Manchester Syntax for OWL 1.1*

Matthew Horridge[1] and Peter F. Patel-Schneider[2]

[1] University of Manchester
Email: matthew.horridge@cs.man.ac.uk
[2] Bell Labs Research, Alcatel-Lucent
Email: pfps@research.bell-labs.com

Abstract. The Manchester OWL syntax is a user-friendly syntax for OWL DL, fundamentally based on collecting all information about a particular class, property, or individual into a single construct, called a frame. The Manchester OWL syntax has been revised to be a syntax for OWL 1.1, involving adding the new OWL 1.1 description constructs and the new axioms allowed in OWL 1.1 ontologies.

1 Introduction

The original OWL Web Ontology Language as defined by W3C has several syntaxes:

- the normative exchange syntax in RDF/XML [2],
- the so-called abstract syntax for OWL DL [7], and
- an XML syntax [6].

The new version of OWL, OWL 1.1, currently being developed by the W3C OWL working group (http://www.w3.org/2007/OWL/wiki/OWL_Working_Group), is likely to also have several different syntaxes:

- RDF/XML, as before,
- a functional-style syntax, replacing the abstract syntax [1], and
- a revised XML syntax [3].

These syntaxes have differing purposes; for transfer within the Semantic Web, to assist in formal definition of the semantics, for transfer as XML. None of them, however, are designed for ease of use by humans when building or analyzing ontologies. Instead, it was thought that these vital tasks would be performed using tools that provide a graphical interface for human users. Indeed there are tools that provide such a graphical interface, including Protégé (http://protege.stanford.edu/overview/) and Swoop (http://code.google.com/p/swoop/). However, it was soon understood that some parts of creating an ontology require the creation and entry of complex descriptions, for example when creating a class. A useful way of creating such complex descriptions is via

* Alan Rector, Nick Drummond, John Goodwin, Robert Stevens, and Hai W. Wang were heavily involved in the development of the initial Manchester OWL syntax.

a linear syntax. Similarly presenting these complex descriptions can usefully be done by using a linear syntax (generally with nice indentation).

The original Manchester OWL syntax [4] was created to meet this need. It was designed mainly as a syntax for OWL descriptions that would be easier to write and understand, particularly for non-logicians. The Manchester OWL syntax is not, however, a natural language syntax for OWL. The Protégé OWL plugin used the Manchester OWL syntax both when entering and displaying complex descriptions. The syntax could also be used to dump and read OWL DL ontologies.

2 Syntax Philosophy

The Manchester OWL syntax tries to minimize syntactic constructs that are difficult to enter or understand, particularly in description. Complex descriptions use short, intuitive English words instead of logical symbols and the usual precedence rules to minimize parentheses.

For example,

```
Student or Person that hasAge all xsd:integer[ <= 25 ]
```

corresponds to

```
ObjectUnionOf(Student
  ObjectIntersectionOf(Person
    DataAllValuesFrom( hasAge
      Datarange( xsd:integer minInclusive "25"^^xsd:integer ))))
```

The Manchester OWL syntax is much more compact and more readable.

The new functional-style syntax for OWL 1.1 has moved from the frame-like abstract syntax for OWL DL and is, instead, based on axioms. This move was taken because of problems encountered with the original abstract syntax having to do with performing operations on OWL ontologies, such as removing axioms. The functional-style syntax thus corresponds very closely to a potential mechanism for storing ontologies in tools, the structural specification of OWL 1.1 ontologies [1].

The Manchester OWL syntax, purposefully, retains a object-centered frame-like organization for ontologies, as a frame organization supports a very natural way of creating, editing, and viewing ontologies, similar to the presentation of ontologies in most ontology-building tools, e.g., Protégé 4. IN this organization, all the information about an object (a class or property or individual) is collected together, both to aid in perusal of ontology documents and to mirror the way graphical user interfaces like Protégé present information about parts of ontologies.

Frames and the parts of a frame are uniformly introduced by keywords that end in colons, allowing for their easy recognition. It is suggested that when printing ontologies that indentation be used to help with readability, as follows:

```
Class: Person
  SubClassOf: hasAge exactly 1
         and hasGender exactly 1
         and hasGender only {female , male}

Individual: John
  Types: Person
  Facts: hasWife Mary,
         hasSon Bill,
         hasDaughter Susan,
         hasAge  33,
         hasGender male
  SameAs: Jack
```

3 OWL 1.1

The advent of OWL 1.1 required some changes to the Manchester OWL syntax.

