2. OWL — Web Ontology Language

- OWL: Web Ontology Language
- Standard for representing ontologies on the web

OWL is the **semantic modeling language** used to represent knowledge in a machine-readable and logically consistent way.

- W (OL): Ontology Language for the Web environment

Why Conusing Name? OWL, not WOL

For example, we have:

- Web browsers: browsers for the Web
- Web servers: servers for the Web
- Web protocols: protocols for the Web

The same pattern applies:

- Web Ontology Language: Ontology language for the Web

However, when we name it WOL, the meaning becomes unclear.

1. Is it "Web Ontology Language" or "World Ontology Language"?
2. Or is it "Web Object Language"?

But, OWL sounds like "owl" (the bird), which is easy to remember and symbolizes wisdom.

Semantic Web Stack: RDF, RDFS, and OWL

Acronym	Full Name	Purpose
RDF	Resource Description Framework	Basic data model for representing triples (subject–predicate–object)
RDFS	RDF Schema	Adds basic vocabulary for class and property hierarchies
OWL	Web Ontology Language	Adds rich semantics, logic, and reasoning capabilities
SPARQL	SPARQL Protocol And RDF Query Language	Query language for RDF and Knowledge Graphs

These three form a semantic web stack:

OWL → Logic & Reasoning

↑

RDFS → Schema & Structure

↑

RDF → Data Format

Easy summary:

- **RDF = Data**
- **RDFS = Schema**
- **OWL = Logic + Inference**

RDF (Resource Description Framework)

RDF stores facts as triples: Subject — Predicate — Object

Example triple sentence:

> "Tom eats Apple"
> "Tom is Person"

This triple can be represented as Turtle syntax or XML syntax.

Turtle

- Turtle is a compact, human-readable syntax for RDF.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex:  <http://example.org/> .

ex:Tom ex:eats ex:Apple .
ex:Tom rdf:type ex:Person .
```

- ex: is used for concepts that we define (example namespace)
- rdf: is used for standard RDF vocabulary such as rdf:type
- rdfs: is used for standard RDFS vocabulary such as rdfs:Class

XML

- XML is verbose and machine-readable

The triple "Tom eats Apple" in RDF/XML:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/">

  <rdf:Description rdf:about="http://example.org/Tom">
    <ex:eats rdf:resource="http://example.org/Apple"/>
  </rdf:Description>

</rdf:RDF>
```

RDF Triple Structure Mapping

The subject/predicate/object should be expressed in XML, and the standard elements are all in the `rdf:` namespace.

- `rdf:Description` + `rdf:about="...Tom"` → Triple subject
- `<ex:eats ...>` → Triple predicate
- `rdf:resource="...Apple"` → Triple object

In plain English:

- I'm describing Tom (subject)
    - What does Tom do? (predicate)
        - Tom eats Apple (object)

Other examples (simplified without namespaces):

```
<rdf:Description rdf:about="Tom">
    <ex:eats rdf:resource="Apple"/>
    <ex:age>25</ex:age>
    <ex:livesIn rdf:resource="Seoul"/>
    <rdf:type rdf:resource="Person"/>
</rdf:Description>
```

Turtle format examples:

```
ex:Tom ex:eats ex:Apple .
ex:Tom ex:age 25 .
ex:Tom ex:livesIn ex:Seoul .
ex:Tom rdf:type ex:Person .
```

RDF Characteristics

RDF only stores facts; It does not understand meaning.

- Stores data ✅
- Defines classes ❌
- Inheritance ❌
- Logic reasoning ❌

RDFS (RDF Schema)

RDFS adds structure and vocabulary.

It introduces:

- Classes
- Subclasses
- Property domain & range

RDFS Example — Class Definition

Define a class:

```
ex:Person rdf:type rdfs:Class .
```

Subclass relation:

```
ex:Student rdfs:subClassOf ex:Person .
```

RDFS Example — Property Definition

Define property meaning:

```
ex:eats rdfs:domain ex:Person .
ex:eats rdfs:range ex:Food .
```

Meaning:

- Subject of eats is a Person
- Object of eats is Food

Domain and Range

For example, $f(x) = x + 1$:

```
Domain: {1, 2, 3}  → Possible input values
Range:  {2, 3, 4}  → Possible output values

f(1) = 2
f(2) = 3
f(3) = 4
```

In RDF, Property is a Relationship between Subject and Object (Function Analogy)

```
ex:eats rdfs:domain ex:Person .
ex:eats rdfs:range ex:Food .
```

Property as a function:

```
eats: Person → Food
      ^^^^^^    ^^^^
      Domain    Range
      (input)   (output)
```

RDFS Automatic Type Inference Example

```
ex:drives rdfs:domain ex:Person .
ex:drives rdfs:range ex:Vehicle .

ex:Alice ex:drives ex:Tesla .
```

RDFS reasoner infers automatically:

```
ex:Alice rdf:type ex:Person (domain)
ex:Tesla rdf:type ex:Vehicle (range)
```

Another RDFS Inference Example

Data:

```
ex:Tom rdf:type ex:Student .
```

Schema:

```
ex:Student rdfs:subClassOf ex:Person .
```

Reasoner automatically infers:

```
Tom is a Person
```

RDFS Characteristics

RDFS provides basic structure.

