# Modeling Knowledge for Industrial Diagnostics

## Objective

In this final combined exercise, we will put in practice all our Web Semantics and Linked Data knowledge.

In this final practical exercise, you will integrate all the concepts explored throughout the Web Semantics & Linked Data unit: **RDF, RDFS, SPARQL, OWL, logical constraints, reasoning, ontology modeling, and conceptual analysis.**

You will design a full industrial diagnostics ontology, expressing:

- What entities exist in a diagnostic system;
- How they relate;
- What rules and constraints govern them;
- How reasoning can derive new knowledge or detect inconsistencies.

This mirrors real-world industrial AI systems, especially those used for predictive and corrective maintenance, where:

- Text, logs, or sensor descriptions (unstructured knowledge) must be turned into structured semantic models
- Machines must infer causes, detect inconsistencies, and propose actions

This also naturally reflects a neuro-symbolic workflow:

- *Neural-like phase:* you extract concepts, roles, and causal structures from descriptions.
- *Symbolic phase:* the ontology formalism enforces structure and enables reasoning.

In this exercise, you are the "neural extractor" and Protégé + OWL is the "symbolic reasoner." In any other exercise, you can ask a language model to be the "neural extractor."

## PART I - Conceptual Modelling: think like a knowledge engineer

We will start by representing an industrial diagnostics system conceptually, i.e., identifying *entities*, *relationships*, and *logical rules* that define how things connect.

In the following exercises, consider the **Cooling Unit of an Industrial Machine** while developing the following exercises.

## Identify the Concepts (Classes)

Write **6–8 nouns** representing the *types of things* that exist in your diagnostic system.

Examples: `Machine, Component, Sensor, Symptom, Cause, Procedure, Operator.`

*Note:* Think of “classes” as categories of things your system must know about.

## Identify the Relationships (Object Properties)

Connect your concepts with *verbs or prepositions* that describe how they relate to each other.

Examples:
```
Machine hasComponent Component
Component monitoredBy Sensor
```

## Add Attributes (Data Properties)

Identify the measurable or descriptive features of your domain. These describe characteristics of things, not relations between them.

Examples:
```
Sensor hasValue 86.4
Procedure hasDuration 2h
```

## Write 3–5 Triples

Express a few of your facts as RDF-like statements following the pattern *subject–predicate–object*.

Examples:

```
:CloggedFilter :causesSymptom :LowCoolantFlow .

:LowCoolantFlow :causesSymptom :Overheat .

:Fan :isPartOf :CoolingSystem .
```

## Sketch your ontology

Use your concepts, relationships, and triples to draw a simple diagram showing how things connect. Add direction and multiplicity if you know them (e.g., "each Machine has multiple Components").

## Reason

Now imagine your ontology is loaded into a diagnostic reasoning engine. Which questions would you expect an intelligent system to answer once this knowledge is represented?

Create at least four questions of different types:

- **Factual:** can be directly retrieved from data (e.g.: *Which components belong to SpindleMachine?)*
- **Relational:** depend on connections between entities (e.g.: *Which symptoms are caused by multiple causes?)*
- **Inferred:** require reasoning or transitive logic (e.g.: *If FanFault causes LowAirFlow and LowAirFlow causes Overheat, what can be inferred about FanFault?)*
- **Constraint Checking:** test logical consistency (e.g.: *Does every Machine have at least one Component?)*

## Define One Rule or Restriction

Create one logical rule or restriction that would help your ontology reason or maintain consistency.

Examples:

"Every *Procedure* mitigates exactly one *Cause*."

"No *Component* can be both *Working* and *Failed*."

"Every *Machine* must have at least one *Component*."

## PART II – Implement your ontology

Next, you will be implementing your industrial diagnostics ontology in Protégé. To do that, you will translate your conceptual model into a formal ontology using Protégé. The goal is to represent your knowledge using RDF, RDFS, and OWL, and to test how reasoning makes implicit knowledge explicit.

The use of your own classes and model, the one you created in the 1st part of this exercise, is advised. The examples used below are generic ones that serve for exercise demonstration purpose.

