 thanks to punning (Class ↔ObjectProperty).
```

Theoretical Perspective

According to OWL 1 Direct Model-Theoretic Semantics:

- (i) the vocabulary VC of classes names and VD of datatypes names are disjoint that is, the same name (URI) cannot be used for a class and a datatype;
- (ii) the vocabularies VDP of data property names, VIP of object property names, VAP of annotation property names, (plus VOP the set of URI references for the built-in OWL ontology properties) are pairwise disjoint thats is, the same name cannot be used for a data property, an object property and an annotation property.

Besides, in order to retain decidability OWL DL imposed additional conditions on the vocabulary:

- (iii) the sets VC of classes names and VIP of object property names and the set of individuals names are disjoint that is, the same name cannot be used for a property or a class and an individual.

In other words, metamodelling was not supported by OWL DL. On the opposite, OWL full did not impose this resctriction, but the style of metamodelling adopted in OWL full led to undecidability.

In OWL 2, following the proposal of simple [*Metamodelling*] based on punning, conditions (iii) have been relaxed while retaining decidability. Below the list of punning (at this time) allowed in OWL 2.

In the following
X | Y
Z | W
should be interpreted as *"a URI u used as an object of type X or Y can also be used as an object of type Z or W"*

- individual and :

class | datatype | object property | data property | annotation property

- a class or datatype:

individual | object property | data property | annotation property

- object property | data property | annotation property:

individual | class | datatype

Punning ObjectProperty ↔ DatatypeProperty and Class ↔ Datatype is forbidden, that is the same name cannot be used for an ObjectProperty and a DatatypeProperty or for a Class and a Datatype.

Implementation Perspective

Punning presents absolutely no problem for reasoning algorithms and can be implemented with minimal effort.

Use Cases

Use Case #12 Use Case #13 Use Case #14 Use Case #15

W3C Editor's Draft

2.5 Extended annotations

2.5.1 F13: <u>Annotation</u>

OWL 1 allows to associate (metalogical) information, such as a label or a comment, like a textual description of the entity, to each ontology entity. But it is also useful to associate metalogical information to axioms. For example, one might want to keep information about who asserted an axiom or when. Therefore, it has been required to extend annotations to allow for the annotation of both entities and axioms.

Feature

OWL 2 provides for annotations on ontologies, entities (such as a class or individual, including anonymous individuals), and axioms. Even annotations of annotations are possible. Annotations however, have no semantic meaning in OWL 2.

- An annotation value is either a literal (e.g., string, integer, or any other OWL datatype), a URI, or an anonymous individual.
- A URI or an anonymous individual can be annotated with an annotation value by an annotation property. More precisely, the assertion `AnnotationAssertion( P u v )` states that the URI or anonymous individual `u` is annotated with the annotation value `v` by the annotation property `P`.
- An axiom can be annotated with an annotation value by an annotation property. The axiom annotation `Annotation( P v )` states that the axiom is annotated with the annotation value `v` by the annotation property `P`.
- There are only three axioms that can be used on annotation properties: `AnnotationPropertyDomain`, `AnnotationPropertyRange` and `SubAnnotationPropertyOf` axioms. These special annotation axioms have no semantics in OWL DL, but the normal semantics in OWL Full via their mapping to the standard RDF vocabulary. <u>Syntax</u>

Below extracts of the syntax used for annotating URIs or anonymous individuals and axioms, for example SubClassOf axioms, respectively, next followed by the related examples (for the complete syntax see <u>Annotations</u> in the Syntax document):

AnnotationValue :`=` **AnonymousIndividual** | **URI** | **Literal**

W3C Editor's Draft

```
AnnotationSubject := URI | AnonymousIndividual
AnnotationAssertion := 'AnnotationAssertion' '(' axiomAnnotations
AnnotationProperty AnnotationSubject AnnotationValue ')'
```

```
Annotation := 'Annotation' '(' annotationAnnotations
AnnotationProperty AnnotationValue ')'
axiomAnnotations := { Annotation }
SubClassOf := 'SubClassOf' '(' axiomAnnotations
subClassExpression superClassExpression ')'
```

Example:
- HCLS

`AnnotationAssertion (Class(`*CARO: anatomical structure*`) hasId ( "0000003"^^xsd:integer )) (`*UC#5*`)`	The value of the annotation property *hasId* for the URI *CARO: anatomical structure* of the CARO ontology is the integer 0000003.
`SubClassOf( Comment("Middle lobe of lungs are necessary right lobe, left lung do not have middle lobe.")` *MiddleLobe RightLobe*`) (`*UC#2*`)`	The comment "Middle lobe of lungs are necessary right lobe." of this axiom explains why *MiddleLobe* is a subclass of *RightLobe*.
`SubPropertyOf(` *a:narrow_synonym a:synonym* `) (`*UC#5*`)`	Having a narrow synonym is a subproperty of having a synonym.

Gene Ontology or other OBO ontologies distinguish different kinds of synonyms: exact_synonym, narrow_synonym, broad_synonym.

Theoretical Perspective

Annotations do not raise any theoretical difficulty since they do not have (direct) semantics

Implementation Perspective

Since annotations do not affect the logical meaning of the ontology, they present no problem for reasoning algorithms.

Use Cases

Use Case #5 Use Case #12 Use Case #19

2.6 Other main innovative features

2.6.1 F14: Declarations

In OWL 1, an entity such as a class or an object property could be used in an ontology without any prior announcement, so there was no way of ensuring that entity names matched in different axioms. In practice, if an entity name was mistyped in an axiom, there was no way of catching the error. As a consequence, it was deemed desirable that entities in an ontology could be declared — that is, they could be explicitly listed as being a part of an ontology.

Feature

A declaration axiom states that an entity is part of the vocabulary of an ontology. A declaration also associates an entity type (class, datatype, object property, data property, annotation property, individual, or a combination) to the declared entity. Declarations are optional for the most part.Syntax

```
Declaration := 'Declaration' '(' axiomAnnotations Entity ')'
```

Properties and classes should be declared in an ontology. Furthermore, declarations are also used during ontology parsing to disambiguate the productions of various syntaxes.

In addition, the OWL 2 specification provides the notion of declaration consistency which, roughly speaking, checks whether each entity used in an ontology also occurs in at least one declaration.

Theoretical Perspective

Other than introducing the set of symbols used in an ontology, declarations do not affect the model-theoretic semantics of OWL 2 ontologies. Therefore, this feature incurs no major theoretical implications on OWL 2.

