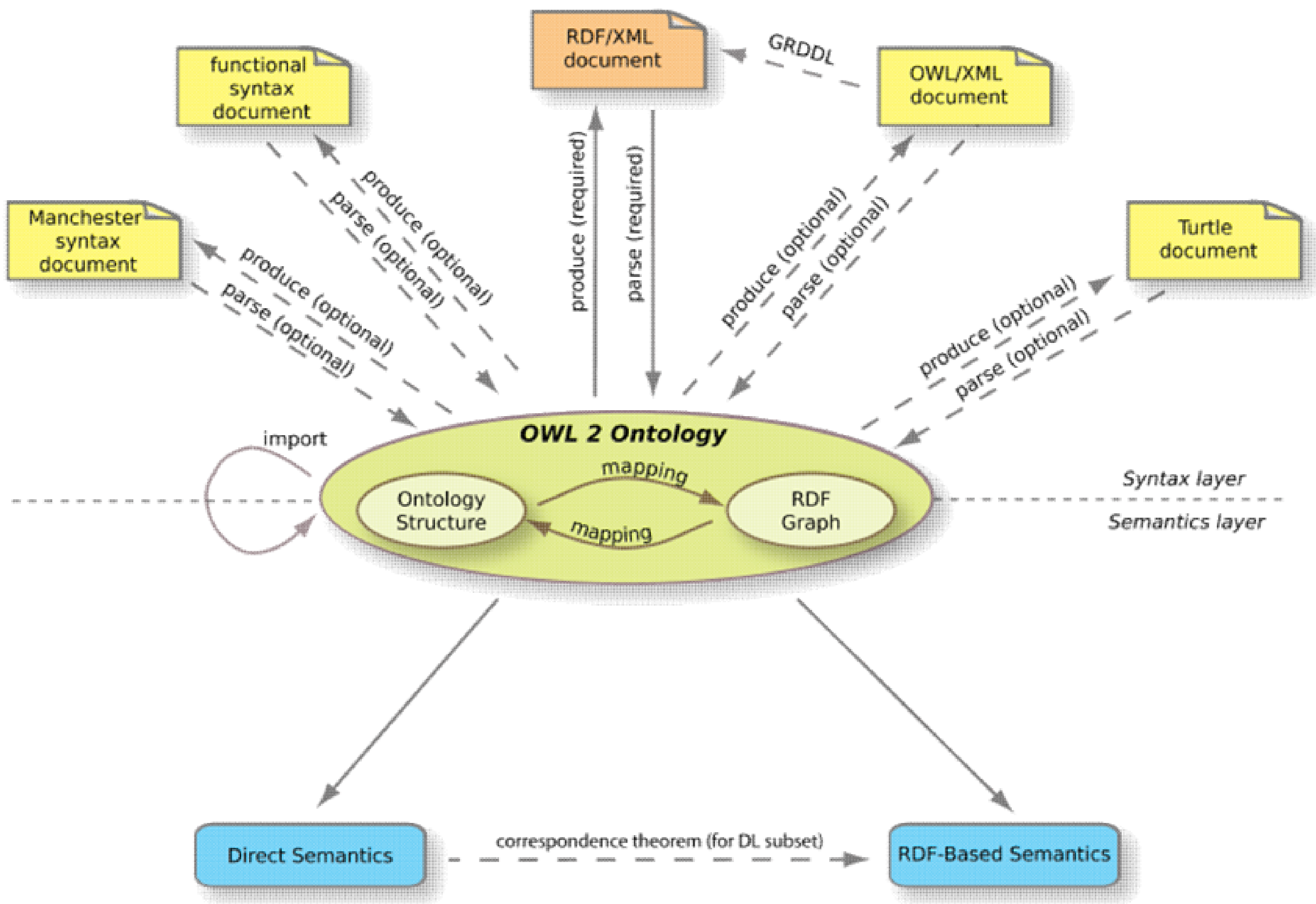
ISIT315
WEEK 5

OWL 2

Ontology

- Formalized vocabularies of terms, often covering a specific domain and shared by a community of users.
  - They specify the definitions of terms by describing their relationships with other terms in the ontology.
- An ontology is a set of <u>precise</u> descriptive statements about some part of the world