The new constructs in descriptions, qualitative cardinality restrictions and self restrictions, were easy to handle, just requiring additional syntax, similar to the existing syntax. Allowing CURIES [5] simply was a matter of extending the syntax of what used to be local names. Punning mostly required no changes, but see below for one problem with punning.

OWL 1.1 allows annotations in many more places than were allowed previously. These annotations are allowed in the Manchester OWL syntax. They are introduced by an `Annotations:` keyword, in a way consistent with the other constructs in the syntax.

The following example shows a class frame with two annotations (a label and a comment) on the frame itself, one annotation on the subclass axiom to `Person` and two annotations on the subclass axiom to `owl:Thing`.

```
Class: Student
  Annotations: rdfs:label "Student"@en
               rdfs:comment "The class of students"
  SubClassOf:
    Annotations: dc:creator "Peter"
    Person,
    Annotations: dc:creator "Matthew"
                 dc:date "12 May 2007"
    owl:Thing
```

Because annotations break up the flow of information, they can be harmful to readability.

The syntax here allows for annotations on annotations which are not (yet) part of OWL 1.1. Extending annotations in this way is one of the issues that the OWL working group is considering.

Some constructs in OWL 1.1, such as the axioms for n-way disjointness of classes and properties and n-way difference and sameness of individuals, are not obviously part of any particular object. In fact, they are equally about each of the n classes, properties, or individuals involved. Attaching these constructss to any one frame would be misleading. The Manchester OWL syntax thus breaks from its general object-centricity with specific syntax for these axioms, as in the following example.

```
DifferentIndividuals: f:Jeff f:Emily f:Jack f:Ellen f:Susan
```

4 Issues

OWL 1.1 does pose a few problems for the Manchester OWL syntax.

The frames for classes, properties, and individuals play a similar role to declarations in OWL 1.1, and thus the syntax does not have a separate construct for declarations. However, it can be the case that an ontology adds information to an imported class, property, or individual. In this case the frame for that class, property, or individual is not a declaration. Global analysis is thus required to determine which declarations are required in an ontology.

Moving between the Manchester OWL syntax and the functional-style syntax and structural specification for OWL 1.1 requires moving between frames and axioms. This is actually not difficult at all, as each frame construct has a number of top-level pieces that each correspond directly to an axiom.

For example, the frame

```
Individual: John
  Types: Person
  Facts: hasWife Mary,
 hasSon Bill,
 hasDaughter Susan,
 hasAge  33,
 hasGender male
  SameAs: Jack
```

corresponds to seven axioms (and possibly one declaration) in the functional-style syntax, as follows:

```
Declaration(Individual(John))
ClassAssertion(Person John)
ObjectPropertyAssertion(hasWife John Mary)
ObjectPropertyAssertion(hasSon John Bill)
ObjectPropertyAssertion(hasDaughter John Susan)
DataPropertyAssertion(hasAge John "33"^^xsd:integer)
ObjectPropertyAssertion(hasGender John male)
SameIndividuals(John Jack)
```

Punning between object and data properties (i.e., an object property and a data property with the same name) could cause ambiguity between object and data property restrictions. Overcoming this ambiguity would require much extra vocabulary, resulting in lessened readability. Therefore the Manchester OWL syntax does not support dumping of ontologies with punned object and data properties. Duplicating the syntax, as in the OWL 1.1 functional-style syntax, was another possible solution, but it would have interfered with readability too much.

5 The Manchester OWL Syntax

Here is a grammar for the Manchester OWL Syntax, with some low-level productions (e.g., **digits**) removed. In the grammar, terminals are enclosed in single quotes (‘’) and non-terminals are in **bold**. Optional parts are in brackets ([]) and repeated parts are in braces (). Some productions are given informally or by references, indicated with *italics*.