Implementation Perspective

Checking of typing constraints and the notion of declaration consistency can be implemented straightforwardly. Neither of these two checks requires reasoning services — that is, both checks operate on the axiom closure of an ontology.

W3C Editor's Draft

Use Cases

[Use Case #17](#)

2.6.2 F15: Profiles OWL 2 EL, OWL 2 QL, OWL 2 RL

Large Life Sciences ontologies like the FMA, NCI Thesaurus, SNOMED CT, Gene Ontology or other OBO ontologies are mainly concerned by scalability issues of the language and do not necessarily need the whole expressivity of OWL. On the other side, applications involving classical databases and also DBM system companies, are mainly concerned about interoperability of the language with standard relational DBMS. While other ones, such as the business rules community and rule companies, feel concerned about interoperability of the ontology language with rules and existing rule engines. Consequently, different profiles of language have emerged, been requested by different types of users - ontologists, DBMS or rule engine developpers - and correspond to various application scenarios:

- a scalable profile for large but (rather) simple ontologies that enables good time performance for ontology (TBox/schema) reasoning.
- a profile that can easily interoperate with relational database systems, useful for applications where scalable reasoning on large datasets is the most important task.
- a profile that can easily interoperate with rules engines and rule extended DBMS, useful for applications where query answering is the most important task.

Feature

OWL 2 defines several profiles, sublanguages with useful computational properties (e.g., reasoning complexity in range of LOGSPACE to PTIME) and implementation possibilities (e.g., fragments implementable using RDBs). These profiles are described in details in the [Profile](#) document.

OWL 2 EL

- Captures expressive power used by many large-scale ontologies, e.g.; SNOMED CT, the NCI thesaurus;
- Features include existential restrictions, intersection, subClass, equivalentClass, class disjointness, range and domain, object property inclusion (SubObjectPropertyOf), possibly involving property chains, and data property inclusion (SubDataPropertyOf)transitive properties, keys (HasKey) …;
- Missing features include value restrictions, cardinality restrictions (min, max and exact), disjunction and negation.

OWL 2 QL

- Captures expressive power of simple ontologies like thesauri, and (most of) expressive power of ER/UML schemas;

W3C Editor's Draft

- Features include limited form of existential restrictions, subClass, equivalentClass, disjointness, range and domain, symmetric properties, …;
- Missing features include existential quantification to a class (ObjectSomeValuesFrom), self restriction (ObjectHasSelf), nominals (ObjectHasValue)(ObjectOneOf),universal quantification to a class (ObjectAllValuesFrom), ObjectMinCardinality, ObjectExactCardinality), disjunction (ObjectUnionOf, DisjointUnion) etc. cf. the Profile document for an exhaustive list missing features.
- Can be implemented on top of standard relational database.

OWL 2 RL

- Includes support for most OWL 2 features;
- But with restrictions placed on the syntax, for example it does not include existential on the right hand side of axioms (which often occurs in Life Sciences ontologies, e.g., SNOMED). Standard semantics only apply when they are used in a restricted way;
- Can be implemented on top of rule extended DBMS e.g., SQL (see Implementation Perspective).

Theoretical Perspective

OWL 2 EL is the maximal language for which reasoning, including query answering, is known to be worst-case polynomial. It is related to the theory of [*EL++*] [*EL++ Update*].

OWL 2 QL is the maximal language for which reasoning, including query answering, is known to be worst case logspace (same as DB).

OWL 2 RL allows for polynomial reasoning (consistency, classification, and instance checking) using rule-based technologies. It is related to the theory of DLP [*DLP*] and pD* [*pD**].

Implementation Perspective

OWL 2 EL enables efficient implementations [*CEL*]. Reasoning, including query answering, is known to be worst-case polynomial [*EL++*] [*EL++ Update*].

- E.g., CEL is the first reasoner for the description logic EL+; CEL implements a polynomial-time algorithm.

OWL 2 QL can be implemented on top of standard relational database: the data can be left in the DBs, and query answering simply uses the ontology to rewrite the queries into equivalent SQL queries against the source DBs.

OWL 2 RL can be implemented on top of rule extended DBMS.

- E.g., Oracle's OWL Prime implemented using forward chaining rules applied to triples of the RDF serialization in Oracle 11g (see ORACLE 11gR1 OWL Prime.) [*OWL Prime*].

Use Cases

Use Case #2 Use Case #3 Use Case #4 Use Case #8 Use Case #16

2.7 Other Differences From OWL 1

OWL 2 exhibits some other differences to OWL 1 in its conceptual design and syntax.

2.7.1 Dropping the Frame-Like Syntax

OWL 1 provides a frame-like syntax that allows several features of a class, property or individual to be defined in a single axiom at once. This may cause problems in practice. First, it bundles many different aspects of the given entity into a single axiom. While this may be convenient when ontologies are being designed, it is not convenient for manipulating them programmatically. In fact, most implementations of OWL 1 break such axioms apart into several "atomic" axioms, each dealing with only a single feature of the entity. However, this may cause problems with round-tripping, as the structure of the ontology may be destroyed in the process. Second, this type of axiom is often misinterpreted as a declaration and unique "definition" of the given entity. In OWL 1, however, entities may be used without being the subject of any such axiom, and there may be many such axioms relating to the same entity. OWL 2 has addressed these problems in several ways. First, the frame-like notation has been dropped in favor of a more fine-grained structure of axioms: each axiom describes just one feature of the given entity. Second, OWL 2 provides explicit declarations, and an explicit definition of the notion of structural consistency.

Example:

The following is an example of an OWL 1 frame-like axiom.

```
ObjectProperty( a:partOf
inverseOf( a:containedIn )
inverseFunctional transitive

Annotation( rdfs:comment
"Specifies that an object is a
part of another object."))
```

The property *partOf* has an inverse property named *containedIn*, is an inverse functional and transitive property, and has the human-friendly comment "Specifies that an object is a part of another object."

W3C Editor's Draft

Example:

This can be represented in OWL 2 using the following axioms.

`Declaration( ObjectProperty(` `a:partOf ) )`	Declaration of the object property *partOf*

`AnnotationAssertion(`
`rdfs:comment a:partOf "Specifies`
`that an object is a part of`
`another object." )`
	A comment on the property *partOf* is "Specifies that an object is a part of another object."

`InverseProperties( a:partOf` `a:containedIn )`	*a:partOf* and *a:containedIn* are inverse properties
`InverseFunctionalProperty(` `a:partOf )`	*partOf* is an inverse functional property
`TransitiveProperty( a:partOf )`	*partOf* is a transitive property

Although OWL 2 is more verbose, this is not expected to lead to problems given that most OWL ontologies are created using ontology engineering tools.