Structure of OWL 2



Different Syntaxes

Name of Syntax	Specification	Status	Purpose
RDF/XML	Mapping to RDF Graphs, RDF/XML	Mandatory	Interchange (can be written and read by all conformant OWL 2 software)
OWL/XML	XML Serialization	Optional	Easier to process using XML tools
Functional Syntax	Structural Specification	Optional	Easier to see the formal structure of ontologies
Manchester Syntax	Manchester Syntax	Optional	Easier to read/write DL Ontologies
Turtle	Mapping to RDF Graphs, Turtle	Optional, Not from OWL-WG	Easier to read/write RDF triples

OWL Syntax

- An OWL ontology is an RDF graph → a set of RDF triples
  - The meaning of an OWL ontology is solely determined by RDF graph
- OWL is a vocabulary extension of RDF
- The built-in vocabulary for OWL comes from OWL namespace

owl: http://www.w3.org/2002/07/owl#

OWL 2: Modeling knowledge

- OWL 2 is a knowledge representation knowledge, designed to formulate, exchange and reason with knowledge about a domain of interest.
- Basic notions:
  - **Axioms:** the basic statements that an OWL ontology expresses
  - **Entities:** elements used to refer to real-world objects
  - **Expressions:** combinations of entities to form complex descriptions from basic ones

To formulate knowledge explicitly

- Ontology consists of <u>statements</u> (or propositions)
  – Example of statements
    - It is raining
    - Every man is mortal
- These statements are called axioms

OWL 2 Ontology

- is a collection of axioms
  - ontology asserts that its axioms are true

In OWL 2

- Objects as *individuals*
- Categories as *classes*
- Relations as *properties*
- *Note: A class is a name and collection of properties that describe a set of individuals*

OWL statements

- Made up of atomic statements
  - Mary is female
  - John and Mary are married
- Objects: Mary, John
- Categories: female
- Relations: married
- All atomic constituents are called **entities**

Properties in OWL 2

- *Object properties* relate objects to objects
  - A person to their spouse
- *Datatype properties* assign data values to objects
  - An age to a person.
- *Annotation properties* are used to encode information about the ontology itself
  - Author and creation date

Constructors

- Names of entities can be combined into expressions using **constructors**
  - atomic classes:
    - female, professor
  - combined conjunctively to form class expressions
    - female professors
- This way, expressions = new entities

Functional-Style syntax

- is designed to be easier for specification purposes and to provide a foundation for the implementation of OWL 2 tools such as APIs and reasoners.

ClassAssertion

- Functional-style syntax

  `ClassAssertion( :Person :Mary )`

- RDF/XML syntax

  `<Person rdf:about="Mary"/>`

- `Mary` belongs to the class of all `Persons`
- Note: one individual can belong to many classes simultaneously

  `ClassAssertion( :Woman :Mary )`

Class Hierarchies

- Functional-style syntax

```
SubClassOf( :Woman :Person)
```

- RDF/XML syntax

```
<owl:Class rdf:about="Woman">
  <rdfs:subClassOf
    rdf:resource="Person"/>
</owl:Class>
```

- able to specify generalization relationships of all classes

Equivalent Class

- Functional-style syntax

```
EquivalentClasses( :Person :Human)
```

- RDF/XML syntax

```
<owl:Class rdf:about="Person">
  <owl:equivalentClass rdf:resource="Human"/>
</owl:Class>
```

Disjoint Class

- Functional-style syntax

```
DisjointClasses( :Woman :Man)
```

- RDF/XML syntax

```
<owl:AllDisjointClasses>
    <owl:members rdf:parseType="Collection">
        <owl:Class rdf:about="Woman"/>
        <owl:Class rdf:about="Man"/>
    </owl:members>
</owl:AllDisjointClasses>
```

Inferencing Example

- The disjointness axiom can be used to deduce
  - Mary is not a Man
  - Mother and Man are disjoint

