



## **Protégé** **Ontology Building Environment**

**Prof Christophe Roche**  
ERA Chair Holder - University of Crete  
<https://christophe-roche.fr/>

May 2025



Horizon ERA Chair TALOS AI4SSH Project funded by the European Commission  
Grant Agreement n° 101087269, <https://cordis.europa.eu/project/id/101087269>

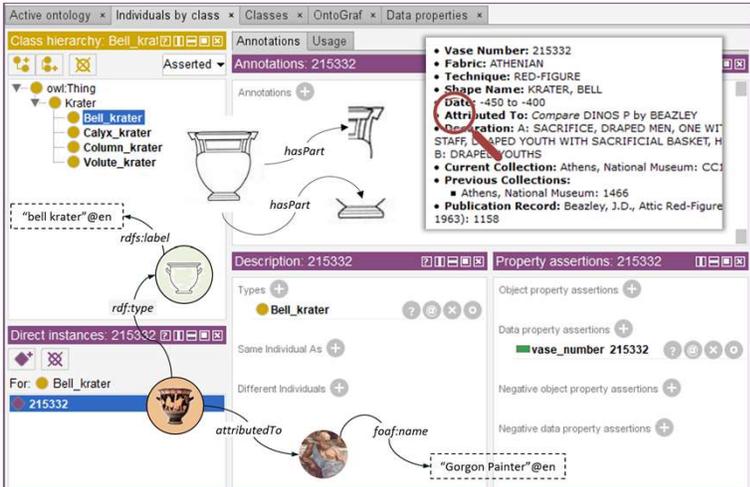
This MOOC is dedicated to Protégé, the currently most widely used ontology building environment.

1. Definitions 2. Principles 3. Example 4. Ontology building 5. Open Questions 6. Conclusion



## Contents

- 0. Introduction**
- 1. Definitions**
  - Ontology
- 2. Principles**
  - Individual
  - Property
  - Class
- 3. Example**
  - Greek Vases
- 4. Ontology Building**
  - Class Hierarchy
  - Property Restriction
  - Populating
  - Annotating
- 5. Open Questions**
  - Definition
  - Linguistic Dimension
- 6. Conclusion**



TALOS ERA Chair AI for SSH – Project n° 101087269

“Protégé” Christophe Roche

CC BY-NC-ND  2

The MOOC is divided into 6 parts. We’ll begin by recalling some definitions, followed by the theoretical principles on which Protégé is based. We’ll then see how to use Protégé to build an ontology with an example from Digital Humanities. We’ll conclude with two open questions.

# 0. Introduction



<https://protege.stanford.edu/>

- ✓ Protégé is a free, open-source ontology editor written in Java developed at Stanford University
- ✓ More than 300,000 users are registered.



- ✓ *W3C standards compliant*
- ✓ *Customizable user interface*
- ✓ *Visualization support*
- ✓ *Ontology refactoring support*
- ✓ *Direct interface to reasoners*
- ✓ *Highly pluggable architecture*
- ✓ *Cross compatible with WebProtégé*

## WHY PROTÉGÉ

Protégé's plug-in architecture can be adapted to build both simple and complex ontology-based applications. Developers can integrate the output of Protégé with rule systems or other problem solvers to construct a wide range of intelligent systems. Most important, the Stanford team and the vast Protégé community are here to help.

 <b>ACTIVE COMMUNITY</b> Protégé is actively supported by a strong community of users and developers that field questions, write documentation, and contribute plug-ins.	 <b>W3C STANDARDS SUPPORT</b> Protégé fully supports the latest OWL 2 Web Ontology Language and RDF specifications from the World Wide Web Consortium.	 <b>EXTENSIBLE OPEN SOURCE ENVIRONMENT</b> Protégé is based on Java, is extensible, and provides a plug-and-play environment that makes it a flexible base for rapid prototyping and application development.
--	--	---

Protégé is the most widely used ontology-building environment. Written in Java, it is a free, open-source software developed by Stanford University. It is supported by a large community of users. Protégé is based on the first-order logic that allows the use of reasoners to verify the consistency of ontologies. Protégé supports W3C formats such as OWL, which allows ontologies to be exported as RDF knowledge graphs.



# 1. Definitions

**Ontology** An ontology is a formally-defined vocabulary for a particular domain of interest. Ontologies are typically based on a class hierarchy (asserted and/or inferred), supplemented by assorted properties.