2.7.2 Inverse Property Expressions

In OWL 1, all properties are atomic, but it is possible to assert that one object property is the inverse of another. In OWL 2, it is not necessary to give a name to an inverse property, as property expressions such as `InverseOf( a:hasPart )` can be used in class expressions. However, names can still be given to inverse properties, if desired.

Example:

The following is an example of an OWL 1 inverse property axiom.

`ObjectProperty( a:hasPart` `inverse a:isPartOf )`	*a:hasPart* has an inverse property named *a:isPartOf*.

Example:

This can be represented in OWL 2 using the following axiom.

`EquivalentProperties( a:hasPart` `InverseOf( a:isPartOf ) )`	*a:isPartOf* is the same as the inverse property of *a:hasPart*

As such axioms are quite common, OWL 2 provides the following syntactic shortcut as well.

W3C Editor's Draft

```
InverseProperties( a:hasPart          a:hasPart and a:isPartOf are
a:isPartOf )                          inverse properties
```

2.7.3 Anonymous Individuals

In OWL 1, anonymous individuals were introduced as individuals without identifiers.

Example:

```
Individual(value( a:city              This axiom does not contain
a:Quahog ) value( a:state a:RI        an individual name, so the
))                                    introduced individual is an
                                      anonymous individual.
```

In contrast, in OWL 2 anonymous individuals are identified using node IDs.

Example:

```
                                      This axiom introduces an
                                      explicit anonymous
PropertyAssertion( a:city _:1         individual _:1 representing
a:Quahog )                            an individual identifying
                                      some 'location' which city is
                                      Quahog
```

```
                                      This axiom introduces an
                                      explicit anonymous
PropertyAssertion( a:state _:1 a:RI ) individual _:1 representing
                                      an individual which state is
                                      RI
```

3 Tables

3.1 Use Cases ↔ Requirements

Use Case	Disjoint Union	Disjoint Classes	Negative property	Local reflexivity	Qualified Cardinality	Reflex., Irrefl., Asymm.	Disjoint properties	Property chain	Keys	Datatype restrictio
UC#1	*	*	-	-	*	-	*	*	-	-
UC#2	*	*	-	-	*	-	*	-	*	-
UC#3	*	*	-	*	*	-	-	-	-	-
UC#4	*	-	-	-	*	-	-	-	-	-

W3C Editor's Draft

UC#5	-	-	-	*	-	*	-	*	-	-
UC#6	-	-	-	-	-	*	-	-	-	-
UC#7	-	-	-	-	-	-	-	*	*	-
UC#8	-	-	-	-	*	*	-	*	-	-
UC#9	-	-	*	-	-	-	-	-	*	*
UC#10	-	-	-	-	-	-	-	-	-	-
UC#11	-	-	-	-	-	-	-	-	-	*
UC#12	-	-	-	-	*	-	-	-	-	*
UC#13	-	-	-	-	-	-	-	-	-	-
UC#14	-	-	-	-	-	-	-	-	-	-
UC#15	-	-	-	-	-	-	-	-	-	-
UC#16	-	-	-	-	-	-	-	-	-	-
UC#17	-	-	-	-	-	-	-	-	-	-
UC#18	-	-	-	-	*	-	-	-	-	*
UC#19	-	-	-	-	-	-	-	-	-	*

3.2 Use Cases / Features / Examples

This table synthesises the relations between the Use Cases of the Appendix and the Features listed in Section 2. Each use case may point to several features required. But one particular feature and a corresponding example have been selected in each Use Case for illustration of this feature. Thus in each line, the feature selected and the related reference from which the example is issued are displayed higlighted in bold. (This is not to say that the other requirements of the line which are abbreviated are less important). Each line is tagged by a scope (domain/user profile).

Use Case	Feature(s) required	Scope	Example	References
UC#1	**DisjointUnion** F2 F5 F7 F8 F11	HCLS	DisjointUnion(*Lobe FrontalLobe ParietalLobe TemporalLobe OccipitalLobe LimbicLobe*) *Lobe is a disjoint union of FrontalLobe FrontalLobe ParietalLobe TemporalLobe OccipitalLobe LimbicLobe*	[***MEDICAL REQ***] [*Ontology with rules*] [*Brain Imaging*]
UC#2	**DisjointClasses** F1 F2 F5 F7 F9	HCLS	DisjointClasses(*LeftLung RightLung*) *a Lung cannot be LeftLung and RightLung*	[***FMA***]

W3C Editor's Draft

UC#3	**Local reflexivity** F1 F2 F5 F15	HCLS	HasSelf(*isConnectedTo*) *class of all individuals that are connected to themselves*	[***Chemistry***]
UC#4	**Qualified Cardinality** F1 F15	Auto	ExactCardinality(2 *hasPart RearDoor*) Class of objects having exactly 2 *RearDoor*	[***Auto***]
UC#5	**Asymmetric property** F6 F8 F13	HCLS	AsymmetricProperty(*proper_part_of*) *if p is a proper part of q then q cannot be a proper part of p*	[***OBO***] [*RO*] [*OBO2OWL*]
UC#6	**Irreflexive property**	Earth &Space	IrreflexiveProperty(*flowsInto*) Nothing *flowsInto* itself.	[***Ordnance***]
UC#7	**Property chain** F9	HCLS	SubPropertyOf(PropertyChain(*locatedIn partOf*) *locatedIn*) anything *locatedIn* a part is *locatedIn* the whole, e.g. a disease.	[***SNOMED REQ***]
UC#8	**Reflexive property** F5 F8	HCLS	ReflexiveProperty(*partOf*) [*Part Whole*] argues about *partOf* as a reflexive property e.g. that a "car is a part of a car".	[***Part Whole***]
UC#9	**Negative property** F9 F10	HCLS	NegativePropertyAssertion(*hasAge ThisPatient* 5^^*xsd:integer*) *This patient* is not five years old.	[***Transplant Ontology***] [*Agence Biomedecine*]
UC#10	**N-ary**	HCLS	AllValuesFrom(*testDate enrollmentDate* x > y + 30)	[***N-ary***]

W3C Editor's Draft

			individuals whose *testDate* is superior to their *enrollmentdate* + 30.	