## Set up:

- Open Protégé (https://protege.stanford.edu/, select Desktop version).
- Save a new ontology (RDF/XML or Turtle advised)
- Set a base IRI such as: `http://example.org/[yournameORyourgroup]diagnostics#`

Optionally, but preferably, load your ontology in Python (as demoed at the end of this Part) to run reasoning and the SPARQL-like queries.

### Create Your Classes (Concepts)

Re-create the main *classes* you identified in the previous exercise

`Machine, Component, Sensor, Symptom, Cause, Procedure`

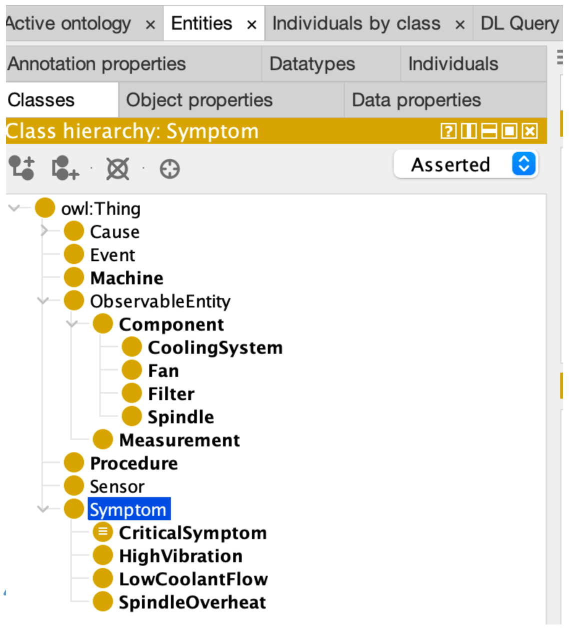
Arrange them hierarchically if possible: `CoolingSystem ⊑ Machine; Fan ⊑ Component; ...`

### Define Object Properties (Relationships)

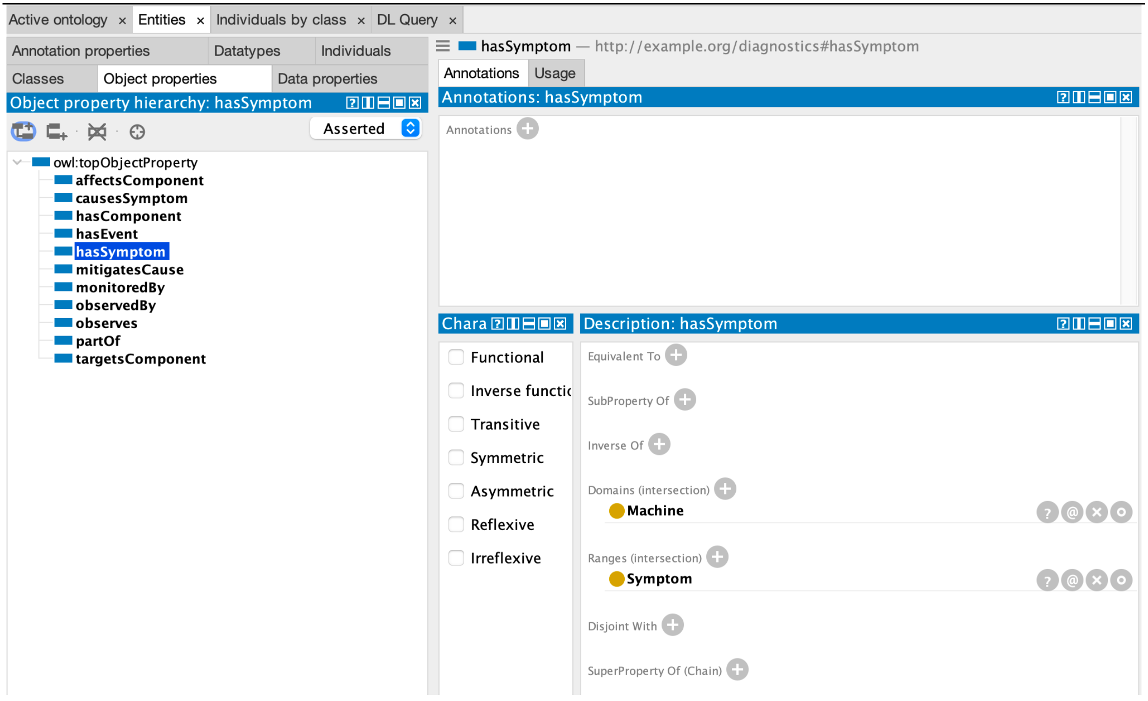
Add the verbs that link your classes and set domain and range for each property.