5.1 CURIES, Names, and Constants

NCName ::= *as defined in XML Namespaces*
prefix ::= **NCName**
irelative-ref ::= *as defined in RFC-3987*
reference ::= **irelative-ref**
curie ::= [[**prefix**] ‘`:`’] **reference**
Abbreviated-IRI ::= **curie**
Full-IRI ::= ‘`<`’ *IRI as defined in RFC-3987* ‘`>`’
URI ::= **Full-IRI** | **Abbreviated-IRI**

datatypeURI ::= **URI** | ‘`integer`’ | ‘`decimal`’ | ‘`float`’ | ‘`string`’
owlClassURI ::= **URI**
objectPropertyURI ::= **URI**
dataPropertyURI ::= **URI**
individualURI ::= **URI**
annotationURI ::= **URI**
ontologyURI ::= **URI**

stringLiteral ::= ‘`"`’ *string with* `\` *and* `"` `\`*-escaped* ‘`"`’
integerLiteral ::= [‘`+`’|‘`-`’] **digits**
decimalLiteral ::= [‘`+`’|‘`-`’] **digits** ‘`.`’ **digits**
exponent ::= (‘`e`’ | ‘`E`’) [‘`+`’ | ‘`-`’] **digits**
floatingPointLiteral ::= [‘`+`’|‘`-`’] (**digits** [‘`.`‘**digits**] [**exponent**]
| ‘`.`’ **digits**[**exponent**]) (‘`f`’ | ‘`F`’)
typedLiteral ::= **stringLiteral** ‘`^^`’ **datatype**

typedConstant ::= **stringLiteral** | **integerLiteral** | **decimalLiteral** | **floatingPointLiteral** | **typedLiteral**
languageTag ::= *a language tag as specified in RFC-4646*
untypedConstant ::= **stringLiteral** ['@' **languageTag**]
constant ::= **typedConstant** | **untypedConstant**

5.2 Lists, Entities, and Annotations

Because comma-separated lists occur in very many places in the syntax, to save space the grammar here has two meta-productions, one for lists and one for lists with annotations in them.

*NT***List** ::= *NT* { ',' *NT* }
*NT***AnnotatedList** ::= [**annotations**] *NT* { ',' [**annotations**] *NT* }

The non-terminal `descriptionList` is thus a comma-separated list of descriptions, such as

```
Person, Student, hasChild exactly 1 Person
```

and the non-terminal `descriptionAnnotationList` is thus a comma-separated list of descriptions each possibly proceeed by annotations, such as

```
Annotations: dc:creator "Peter"
Person,
Annotations: dc:creator "Matthew"
             dc:date "12 May 2007"
owl:Thing
```

Annotations and entities are formed as follows:

annotation ::= **annotationURI** (**constant** | **individualURI** | **entity**)
annotations ::= 'Annotations:' **annotationAnnotatedList**
entity ::= 'Datatype' '(' **datatypeURI** ')'
| 'OWLClass' '(' **OWLClassURI** ')'
| 'ObjectProperty' '(' **objectPropertyURI** ')'
| 'DataProperty' '(' **dataPropertyURI** ')'
| 'Individual' '(' **individualURI** ')'
| 'AnnotationProperty' '(' **annotationPropertyURI** ')'

This grammar allows annotations on annotations, which are not (yet) in OWL 1.1. The current state of annotations in OWL 1.1 corresponds to replacing the production for annotations with

annotations ::= 'Annotations:' **annotationList**

5.3 Ontology Documents

namespace ::= 'Namespace:' [**prefix**] '=' **Full-IRI**
ontologyDoc ::= { **namespace** } **ontology**
ontology ::= 'Ontology:' **ontologyURI** { **import** } { **annotations** } { **frame** }
import ::= 'Import:' **URI**
frame ::= **classFrame** | **objectPropertyFrame** | **dataPropertyFrame** | **individualFrame** | **misc**

5.4 Property Expressions and Datatypes

inverseObjectProperty ::= 'inverseOf' **objectPropertyExpression**
objectPropertyExpression ::= **objectPropertyURI**
| **inverseObjectProperty**
dataPropertyExpression ::= **dataPropertyURI**

dataComplementOf ::= 'not' **dataRange**
dataOneOf ::= '{' **constant** { ',' **constant** } '}'
datatypeFacet ::= 'length' | 'minLength' | 'maxLength'
| 'pattern' | '<=' | '<' | '>=' | '>'
| 'digits' | 'fraction'
restrictionValue ::= **constant**
datatypeRestriction ::= **datatypeURI**
'[' **datatypeFacet restrictionValue**
{ , **datatypeFacet restrictionValue** } ']'
dataRange ::= **datatypeURI** | **dataComplementOf**
| **dataOneOf** | **datatypeRestriction**

Note: If the datatypeURI involved is an XML Schema datatype then the datatypeFacets and restrictionValues have to be valid for that datatype.