UC#11	**N-ary** F10	HCLS	AllValuesFrom(*admissionTemperature currentTemperature* x < y) individuals whose *admissionTemperature* is inferior to *currentTemperature*.	[***N-ary***]
UC#12	**Datatype restriction** F5 F12 F13	Tool	DatatypeRestriction(xsd:integer minInclusive 18) new datatype with a lower bound of 18 on the XML Schema datatype xsd:integer, e.g. to describe the class *Adult*.	[***Protege***]
UC#13	**Metamodelling**	Telecom	Declaration(Class(*a:Person*)) *a:Person* is declared to be a class ClassAssertion(*a:Service a:s1*) *a:s1* is an instance of *a:Service* PropertyAssertion(*a:hasInput a:s1 a:Person*) *a:s1* has input *a:Person* *this is an example of punning for Class ↔ Individual.*	[***Web Service***] [*Punning*]
UC#14	**Metamodelling**	Wiki	Declaration(ObjectProperty(*is_located_in*)) *is_located_in* is declared to be an *ObjectProperty* ClassAssertion(*Deprecated_Properties is_located_in*) *is_located_in* is an individual of the class *Deprecated_Properties*	[***Wiki***] [*Punning*]

W3C Editor's Draft

			this is an example of punning for Property ↔Individual.	
UC#15	**Metamodelling**	Designer	`Declaration( Class( a:Person ) ) Declaration( Class( a:Company ) )` *a:Person* and *a:Company* are declared to be classes `SubClassOf ( a:PersonCompany a:Association) )` association between classes *a:Person* and *a:Company* `PropertyDomain( a:PersonCompany a:Person )` The domain of the property *a:PersonCompany* is *a:Person*. `PropertyRange( a:PersonCompany a:Company )` The range of the property *a:PersonCompany* is *a:Company*. *this is an example of punning for Class ↔ ObjectProperty.*	[**UML**] [*Punning*]
UC#16	**Profiles**	Designer	*This Use Case motivates a profile e.g., OWL QL, where conjunctive query answering is implemented using conventional relational database systems*	[**Who reads?**]
UC#17	**Declaration**	Tool	`Declaration( Class( a:Person ) )` *a:Person* is declared to be a class.	[**Syntax Problem**] [*TOOLS*] [*DIG2*] [*OWL API*]
UC#18	**Datatype** F5	Earth&Space	`DatatypeRestriction( xsd:integer minInclusive "18000"^^xsd:integer maxExclusive "19600"^^xsd:integer )` *The data range for atmosphere above 18000 [feet] and below 19600 [feet]*	[**VSTO**]
UC#19	**Annotation** F10	Earth&Space	`SubClassOf( Comment("data generated by the LogParser`	[**NCAR**]

| | | | | using the ObserverLog")
a:LogInformation a:Information)

This is an example of an
annotation of axioms | |

Legend:

F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
Disjoint Union	Disjoint Classes	Negative Property Assertion	Local reflexivity	Qualified Cardinality	Reflexive, Irreflexive, Asymmetric	Disjoint properties	Property chain inclusion	Keys	Datatype restriction	N-ary datatype

4 References

[OWL Use Cases and Requirements]
> *OWL Web Ontology Language: Use Cases and Requirements* Jeff Heflin, ed. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-webont-req-20040210/. Latest version available at http://www.w3.org/TR/webont-req/.

[SROIQ]
> *The Even More Irresistible SROIQ*. Ian Horrocks, Oliver Kutz, and Uli Sattler. In Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006). AAAI Press, 2006.

[SHOIQ]
> *A Tableaux Decision Procedure for SHOIQ*. Horrocks, I., and Sattler, U. In Proc. of 19th International Joint Conference on Artificial Intelligence (IJCAI 2005) (2005), Morgan Kaufmann, Los Altos.).

[Next Steps]
> *Next Steps to OWL*. B. Cuenca Grau, I. Horrocks, B. Parsia, P. Patel-Schneider, and U. Sattler. In Proc. of OWL: Experiences and Directions, CEUR, 2006.

[Syntax Problem]
> *Problem with OWL Syntax*. Boris Motik and I. Horrocks, OWLED 2006, 2006.
> *Reusing Ontological Background Knowledge in Semantic Wikis* *Denny Vrandecic, Markus Krötzsch, Proceedings 1st Workshop on Semantic Wikis. Budva, Montenegro, June 2006* .

[CEL]
> *CEL—A Polynomial-time Reasoner for Life Science Ontologies*. F. Baader, C. Lutz, and B. Suntisrivaraporn. In U. Furbach and N. Shankar, editors, Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06), volume 4130 of Lecture Notes in Artificial Intelligence, pages 287–291. Springer-Verlag, 2006.

[SNOMED EL+]
> *Replacing SEP-Triplets in SNOMED CT using Tractable Description Logic Operators*. B. Suntisrivaraporn, F. Baader, S. Schulz, K. Spackman, AIME 2007

W3C Editor's Draft

[EL++]
> *Pushing the EL Envelope*. Franz Baader, Sebastian Brandt, and Carsten Lutz. In Proc. of the 19th Joint Int. Conf. on Artificial Intelligence (IJCAI 2005), 2005.

[EL++ Update]
> *Pushing the EL Envelope Further*. Franz Baader, Sebastian Brandt, and Carsten Lutz. In Proc. of the Washington DC workshop on OWL: Experiences and Directions (OWLED08DC), 2008.

[DL-Lite]
> *Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family*. Diego Calvanese, Giuseppe de Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati. J. of Automated Reasoning 39(3):385–429, 2007.

[DLP]
> *Description Logic Programs: Combining Logic Programs with Description Logic*. Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. in Proc. of the 12th Int. World Wide Web Conference (WWW 2003), Budapest, Hungary, 2003. pp.: 48–57

[pD*]
> *Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary*. Herman J. ter Horst. J. of Web Semantics 3(2–3):79–115, 2005.

[OWLPrime]
> *Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle*. Zhe Wu Eadon, G. Das, S. Chong, E.I. Kolovski, V. Annamalai, M. Srinivasan, J. Oracle, Nashua, NH; Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on, pages 1239-1248, Cancun, 2008.

[Metamodeling]
> *On the Properties of Metamodeling in OWL*. Boris Motik. On the Properties of Metamodeling in OWL. Journal of Logic and Computation, 17(4):617–637, 2007.

[Datatype]
> *OWL Datatypes: Design and Implementation*. Boris Motik, Ian Horrocks, ISWC 2008, Karlsruhe, Deutshland, 2008.

5 Appendix: Use Cases

The following list of Use Cases is not exhaustive. Use Cases included in that bibliograpgy are only some among many that motivated the **OWL 2 new features** - whatever user/implementor/theoretical reasons - that appear, at this time, accepted by the Working Group for OWL 2. Other extensions (such as rules, default, etc.), possibly needed in the future, are indicated within brackets.