Examples:

- `hasComponent (domain: Machine, range: Component)`
- `partOf (domain: Component, range: Machine)`
- `causesSymptom (domain: Cause, range: Symptom)`
- `mitigatesCause (domain: Procedure, range: Cause)`
- `monitoredBy (domain: Component, range: Sensor)`
- `observes (domain: Sensor, range: ObservableEntity)`



```
owl:Thing
  Cause
  Event
  Machine
  ObservableEntity
    Component
      CoolingSystem
      Fan
      Filter
      Spindle
    Measurement
  Procedure
  Sensor
  Symptom
    CriticalSymptom
    HighVibration
    LowCoolantFlow
    SpindleOverheat
```



Active ontology × | Entities × | Individuals by class × | DL Query ×

Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Object property hierarchy: hasSymptom

Asserted

- owl:topObjectProperty
  - affectsComponent
  - causesSymptom
  - hasComponent
  - hasEvent
  - hasSymptom
  - mitigatesCause
  - monitoredBy
  - observedBy
  - observes
  - partOf
  - targetsComponent

hasSymptom — http://example.org/diagnostics#hasSymptom

Annotations | Usage

Annotations: hasSymptom

Annotations

Chara

- Functional
- Inverse functic
- Transitive
- Symmetric
- Asymmetric
- Reflexive
- Irreflexive

Description: hasSymptom

Equivalent To

SubProperty Of

Inverse Of

Domains (intersection)
Machine

Ranges (intersection)
Symptom

Disjoint With

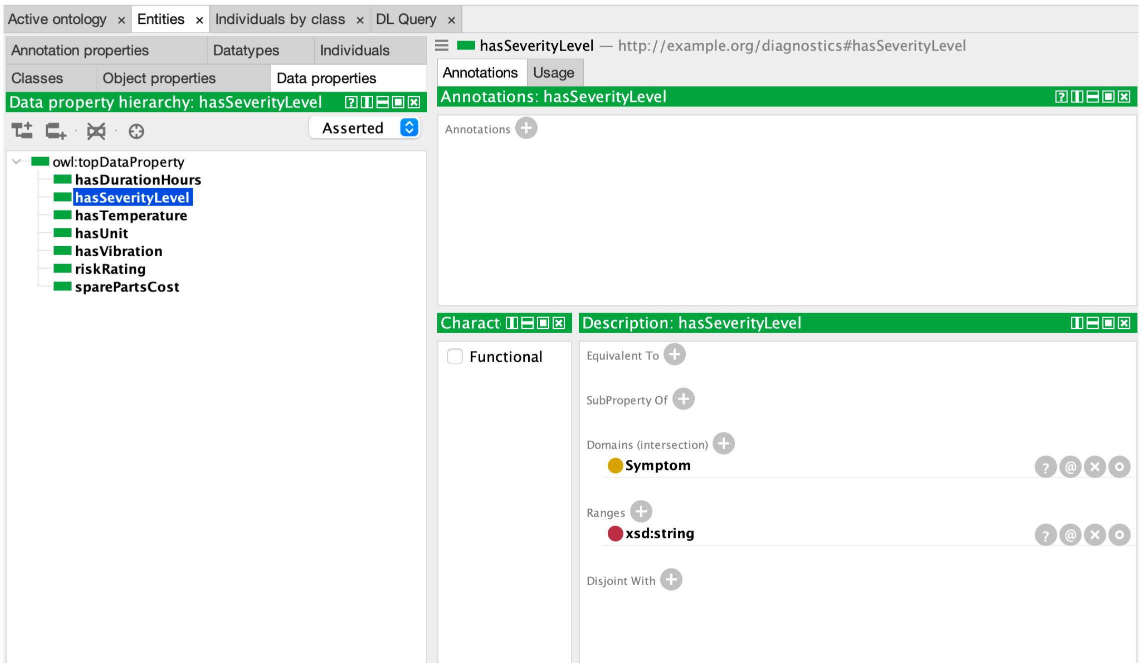
SuperProperty Of (Chain)