Object properties

- Functional-style syntax

```
ObjectPropertyAssertion(:hasWife :John :Mary)
```

- RDF/XML syntax

```
<rdf:Description rdf:about="John">
  <hasWife rdf:resource="Mary"/>
</rdf:Description>
```

Negative Property

- Functional-style syntax

```
NegativeObjectPropertyAssertion(:hasWife :Bill :Mary)
```

- RDF/XML syntax

```
<owl:NegativePropertyAssertion>
  <owl:sourceIndividual rdf:resource="Bill"/>
  <owl:assertionProperty rdf:resource="hasWife"/>
  <owl:targetIndividual rdf:resource="Mary"/>
</owl:NegativePropertyAssertion>
```

Property Hierarchies

- Functional-style syntax

```
SubObjectPropertyOf(:hasWife :hasSpouce)
```

- RDF/XML syntax

```
<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:subPropertyOf rdf:resource="hasSpouse"/>
</owl:ObjectProperty>
```

Domain and Range restrictions

- Functional-style syntax

```
ObjectPropertyDomain(:hasWife :Man)
ObjectPropertyRange(:hasWife :Woman)
```

- RDF/XML syntax

```
<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:domain rdf:resource="Man"/>
  <rdfs:range rdf:resource="Woman"/>
</owl:ObjectProperty>
```

Equality and Inequality of individuals

- Functional-style syntax

```
DifferentIndividuals(:John :Bill)
SameIndividuals(:James :Jim)
```

- RDF/XML syntax

```
<rdf:Description rdf:about="John">
  <owl:differentFrom rdf:resource="Bill"/>
</rdf:Description>

<rdf:Description rdf:about="James">
  <owl:sameAs rdf:resource="Jim"/>
</rdf:Description>
```

Datatypes

- Relates individuals to data values
- Use XML schema datatypes
- Functional-style syntax

```
DataPropertyAssertion(:hasAge :John “51”^^xsd:integer)
```

- RDF/XML syntax

```
<Person rdf:about="John">
   <hasAge
   rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">51<
   /hasAge>
</Person>
```

NegativeDataPropertyAssertion

- Functional-style syntax

```
NegativeDataPropertyAssertion( :hasAge :Jack
  "53"^^xsd:integer )
```

- RDF/XML syntax

```
<owl:NegativePropertyAssertion>
<owl:sourceIndividual rdf:resource="Jack"/>
<owl:assertionProperty rdf:resource="hasAge"/>
<owl:targetValue
  rdf:datatype="http://www.w3.org/2001/XMLSchema#in
  teger"> 53
</owl:targetValue>
</owl:NegativePropertyAssertion>
```

Complex classes

- ```
  EquivalentClasses(
   :Mother
   ObjectIntersectionOf(:Woman:Parent)
   )
  ```

- ```
  EquivalentClasses(
  :Parent
  ObjectUnionOf( :Mother :Father )
  )
  ```

- ```
  EquivalentClasses(
  :ChildlessPerson
   ObjectIntersectionOf(
     :Person
      ObjectComplementOf( :Parent )
     )
   )
  ```
  ```

Property restrictions

- Use constructors involving properties
- *Existential quantification*
  - Defines a class as the set of all individuals that are connected via a particular property to another individual which is an instance of a certain class.
    - Natural language indicators for the usage of existential quantification are words like “some,” or “one.”
- *Universal quantification*
  - Describe a class of individuals for which all related individuals must be instances of a given class.
    - Natural language indicators for the usage of universal quantification are words like “only,” “exclusively,” or “nothing but.”

Existential quantification

- For every instance of Parent, there exists at least one child, and that child is a member of the class Person.