All use cases are presented using the following pattern: *Overview, Features, Example, Literature*. The *Overview* only gives a general description of the use case listed in this bibliography. Interested readers should see the paper available online

for more details (cf. the reference(s) within brackets [Literature]). *Features* lists several features required by the use case after the literature. *Example* points to a feature and short example which has been selected to illustrate a specific new feature of OWL 2. This same information can be seen in an abbreviated form in Table 3.2. [Literature] points to related papers available online.

5.1 Use Case #1 - Brain image annotation for neurosurgery [HCLS]

Overview: The system being developed concerns the preparation of surgical procedures in neurosurgery. Specifically, the aim is to assist a user in labelling the cortical gyri and sulci in the region surrounding a lesion whose resection is the primary objective. Providing anatomical landmarks, especially in eloquent cortex, is highly important for surgery. Brain image annotation is also useful for documentation of clinical cases, which then enables retrieval of similar cases for decision support in future procedures. A shared ontology of brain anatomy is also needed to integrate multiple distributed image sources indexed by anatomical features. This is useful for large-scale federated systems for statistical analysis of brain images of major brain pathologies.

Features: **Disjoint Union, Disjoint Classes, Qualified Cardinality Restrictions, Disjoint Properties, Property chain inclusion axioms, N-ary predicate, [Rules]**

Example for: **Disjoint Union**

- E.g.; Lobe is a disjoint union of FrontalLobe ParietalLobe TemporalLobe OccipitalLobe and LimbicLobe.

References: [*MEDICAL REQ*] [*Ontology with rules*] [*Brain Imaging*]

5.2 Use Case #2 – The Foundational Model of Anatomy [HCLS]

Overview: The Foundational Model of Anatomy (FMA) is the most comprehensive ontology of human 'canonical' anatomy. Anatomy plays a prominent role in biomedicine, and many biomedical ontologies and applications refer to anatomical entities. FMA is a tremendous resource in bioinformatics that facilitates sharing of information among applications that use anatomy knowledge. As its authors claim, the FMA is "a reference ontology in biomedical informatics for correlating different views of anatomy, aligning existing and emerging ontologies in bioinformatics..." . Anatomy, together with Gene and Disease reference ontologies constitute the backbone of the future Semantic Web for Life Sciences. But the FMA would benefit from new features of OWL to state that some properties are exclusive (e.g.; proper-part and bounded-by). Since many biomedical ontologies and applications refer to the FMA anatomical entities through cross-references, keys would also be useful.

Features: **Disjoint Union, Disjoint Classes, Qualified Cardinality Restrictions, Disjoint Properties, Keys**

Example for: **Disjoint Classes**

- E.g.; Nothing can be both a *Leftlung* and a *RightLung*.

References: [*FMA*]

5.3 Use Case #3 - Classification of chemical compounds [HCLS]

Overview: Functional groups describe the semantics of chemical reactivity in terms of atoms and their connectivity, which exhibit characteristic chemical behavior when present in a compound. In this use case the authors take a first step towards designing an OWL-DL ontology of functional groups for the classification of chemical compounds, and highlight the capabilities and limitations of OWL 1 and the proposed OWL 1.1 in terms of domain requirements. They also describe the application of expressive features in the design of an ontology of basic relations and how an upper level ontology can be used to guide the formulation of life science knowledge. They report on experiences to enhance existing ontologies so as to facilitate knowledge representation and question answering.

"Monocyclic and polycyclic ring structures are important parts of molecules that participate in several kinds of chemical reactions." A new OWL language feature such as self restriction, allowing some local reflexivity, would be helpful to describe the characteristic that ring structures are auto-connected, that is, rings are a kind of chemical structure which *isConnectedTo* to itself.

Features: **Disjoint Union, Disjoint Classes, Local reflexivity, Qualified Cardinality, Profiles**

Example for: **Local reflexivity**

- E.g.; A RingMolecule is a Molecule that connectedTo itself.

References: [*Chemistry*]

5.4 Use Case #4 - Querying multiple sources in an automotive company [Automotive]

Overview: Large companies often store information and knowledge in multiple information systems using various models and formats. The key objective in this use case is the retrieval of relevant information from multiple data and knowledge sources for a large automotive company. For this application a language with a profile facilitating querying multiple databases and easy representation of Parts Library ISO 13584 Standard (PLIB) ontologies of Products, which is particularly used for e-business catalogues, would be helpful.

Features: **Disjoint Union, Qualified Cardinality, Profiles (DB)**

Example for: **Qualified Cardinality**

- E.g.; the class of automobile having exactly 2 rear doors.

References: [*Auto*]

5.5 Use Case #5 - OBO ontologies for biomedical data integration [HCLS]

Overview: The Open Biomedical Ontologies (OBO) consortium is pursuing a strategy to facilitate the integration of biomedical data through their annotation using common controlled ontologies. Existing OBO ontologies, including the Gene Ontology, are undergoing coordinated reform, and new ontologies are being created on the basis of an evolving set of shared principles governing ontology development. The result is an expanding family of OBO ontologies designed to be interoperable and to incorporate accurate representations of biological reality. Within that effort the OBO ontology of relations is designed to define a set of basic relations with their semantics. OBO qualifies each relation using characteristics of being transitive, symmetric, reflexive, anti-symmetric. More generally OBO format offers constructs such as is_reflexive, is_symmetric, is_cyclic, is_anti_symmetric, etc. that are used in the OBO obtologies. Converting OBO ontologies requires the new OWL 2 property axioms reflexive, irreflexive, asymmetric to map corresponding OBO constructs, otherwise they should be transformed into annotations.

Features: **Local reflexivity, Reflexive, Irreflexive, Asymmetric, Property chain inclusion axioms, [Antisymmetric]**

Example for: **Asymmetric**

- E.g.; if p is a proper part of q then q cannot be a proper part of p.

References: [*OBO*] [*RO*] [*OBO2OWL*]

5.6 Use Case #6 – Spatial and topological relationships at the Ordnance Survey [Earth and Space]

Overview: Ordnance Survey is Britain's National Mapping Agency. It currently maintains a continuously updated database of the topography of Great Britain. The database includes around 440 million man-made and natural landscape features. These features include everything from forests, roads and rivers down to individual houses, garden plots, and even pillar boxes. In addition to this topographic mapping, entire new layers of information are progressively being added to the database, such as aerial photographic images which precisely match the mapping; data providing the addresses of all properties; and integrated transport information. For topological and spatial relationships, and in many other places, "we need to be able to say whether a property is reflexive, irreflexive, asymmetric or antisymmetric in order to capture the true intentions of our axioms".