## Define Data Properties (Attributes)

Add data properties to hold numeric / textual data:

- `hasTemperatureValue (range: xsd:float) — e.g. for Measurement`
- `hasSeverityLevel (range: xsd:string) — e.g. for Symptom`
- `hasDurationHours (range: xsd:integer) — e.g. for Procedure`
- `hasRiskScore (range: xsd:decimal) — e.g. for Procedure or Cause`

Define them in the Data properties tab and assign appropriate domains.



hasSeverityLevel — http://example.org/diagnostics#hasSeverityLevel

- owl:topDataProperty
  - hasDurationHours
  - hasSeverityLevel
  - hasTemperature
  - hasUnit
  - hasVibration
  - riskRating
  - sparePartsCost

Annotations

Functional

Equivalent To

SubProperty Of

Domains (intersection)
Symptom

Ranges
xsd:string

Disjoint With

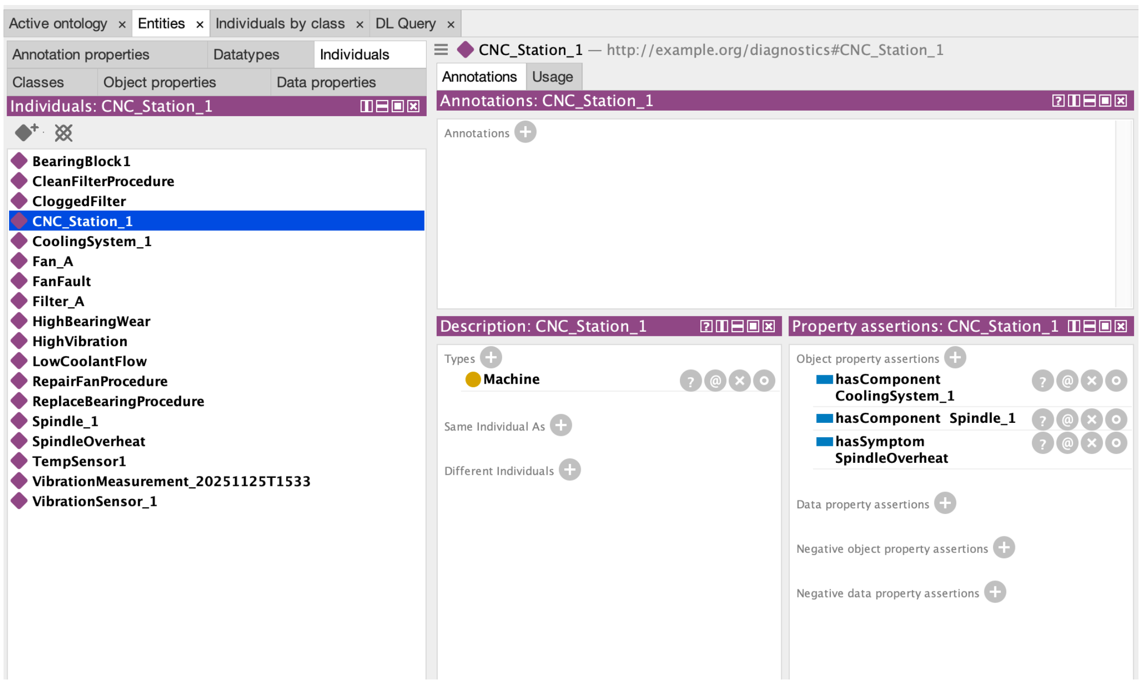
## Add Individuals (Concrete examples)

Create a few example instances to test your ontology and connect them with your object properties. For example:

- `CNC_Station_1 hasComponent Spindle_1`
- `CNC_Station_1 hasComponent CoolingSystem_1`
- `CoolingSystem_1 hasComponent Fan_A`
- `CoolingSystem_1 hasComponent Filter_A`
- `Spindle_1 hasComponent BearingBlock_1`

- `CloggedFilter causesSymptom LowCoolantFlow`
- `BearingWearHigh causesSymptom HighVibration`
- `LowCoolantFlow causesSymptom SpindleOverheat`
- `HighVibration causesSymptom SpindleOverheat`

- `CleanFilterProcedure mitigatesCause CloggedFilter`
- `ReplaceBearingProcedure mitigatesCause BearingWearHigh`
- `RepairFanProcedure mitigatesCause FanFault`



Individuals: CNC_Station_1

- BearingBlock1
- CleanFilterProcedure
- CloggedFilter
- CNC_Station_1
- CoolingSystem_1
- Fan_A
- FanFault
- Filter_A
- HighBearingWear
- HighVibration
- LowCoolantFlow
- RepairFanProcedure
- ReplaceBearingProcedure
- Spindle_1
- SpindleOverheat
- TempSensor1
- VibrationMeasurement_20251125T1533
- VibrationSensor_1

CNC_Station_1 — http://example.org/diagnostics#CNC_Station_1

Annotations: CNC_Station_1

Annotations

Description: CNC_Station_1

Types
Machine

Same Individual As

Different Individuals

Property assertions: CNC_Station_1

Object property assertions
hasComponent CoolingSystem_1
hasComponent Spindle_1
hasSymptom SpindleOverheat

Data property assertions

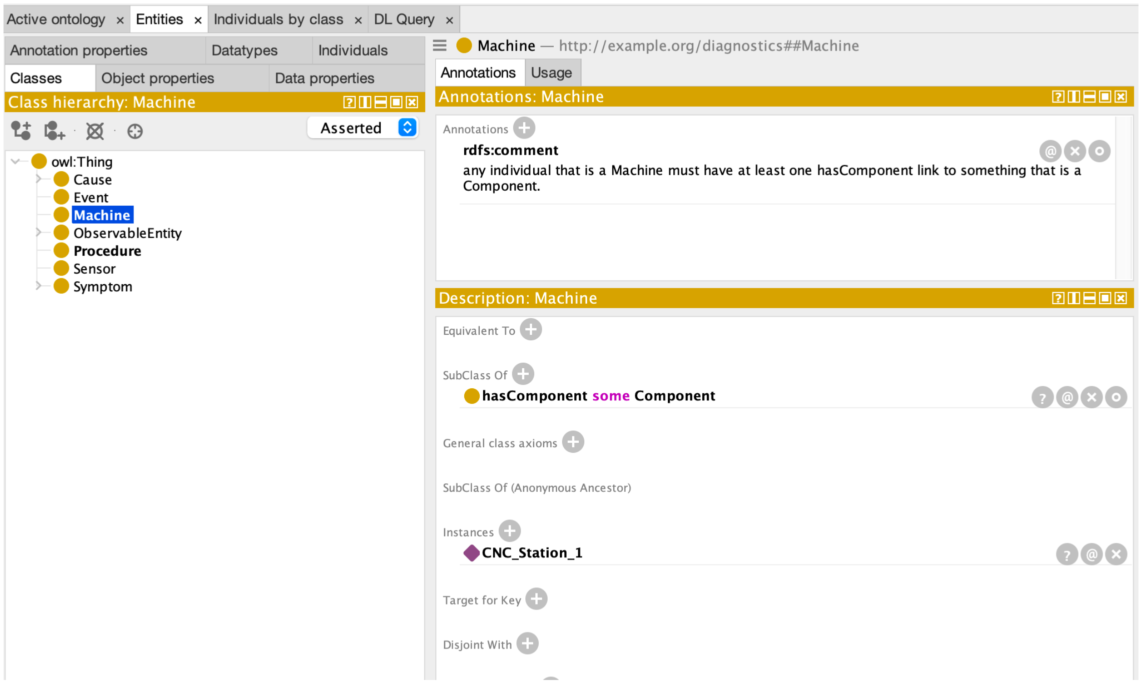
Negative object property assertions

Negative data property assertions

## Add Logical Restrictions / Axioms

To make the ontology intelligent, not just a glossary, you need to add the axioms you identified in the first part of this exercise. Use the *Class Expression Editor* available in the 'SubClass Of' tab for expressing your axioms.