5.5 Descriptions

atomic ::= **OWLClassURI**
| '{' **individualURI** { ',' **individualURI** } '}'
| '(' **description** ')'
restriction ::= **objectPropertyExpression** 'some' **primary**
| **objectPropertyExpression** 'only' **primary**
| **objectPropertyExpression** 'value' **individualURI**
| **objectPropertyExpression** 'min' **nonNegativeInteger** [**primary**]
| **objectPropertyExpression** 'exactly' **nonNegativeInteger** [**primary**]
| **objectPropertyExpression** 'max' **nonNegativeInteger** [**primary**]
| **objectPropertyExpression** 'Self'
| **dataPropertyExpression** 'some' **dataRange**
| **dataPropertyExpression** 'only' **dataRange**
| **dataPropertyExpression** 'value' **constant**
| **dataPropertyExpression** 'min' **nonNegativeInteger** [**dataRange**]
| **dataPropertyExpression** 'exactly' **nonNegativeInteger** [**dataRange**]
| **dataPropertyExpression** 'max' **nonNegativeInteger** [**dataRange**]
primary ::= ['not'] (**restriction** | **atomic**)
conjunction ::= | **primary** 'and' **primary** { 'and' **primary** }
OWLClassURI 'that' ['not'] **restriction** { 'and' ['not'] **restriction** }
| **primary**
description ::= **conjunction** 'or' **conjunction** { 'or' **conjunction** }
| **conjunction**

5.6 Class, Property, and Individual Frames

classFrame ::= 'Class:' **OWLClassURI**
{ 'Annotations:' **annotationAnnotatedList**
| 'SubClassOf:' **descriptionAnnotatedList**
| 'EquivalentTo:' **descriptionAnnotatedList**
| 'DisjointWith:' **descriptionAnnotatedList**
| 'DisjointUnionOf:' **annotations descriptionList** }

objectPropertyFrame ::= 'ObjectProperty:' **objectPropertyURI**
{ 'Annotations:' **annotationAnnotatedList**
| 'Domain:' **descriptionAnnotatedList**
| 'Range:' **descriptionAnnotatedList**
| 'Characteristics:' **objectPropertyCharacterAnnotatedList**
| 'SubPropertyOf:' **objectPropertyExpressionAnnotatedList**
| 'EquivalentTo:' **objectPropertyExpressionAnnotatedList**
| 'DisjointWith:' **objectPropertyExpressionAnnotatedList**
| 'Inverses:' **objectPropertyExpressionAnnotatedList**
| 'SubPropertyChain:' **annotations objectPropertyExpression**
'o' **objectPropertyExpression** {'o' **objectPropertyExpression**} }

objectPropertyCharacter ::= 'Functional' | 'InverseFunctional'
| 'Reflexive' | 'Irreflexive'
| 'Symmetric' | 'Asymmetric' | 'Transitive'

dataPropertyFrame ::= 'DataProperty:' **dataPropertyURI**
{ 'Annotations:' **annotationAnnotatedList**
| 'Domain:' **descriptionAnnotatedList**
| 'Range:' **dataRangeAnnotatedList**
| 'Characteristics:' **annotations** 'Functional'
| 'SubPropertyOf:' **dataPropertyExpressionAnnotatedList**
| 'EquivalentTo:' **dataPropertyExpressionAnnotatedList**
| 'DisjointWith:' **dataPropertyExpressionAnnotatedList** }

objectPropertyFact ::= **objectPropertyURI individualURI**
dataPropertyFact ::= **dataPropertyURI constant**
fact ::= ['not'] (**objectPropertyFact** | **dataPropertyFact**)

individualFrame ::= 'Individual:' **individualURI**
{ 'Annotations:' **annotationAnnotatedList**
| 'Types:' **descriptionAnnotatedList**
| 'Facts:' **factAnnotatedList**
| 'SameAs:' **individualURIAnnotatedList**
| 'DifferentFrom:' **individualURIAnnotatedList** }