Features: **Reflexive, Irreflexive, Asymmetric, [Antisymmetric]**

Example for: **Irreflexive**

- E.g.; Nothing flows into itself.

References: [*Ordnance*]

5.7 Use Case #7 - The Systematized Nomenclature of Medicine [HCLS]

Overview: The Systematized Nomenclature of Medicine, Clinical Terms (SNOMED CT) is a work of clinical terminology with broad coverage of the domain of health care, and it has been selected as a national standard for use in electronic health applications in many countries, including the U.S., U.K., Canada, Australia, Denmark, and others. SNOMED was originally published in 1976, while SNOMED CT became available in 2002 as a major expansion resulting from the merger of SNOMED RT with the U.K.'s Clinical Terms version 3. A major distinguishing feature differentiating it from prior editions is the use of description logic (DL) to define and organize codes and terms. Another major distinguishing feature of SNOMED is its size and complexity. With over 350,000 concept codes, each representing a different class, it is an order of magnitude larger than the next largest DL-based ontology of which we are aware.

Without property chain inclusion axioms, adoption of OWL by the SNOMED community would have required awkward workarounds with their attendant complications and complexities - effectively killing movement in that direction. With [them], we have a clear path to using OWL 2 for further development and integration with other biomedical ontologies. The required property chain inclusion axioms allow to encode inheritance of properties along another property, e.g., *part-of*, which is of utmost importance in anatomy. For example, with axioms such as `has-location ∘ proper-part-of < has-location` injury to finger can be inferred as injury to hand. As reported in [*SNOMED EL+*] by re-engineering SNOMED-CT in this way, the number of anatomical classes dropped from 54,380 to 18,125, and the time needed by the CEL reasoner [*CEL*] (version 0.94) from 900.15 seconds to 18.99 seconds.
Like the FMA, given the common use of cross-references between SNOMED and other biomedical ontologies via concepts ID, keys would be highly useful as well.

Features: **Property chain inclusion axioms, Keys, Profiles (OWL 2 EL)**

Example for: **Property chain**

- E.g.; anything located in a part is located in the whole

References: [*SNOMED REQ*]

5.8 Use Case #8 - Simple part-whole relations in OWL Ontologies [HCLS]

Overview: Representing part-whole relations is a very common issue for those developing ontologies for the Semantic Web. OWL does not provide any built-in primitives for part-whole relations (as it does for the subclass relation), but contains sufficient expressive power to capture most, but not all, of the common cases. The study of part-whole relations is an entire field in itself - "mereology" - this note is intended only to deal with straightforward cases for defining classes involving part-whole relations. Several extensions of whole needed for part-whole are discussed in this study, namely, needs of qualified cardinality restriction, reflexivity, propagation from parts to whole

Features: **Qualified cardinality restriction, Reflexivity, Property chain inclusion**

Example for: **Reflexive**

- E.g.; a frontal lobe is *part of* a brain memisphere or a car is *part of* a car

Note: according to the definition given in OBO, the whole is being considered as a part [*Part Whole*] but there are controversial opinions asserting that 'part of' is not reflexixe.

References: [*Part Whole*]

5.9 Use Case #9 - Kidney Allocation Policy in France [HCLS]

Overview: Allocation in France falls under the responsibility of the Agence de la biomedicine. It includes general rules such as: donor-recipient ABO blood group identity, unique registration on the national waiting list (a registration number is assigned at the registration of the waiting list which uniquely identifies a patient on the waiting list) and definition of some organ specific nation-wide allocation priorities. For each kidney recipient, minimal HLA matching and forbidden antigens can be specified. Pediatric recipients get a priority for pediatric donors. Kidneys are proposed by order of priority to (1) urgent patients, (2) patients with panel reactive antibodies level = 80% included in a specific acceptable antigen protocol or =1 HLA mismatch with the donor, then (3) zero mismatch patients, and (4) patients with low transplantation accessibility. This real-life application and allocation system show how distinguishing between adults and children has strong implications in health care: at hospital, patients under 18 (child) depend on pediatric services while over 18 (adult) depend on adult services; only children less than 16 years waiting for a transplant have a priority on the waiting list.

Features: **Negative Property Assertion, Datatypes restriction, Keys**

Example for: **Negative Property Assertion**

W3C Editor's Draft

• E.g.; This patient is not 5 years old.

References: [*Agence Biomedecine*] [*Transplant Ontology*]

Editor's Note: 1) Use Case #10 and Use Case #11 might be merged which implies renumbering or replacing Use Case #10 2) points to N-ary, see OWL WG ISSUE-127 related to n-ary data ranges Status

---

5.10 Use Case #10 – Eligibility Criteria for Patient Recruitment

Overview: This use case is based on an ongoing W3C task force on Clinical Observations Interoperability where the goal is to enable re-use and sharing of clinical data created in healthcare delivery in the Clinical Trials context. In particular the first application chosen to demonstrate feasibility of the interoperability approach is that of patient recruitment. In this case, a sample set of clinical trial protocols available from http://www.clinicaltrials.gov each of which contains a list of eligibility (inclusion and exclusion criteria). These eligibility criteria are used for identify eligible patients and potentially form conditions in a SPARQL query or could be represented as OWL classes. They also need to be mapped as per the discussion in the use case above. A list of requirements based on an analysis of these clinical trial protcols is available from http://esw.w3.org/topic/HCLS/ClinicalObservationsInteroperability?action=AttachFile&do=get&target=FunctionalRequirements_v1.xls

In particular, one of the clinical trials requires that the enrollment date of a clinical trial participant be within 30 days after the patient has been started on a particular therapy. This motivated the need for N-ary datatypes with inequality expressions.

Features: **N-Ary**

Example for: **N-ary datatype**

• E.g.; the enrollment date of a clinical trial participant should be within 30 days after the patient has been started on a particular therapy

5.11 Use Case #11 – Multiple UCs on datatype [HCLS]

Overview: [*N-ary*] presents many Use cases that would benefit from various datatype extensions

Features: **Datatypes restriction, N-Ary**

Example for: **N-ary datatype**

• E.g.; datatypes restrictions like intervals, or N-Ary datatype with inequality such as needed in Use Case #10.