Example: *Every Machine has at least one Component*



Machine — http://example.org/diagnostics##Machine

Annotations
rdfs:comment
any individual that is a Machine must have at least one hasComponent link to something that is a Component.

SubClass Of
hasComponent some Component

Instances
CNC_Station_1

## Run the Reasoner

Check for:

- Inferred subclasses
- Inferred types of individuals
- If any class becomes *unsatisfiable*
- If any individual becomes inconsistent

Common issues include:

- Missing domain/range
- Misuse of inverses
- Disjointness violations
- Data property inconsistencies

### OPTIONAL – Use Python to Explore Reasoning

You can load your OWL ontology in Python[1] and inspect some inferences programmatically.

```
pip install owlready2
```

Example script:

```python
from owlready2 import get_ontology, sync_reasoner

# Adjust path to your .owl file
onto = get_ontology("file:///absolute/path/to/diagnostics_groupX.owl").load()

with onto:
    sync_reasoner()  # runs the reasoner

# Access classes
Machine = onto.Machine
OverheatedMachine = onto.OverheatedMachine

print("All machines:")
for m in Machine.instances():
    print(" -", m)

print("\nOverheated machines (inferred):")
for m in OverheatedMachine.instances():
    print(" -", m)
```

To iterate relationships:

```python
for proc in onto.Procedure.instances():
    for cause in proc.mitigatesCause:
        print(proc.name, "mitigates", cause.name)
```

## PART III — SPARQL Querying & Reasoning over Your Ontology

Create and run SPARQL queries on your own ontology (the questions below must be adapted to *your* classes, individuals, and property names).

You may use the SPARQL tab in Protégé, or use Python + rdflib (see OPTIONAL code example below).

When possible, compare results before and after running the reasoner.

[1] https://owlready2.readthedocs.io/en/v0.49/

1. Check whether your *structural modeling* (classes, properties, part–whole relations) is correct.
    a. List all components of your main machine.
    b. List all subcomponents of your cooling system.

       If you modeled a hierarchy (CoolingSystem → Fan → Bearing), use your part–whole relation.

       If it is transitive, some subcomponents may only appear after reasoning.

    c. List all sensors and what they observe.

       Return pairs *(sensor, observed entity)* using your property (e.g., :observes).

       Check if every sensor is connected to something meaningful.

2. Diagnostics Queries
    a. List all causes and the symptoms they generate.
    b. List all procedures and which cause they mitigate.
3. Multi-Step & Inferred Queries

   Example: Find all causes that are connected to SpindleOverheat through one or more intermediate symptoms or conditions.

4. Finally, write SPARQL versions of the questions you identified in Part I (your competency questions), for example:
    a. "Which components are monitored by sensors associated with abnormal vibration?"
    b. "Which procedures mitigate the root causes of spindle overheat?"
    c. "Which causes affect more than one component?"

## OPTIONAL - Python Reasoning & Querying (example code)

```
pip install rdflib owlready2 owlrl
```

Load and inspect the ontology:

```python
from owlready2 import get_ontology, sync_reasoner

onto = get_ontology("file:///absolute/path/to/diagnostics_groupX.owl").load()

with onto:
    sync_reasoner()

OverheatedMachine = onto.OverheatedMachine

print("Inferred overheated machines:")
for m in OverheatedMachine.instances():
    print(m)
```

Example SPARQL-like query in Python using RDFLib:

```python
from owlready2 import get_ontology, sync_reasoner

onto = get_ontology("file:///absolute/path/to/diagnostics_groupX.owl").load()

with onto:
    sync_reasoner()

OverheatedMachine = onto.OverheatedMachine

print("Inferred overheated machines:")
for m in OverheatedMachine.instances():
    print(m)
```