```
EquivalentClasses(
  :Parent
  ObjectSomeValuesFrom( :hasChild :Person )
)
```

Universal quantification

Somebody is a happy person exactly if all their children are happy persons

```
EquivalentClasses(
  :HappyPerson
  ObjectAllValuesFrom( :hasChild
  :HappyPerson )
 )
```

Property Cardinality Restrictions

- To specify the number of individuals involved in the restriction

Property cardinality restrictions

```
ClassAssertion(
  ObjectMaxCardinality( 4 :hasChild :Parent)
  :John
)
```

```
ClassAssertion(
  ObjectMinCardinality( 2 :hasChild :Parent)
  :John
)
```

```
ClassAssertion(
  ObjectExactCardinality( 2 :hasChild :Parent)
  :John
)
```

Enumeration of Individuals

```
EquivalentClasses(
   :MyBirthdayGuests
     ObjectOneOf( :Bill :John :Mary)
  )
```

- Classes defined this way are sometimes referred to as *closed classes* or enumerated sets
  - Bill, John, and Mary are the *only* members of MyBirthdayGuests

Advanced property characteristics

```
InverseObjectProperties( :hasParent :hasChild )
SymmetricObjectProperty( :hasSpouse )
AsymmetricObjectProperty( :hasChild )
DisjointObjectProperties(:hasParent :hasSpouse )
ReflexiveObjectProperty( :hasRelative )
IrreflexiveObjectProperty( :parentOf )
FunctionalObjectProperty( :hasHusband )
InverseFunctionalObjectProperty( :hasHusband )
TransitiveObjectProperty( :hasAncestor )
```

Property chains

```
SubObjectPropertyOf(
  ObjectPropertyChain( :hasParent :hasParent
  )
  :hasGrandparent
)
```

- Enable `hasGrandparent` property to be defined more specific
- `hasGrandparent` connects all individuals that are linked by a chain of exactly two `hasParent` properties

Keys

- Each named instance of the class expressions is uniquely identified by a set of values

```
HasKey( :Person () ( :hasSSN ) )
```

Advanced Use of Datatypes

```
DatatypeDefinition(
  :personAge
   DatatypeRestriction( xsd:integer
 xsd:minInclusive "0"^^xsd:integer
 xsd:maxInclusive "150"^^xsd:integer
   )
 )
```

Another example

```
DatatypeDefinition(
:toddlerAge DataOneOf(
  "1"^^xsd:integer
  "2"^^xsd:integer )
 )
```

Annotations

- Functional-style syntax

```
AnnotationAssertion( rdfs:comment :Person
  "Represents the set of all people." )
```

- RDF/XML syntax

```
<owl:Class rdf:about="Person">
   <rdfs:comment>Represents the set of all
     people.</rdfs:comment>
   </owl:Class>
```

References

- https://www.w3.org/TR/owl2-primer/
- Refer to the above document for other syntaxes

The buttons below can be used to show or hide the available syntaxes.

Hide Functional-Style Syntax | Show RDF/XML Syntax | Show Turtle Syntax | Show Manchester Syntax | Show OWL/XML Syntax

How is ontology different from XML or XML Schema

- OWL Ontology → knowledge representation
- XML/XMLSchema → message format
- Most industry based web standards consist of a combination of message formats and protocol specifications → operational semantics
- OWL 2 does not provide means to prescribe how a document should be structured syntactically

Consider the following example

- Upon receipt of this `PurchaseOrder` message, transfer `Amount` dollars from `AccountFrom` to `AccountTo` and ship `Product`
- This specification is not designed to support reasoning outside the transaction context, e.g. `Product` is a type of Chardonnay therefore it must be a white wine.

Advantage of OWL ontologies

- Availability of reasoning tools that provide <u>generic</u> support that is not specific to the particular subject domain
- Note: building a sound and useful reasoning system is not a simple effort.
-

Considerations

- Must consider which species of OWL (OWL Lite, OWL DL or OWL Full) meet their needs
- OWL Lite vs. OWL DL
  - Depends on the extent to which users require the more expressive restriction constructs provided by OWL DL
- OWL DL vs. OWL Full
  - Depends on the extent to which users require meta-modelling facilities of RDF Schema (i.e. defining classes of classes).
    - Reasoning support for OWL Full is less predicatable

OWL 2 vs. Database

- Closed-world assumption
  - If some fact is not present in the database, it is usually considered to be FALSE
- Open-world assumption
  - If some fact is not present in ontology (OWL 2 document) it may simply be missing (but possibly true)