References: [*N-ary*]

5.12 Use Case #12 – Protégé report on the experiences of OWL users [Tool]

Overview: [*Protege*] reported in 2005 on Protégé experiences with the development of OWL support, and on the experiences of the user community with OWL at that time. While the overall feedback from the community was positive, their experience suggested that there were considerable gaps between the user requirements, the expressivity of OWL, and users' understanding of OWL. To summarize, based on their experiences, Protégé developpers suggested a number of extensions to a future version of OWL namely, Integration of user-defined datatypes (esp. for numeric ranges), Qualified Cardinality Restrictions, Management of disjointness (owl:AllDisjoint), More flexible annotation properties (at least as best practices). This report underlined that one of the omissions in the OWL language that users complain about most often is poor representation of numeric expressions. Almost all groups, except for those developing traditional medical terminologies, sorely need to be able to express quantitative information. Typical examples include the length between 1mm and 2mm, age greater than 18 years, pressure in the range of 1030mb to 1035mb. Such range declarations are needed to classify individuals and to build class definitions such as *Adult*, and should therefore be supported by reasoners. User base points out that the current OWL datatype formalism is much too weak to support most real world applications and that many potential users therefore cannot adopt OWL. *"The user communities anxiously await an extension to the OWL specification to represent user-defined datatypes with XML Schema facets such as xsd:minInclusive."* It also points out some limitations related to annotations or metamodelling from an implementors perspective: *"Despite the value of annotation properties, in OWL DL, properties that are declared as annotation properties are greatly limited in so far that they can neither have range or domain constraints, nor can they be arranged in sub-property hierarchies. This type of information about a property enables tools to control the values that annotation properties can acquire. Without range constraints it is difficult to provide the user with appropriate input widgets. In a similar sense, it is often helpful to declare meta-classes so that classes can be categorized into types and different interfaces be pro-vided for each type. Currently, using these features means that the ontology will be forced into OWL Full."*

Features: **Qualified cardinality restriction, Datatypes restriction, Annotations, Metamodelling**

Example for: **N-ary datatype**

  • E.g.; adults are individuals whose age is greater than 18 years.

References: [*Protege*]

5.13 Use Case #13 - Web service modelling [Telecom]

Overview: People often want to use a class to specify the value of some property. An example originating at the University of Karlsruhe [*Web Service*] is in service modeling. Services are modeled as instances of the a:Service class. For each concrete service (i.e., for each instance of a:Service), the users wanted to state what the input to the service is. Here is an example of a service description:

(1) a:Service rdf:type owl:Class
(2) a:Person rdf:type owl:Class
(3) s1 rdf:type a:Service
(4) s1 a:input a:Person

s1 is an individual of the class a:Service due to (1) and (3), and a:Person is a class due to (2); hence, in (4) we have a relationship a:input between an individual and a class. Hence, you need some kind of metamodeling to solve this problem. One way would be that the name 'Person' may refer both to Person as a class and as an individual denoting Person as a whole (Class ↔ Individual)

Features: **Metamodelling**

Example for: **Simple metamodelling**

- E.g.; a class and an individual : *Person* may be used both for a class and an individual

References: [*Web Service*] [*Punning*]

5.14 Use Case #14 - Managing vocabulary in collaborative environments [Wiki]

Overview: It can be useful to relate schema elements (classes/properties) with each other in order to capture pragmatic relationships between them. An example observed in applications of Semantic MediaWiki (a simple but widely used OWL-based semantic content management system with light-weight expressiveness) [*OWL1.1 Wiki*] is that users wish to relate schema elements to indicate domain-specific relationships, and generally to organise ontological vocabulary. Examples are statements such as:

- "The property *is_located_in* is in the class *Deprecated_Properties* and was replaced by property *has_location*."
- "Objects of the class *City* should have a value for the property *population*." (expressed by relating class and property)

These are merely pragmatic descriptions, and no logical relationship on schema-level is intended. However, in collaborative vocabulary creation, it is relevant that users can express such intended relationships. An important aspect of Semantic MediaWiki is that users can also query for semantic information, and this is

currently realised as intended by punning. Semantic MediaWiki has already been extended by using off-the-shelf OWL reasoners, and it would be desirable if such systems would be able to deal with the use of punning in such simple cases; (Class/Property ↔ Individual)

Features: **Metamodelling**

Example for: **Simple metamodelling**

- E.g.; a property and an individual: to make a statement asserting that a property is an individual of the class Deprecated_properties

References: [*Wiki*] [*Punning*]

5.15 Use Case #15 - UML Association Class [Designer]

Overview: The Unified Modeling Language (UML) includes a modeling element known as an Association Class which combines the features of a UML Class and a UML Association (UML's construct for defining class to class relationships Association). The Association Class, e.g., the association between classes Person and Company allows a modeler to define a relation as an association and reify it simultaneously. This is convenient when one wants to model attributes of relations themselves. One way to support such case might be Class and ObjectProperty punning (Class ↔ ObjectProperty).

Features: **Metamodelling**

Example for: **Simple metamodelling**

- E.g.; an object property and a class: *PersonCompany* may be used both for an object property and a class.

References: [*UML*] [*Punning*]

5.16 Use Case #16 - Database federation [Designer]

Overview: Some life sciences application designer has been building a database federation scheme. The scheme involves designing an XML schema that describes the fields and values in a variety of databases, and associated query tools that, from a query interface, can write queries (in several variants of SQL) to databases that have relevant information. Those results are presented as a single integrated view. He hears that OWL and Semantic Web technologies might be a suitable technology for implementing the same functionality and making it available using Web standards, but doesn't know where to start. This application illustrates common needs of a wide community of users that would like to use their databases and can easily query them in a convivial way. This motivates a profile where conjunctive query answering is implemented using conventional relational database systems.

Features: **Profiles (OWL 2 QL)**

Example for: **Profiles**

- E.g.; OWL 2 QL profile to easily query a federation of databases in a convivial way

References: [*Who reads?*]

5.17 Use Case #17 - Tools developers [Tools]

Overview: A user adds an assertion to an ontology; however, he accidentally mistypes the URI of an individual. It should be possible to detect this error by comparing the URI of the individual in the axiom with the URIs explicitly declared to be a part of the ontology: if the individual URI has not been explicitly introduced as being in the ontology, the user should be given the opportunity to correct his error.

Features: **Declaration**

Example for: **Declaration**

- E.g.; A person is declared to be a class of an ontology.