- Feature RDFS
- Classes ✅
- Inheritance ✅
- Domain / Range ✅
- Logical constraints ❌
- Advanced reasoning ❌

What OWL Is Used For

OWL allows you to define:

- Concepts (Classes)
- Relationships (Properties)
- Individuals (Instances)
- Constraints (Rules)
- Logical axioms (Inference rules)

Goal:

> Represent domain knowledge so machines can reason about it.

OWL Example (Class Hierarchy)

```
<owl:Class rdf:about="http://example.org#Mammal"/>

<owl:Class rdf:about="http://example.org#Person">
    <rdfs:subClassOf rdf:resource="http://example.org#Mammal"/>
</owl:Class>

<owl:Class rdf:about="http://example.org#Student">
    <rdfs:subClassOf rdf:resource="http://example.org#Person"/>
</owl:Class>
```

Meaning:

Student ⊆ Person ⊆ Mammal

Why OWL, not RDF or RDFS?

- RDF: A general graph data model with an **object–property–value** structure.
- RDFS: Lightweight meta-information that says “these things form a type system/schema.”
- OWL: An ontology language that uses logic to support strong semantics and reasoning, such as “Student $\subseteq$ Person $\subseteq$ Mammal.”

OWL Components

Classes (Concepts)

Represent categories:

Examples:

- Person
- Animal
- Food
- Company

Used to classify entities.

Individuals (Instances)

Represent real objects:

Examples:

- Alice (Person)
- Pizza (Food)
- Google (Company)

OWL connects instances to classes:

```
Alice rdf:type Person
```

Object Properties (Relationships)

Connect two entities:

Example:

```
eats(Person → Food)
```

OWL definition:

```
<owl:ObjectProperty rdf:about="#eats">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Food"/>
</owl:ObjectProperty>
```

Used for semantic relationships between objects.

Data Properties (Attributes)

Connect entity to literal values:

Example:

```
Alice hasAge 20
```

OWL definition:

```
<owl:DatatypeProperty rdf:about="hasAge">
  <rdfs:domain rdf:resource="Person"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>

<ex:Person rdf:about="Alice">
  <ex:hasAge rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">20</ex:hasAge>
</ex:Person>
```

Used for numbers, strings, dates.

Object vs Data Properties

Property Type	Connects To	Example Triple	Meaning
ObjectProperty	Another resource (URI)	`Alice knows Bob`	Relationship between two individuals
DatatypeProperty	Literal value (string, number, date, etc.)	`Alice hasAge 20`	Attribute value of an individual

Automatic Type Inference Example

Given data:

```
Bob eats Pizza
Pizza rdf:type Food
```

OWL rule:

```
eats domain = Person
```

System infers:

```
Bob rdf:type Person
```

This is logical reasoning, not hardcoded logic.

OWL Logical Constraints Example

Example rule:

```
Person must have at least 1 name
```

OWL concept:

```
Person ⊑ (hasName min 1)
```

OWL can detect:

- Missing data
- Invalid structures
- Logical contradictions

```
# 1. Exactly 1
ex:Person rdfs:subClassOf [
owl:onProperty ex:hasName ;
owl:cardinality 1
] .

# 2. Min 1
ex:Student rdfs:subClassOf [
owl:onProperty ex:takes ;
owl:minCardinality 1
] .

# 3. Max 1

ex:Student rdfs:subClassOf [
owl:onProperty ex:hasAdvisor ;
owl:maxCardinality 1
] .
```

OWL vs Traditional Schema

Database Schema

- Defines structure only
- No logical inference
- Validation done by application code

OWL Ontology

- Defines structure + meaning
- Supports reasoning
- Validation done by reasoner

OWL acts as knowledge logic layer.

Why OWL Is Powerful

OWL provides:

- Semantic meaning
- Automatic classification
- Consistency checking
- Inference generation
- Interoperability across systems

This makes it ideal for:

- Knowledge Graphs
- AI reasoning systems
- Data integration
- Scientific ontologies

Protégé and OWL

- Protégé is a GUI tool
- OWL is the underlying representation language

Relationship:

Protégé (Editor) → OWL (Ontology Language)

Protégé helps humans create OWL without writing XML manually.

Making Ontology in Protégé

1. File → New → IRI: http://example.org/food2
2. Import Ontology (for example, foodpoison.ttl)