5.7 Non-Frame Axioms

misc ::= 'EquivalentClasses:' **annotations descriptionList**
| 'DisjointClasses:' **annotations descriptionList**
| 'EquivalentObjectProperties:' **annotations objectPropertyList**
| 'DisjointObjectProperties:' **annotations objectPropertyList**
| 'EquivalentDataProperties:' **annotations dataPropertyList**
| 'DisjointDataProperties:' **annotations dataPropertyList**
| 'SameIndividual:' **annotations individualURIList**
| 'DifferentIndividuals:' **annotations individualURIList**

6 Example

Here is a portion of an ontology (from the OWL Primer).

```
Class: Person
  Annotations: rdfs:label "Person"@en
  SubClassOf: hasAge exactly 1
              and hasGender exactly 1
              and hasGender only {female , male}

Class: Man SubClassOf: Person
  EquivalentTo: Person that hasGender value male

Class: Parent SubClassOf: Person
  EquivalentTo: Person that hasChild min 1 Person

Class: Teenager
  EquivalentTo: Person that hasAge some integer[>= 13 , < 20]

Class: Narcissist EquivalentTo: Person that loves Self

ObjectProperty: hasWife
  Characteristics: Functional, InverseFunctional,
                   Irreflexive, Asymmetric
  Domain: Person, Man
  Range: Person, Woman
  SubPropertyOf: hasSpouse, loves

Individual: Jeff
  Annotations: rdfs:comment "Jeff is a narcissist"
  Types: hasChild exactly 2
  Facts: hasWife Emily,
         hasChild Ellen,
         hasAge  77,
         loves Jeff
```

```
Individual: Jack
  Facts: not hasAge "53"^^integer
```

7 Status and Future Work

The Manchester OWL syntax is used in the Protégé 4 ontology development tool (http://www.co-ode.org/downloads/protege-x/). Protégé 4 can dump and load OWL 1.1 ontologies in the Manchester OWL syntax. As well, complex descriptions can be entered using the Manchester OWL syntax. Most of the examples in this paper were generated using Protégé 4. Although the Manchester OWL syntax was initially developed to be more human-readable than other OWL syntaxes, there has as yet been no formal user studies to support this aspect of the current language.

The OWL Primer (http://www.w3.org/2007/OWL/wiki/Primer), under consideration in the W3C OWL Working Group, currently uses the Manchester OWL syntax as its primary syntax because this syntax is the most readable OWL syntax. The OWL Working Group may also decide to produce a Working Group note on the Manchester OWL syntax.

The OWL 1.1 language is still under development. The syntax in this paper attempts to track the current version of OWL 1.1 with a bit of anticipation related to annotations. The Manchester OWL syntax will be updated as OWL 1.1 is updated. Protégé 4 currently implements a slightly older version of the syntax (and has a couple of bugs in this area as well).

References

1. OWL 1.1 web ontology language: Structural specification and functional-style syntax. W3C Working Draft, http://www.w3.org/TR/owl11-syntax/, 2008.
2. RDF/XML syntax specification (revised). W3C Recommendation, http://www.w3.org/TR/rdf-syntax-grammar/, 2004.
3. Bernardto Cuenca Grau, Boris Motik, and Peter F. Patel-Schneider. OWL 1.1 web ontology language: XML syntax. W3C Member Submission, http://www.w3.org/Submission/2006/SUBM-owl11-xml_syntax-20061219/, 2006.
4. Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai H. Wang. The Manchester OWL syntax. In *OWL Experiences and Directions Workshop*, 2006.
5. CURIE syntax 1.0: A syntax for expressing compact URIs. W3C Working Draft, http://www.w3.org/TR/2007/WD-curie-20071126/, 2007.
6. OWL web ontology language: XML presentation syntax. W3C Note, http://www.w3.org/TR/owl-xmlsyntax/, 2003.
7. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL web ontology language: Semantics and abstract syntax. W3C Recommendation, http://www.w3.org/TR/owl-semantics/, 2004.