References: [*TOOLS*] [*DIG2*] [*OWL API*] [*Syntax Problem*]

5.18 Use Case #18 - Virtual Solar Terrestrial Observatory [Earth and Space]

Overview: A virtual observatory is a suite of software applications on a set of computers that allows users to uniformly find, access, and use resources (data, software, document, and image products and services using these) from a collection of distributed product repositories and service providers. A VO is a service that unites services and / or multiple repositories. from http://lwsde.gsfc.nasa.gov/VO_Framework_7_Jan_05.doc

Numerous single discipline and multi-discipline virtual observatories (e.g., http://vsto.org, http://vmo.nasa.gov/) are beginning to use semantic technologies to provide data access and integration. Some Virtual Observatories are focusing quite heavily on provenance encoding at data ingest time (e.g., http://spcdis.hao.ucar.edu/). The Virtual Solar Terrestrial Observatory (VSTO) is a National Science Foundation and National Center for Atmospheric Research supported effort that allows researchers to find solar and solar-terrestrial data. It provides an ontology-enhanced interface to semantically-enhanced web services that help access a number of online repositories of scientific data. The background OWL ontology contains term descriptions for science terms including instruments, observatories, parameters, etc. Users essentially need to specify a description of the data they wish to retrieve which includes either a specific instrument class or a description of that class, a date range for the data taken, and the parameters. In

order to specify that in relevant science terms, scientists need to be able to represent numerical ranges and comparisons going beyond the numeric support of OWL 1. The application also needs to expand to include spatial descriptions. It would use representational power if provided for spatial/geographic containment.

Requirements: **Qualified Cardinality, Datatype restriction, [Defaults]**

Example for: **Datatype restriction**

- E.g.; the range for atmosphere is above 18000 and below 19600 [feet]

References: [*VSTO*]

5.19 Use Case #19 – Semantic Provenance Capture [Earth and Space]

Overview: In an effort to provide better search capabilities over meta information in addition to scientific data, the SPCDIS effort is providing infrastructure to capture declarative descriptions of scientific provenance information at data ingest time. The initial domain of the effort is solar coronal physics. This effort requires (among other things) extended annotations as well as datatype restriction.

Features: **Datatype restriction, Extended Annotations**

Example for: **Extended annotation** to attach annotations

- E.g.; comments on axioms, such as a SubClass axiom, to express for instance that the the elements of the subclass are data generated by a log parser.

References: [*NCAR*]

6 Appendix: Use Cases Bibliography

[Medical Req]
Web ontology language requirements w.r.t expressiveness of taxononomy and axioms in medicine. Christine Golbreich, Olivier Dameron, Bernard Gibaud, Anita Burgun. In Proc. of ISWC 2003, 2003.
[Micro Theory]
[Creation and Usage of a "Micro Theory" for Long Bone Fractures: An Experience Report]. Howard Goldberg, Vipul Kashyap and Kent Spackman, Proceedings of KR-MED 2008.
[Ontology with Rules]
Ontology enriched by rules for identifying brain anatomical structures. Christine Golbreich, Olivier Dameron, Bernard Gibaud, Anita Burgun. In RIF 2004, Washington, 2004.
and *Annex*.

W3C Editor's Draft

[Brain Imaging]

Towards an Hybrid System Using an Ontology Enriched by Rules for the Semantic Annotation of Brain MRI Images In Proc. of RR 2007
The Brain Anatomy Case Study Christine Golbreich, Olivier Bierlaire, Olivier Dameron, Bernard Gibaud. In Proc. of Protege 2005.

[FMA]

The Foundational Model of Anatomy A
The Foundational Model of Anatomy B
The Foundational Model of Anatomy C.

[Chemistry]

Describing chemical functional groups in OWL-DL for the classification of chemical compounds Natalia Villanueva-Rosales and Michel Dumontier..
Modelling Life Sciences knowledge with OWL1.1

[Auto]

An exploratory study in an automotive company.

[OBO]

The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Barry Smith et al. .

[RO]

-8- Relations in Biomedical Ontologies. .

[OBO2OWL]

OBO to OWL: Go to OWL1.1!
OBO and OWL: Leveraging Semantic Web Technologies for the Life Sciences (ISWC 2007).

[Ordnance]

Experiences of using OWL at the Ordnance Survey.

[SNOMED REQ]

An examination of OWL and the requirements of a large health care terminology.

[Agence Biomedecine]

Changing Kidney Allocation Policy in France: the Value of Simulation.

[Transplant Ontology]

Construction of the dialysis and transplantation ontology.

[Little Web]

A little semantic web goes a long way in biology Wolstencroft, K., Brass, A., Horrocks, I., Lord, P., Sattler, U., Stevens, R., Turi, D.: In: Proceedings of the 2005 International Semantic Web Conference (ISWC 2005), pp. 786-800. Springer, Berlin Heidelberg New York (2005).

[Part Whole]

Simple part-whole relations in OWL Ontologies Alan Rector, Chris Welty. W3C Editor's Draft 11 Aug 2005 .

[TOOLS]

Supporting Early Adoption of OWL 1.1 with Protege-OWL and FaCT++. Matthew Horridge and Dmitry Tsarkov and Timothy Redmond. In OWL: Experiences and Directions (OWLED 06), Athens, Georgia.

[OWL API]

Igniting the OWL 1.1 Touch Paper: The OWL API Matthew Horridge and Sean Bechhofer and Olaf Noppens (2007). In OWL: Experiences and Directions (OWLED 07), Innsbruck, Austria.

[DIG2]

DIG 2.0 Reference Middleware Timo Weith¨oner, Thorsten Liebig, Marko Luther, and Sebastian B¨ohm In OWL: Experiences and Directions (OWLED 07), Innsbruck, Austria.

[Protege OWL]

The Protégé OWL Experience Holger Knublauch, Matthew Horridge, Mark Musen, Alan Rector, Robert Stevens, Nick Drummond, Phil Lord, Natalya F. Noy2, Julian Seidenberg, Hai Wang. In OWL: Experiences and Directions (OWLED 05), Galway, Ireland, 2005.

[N-ary]

N-ary Data predicate use case.

[Web Service]

Preference-based Selection of Highly Configurable Web Services Steffen Lamparter, Anupriya Ankolekar, Stephan Grimm, Rudi Studer: WWW-07, Banff, Canada, 2007.

[Wiki]

Reusing Ontological Background Knowledge in Semantic Wikis Denny Vrandecic, Markus Krötzsch, Proceedings 1st Workshop on Semantic Wikis. Budva, Montenegro, June 2006 .

[UML]

Association.

[Punning]

Punning Use Cases.

[Who reads?]

Who reads our documents?
NIF
NIF Data-Integration slides

[VSTO]

The Virtual Solar-Terrestrial Observatory: A Deployed Semantic Web Application Case Study for Scientific Research McGuinness, D.L., Fox, P., Cinquini, L., West, P., Garcia, J., Benedict, J.L., Middleton, D..
VSTO2.
VMO.

[NCAR]

Semantic Provenance Capture in Data Ingest Systems.