[https://protegewiki.stanford.edu/wiki/Pr4\\_UG\\_mi\\_Glossary#Ontology](https://protegewiki.stanford.edu/wiki/Pr4_UG_mi_Glossary#Ontology)



**Ontologies are used to capture knowledge about some domain of interest. An ontology describes the concepts in the domain and also the relationships that hold between those concepts.**

*A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools - Edition 1.3*

**Language** OWL provides the theoretical basis for Protégé ontologies.

[https://protegewiki.stanford.edu/wiki/Pr4\\_UG\\_mi\\_Glossary#Ontology](https://protegewiki.stanford.edu/wiki/Pr4_UG_mi_Glossary#Ontology)

Different ontology languages provide different facilities. The most recent development in standard ontology languages is OWL from the World Wide Web Consortium (W3C).

*A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools - Edition 1.3*

**Reasoner** The logical model allows the use of a reasoner which can check whether or not all of the statements and definitions in the ontology are mutually consistent and can also recognise which concepts fit under which definitions. The reasoner can therefore help to maintain the hierarchy correctly.

*A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools - Edition 1.3*



**an inconsistent class is a class which cannot contain any individual because of its definition**



Let's start by recalling what an ontology is. Some define an ontology as a formally defined vocabulary. But it is above all a specification of a conceptualization used to represent the knowledge of a domain. An ontology defines the concepts of a domain and the relationships between these concepts.

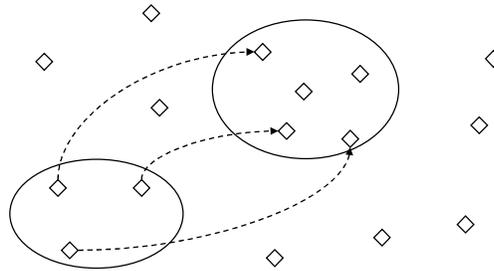
These definitions are written using a knowledge representation language, including the W3C standard Ontology Web Language (OWL).

Logical foundations allow the use of reasoners to verify, for example, the consistency of an ontology.

## 2. Principles

### Extensional Logic

**Goal:** Organising the objects which populate the world into **classes** according to the **relationships** that linked objects together



An object is not defined by its “nature”, but by its relations with other objects

To master a tool, whatever it is, it is mandatory to master the principles on which it is based. Protégé is based on an extensional logic, that is, on manipulating objects, more generally called individuals. The goal is to organise the objects that populate a reality into sets, called classes, according to the relationships linking the objects to each other. The notion of Class replaces that of Concept. Thus, an object is not defined by its nature, but by its relationships with other objects.



## 2. Principles

### Components of OWL Ontologies

#### 1) Individuals

Individuals, represent objects in the domain in which we are interested



Terminology: « individual », « instance », « object »

The first principle is therefore the notion of individual, or object, even if it is not necessary to create individuals to define classes. Individuals can represent any object in the field of application.

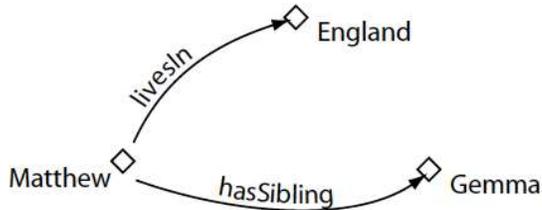
## 2. Principles



### Components of OWL Ontologies

#### 2) Object Properties

Properties are binary relations on individuals, i.e. properties link two individuals together.



Terminology: « properties », « slots » (Protégé), « roles » (DL), « relations », « attributes »



The second principle is that of relation or property. Properties are binary relationships between two individuals. The same individual can, as in this example, be linked to several individuals, whether through the same relationship or different relationships. Relationships are always oriented and named.

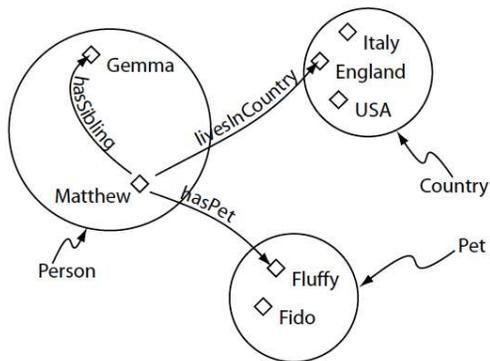
## 2. Principles



### Components of OWL Ontologies

#### 3) Classes

OWL classes are interpreted as sets that contain individuals.



Classes are a concrete representation of concepts.

Classes are defined using formal descriptions that state precisely the requirements for membership of the class.

Person = ?

Person = {  $x / \exists y \text{ Country}(y) \wedge \text{livesInCountry}(x,y)$  }

Person = {  $x / \exists y \text{ Pet}(y) \wedge \text{hasPet}(x,y)$  }

Person = {  $x / \exists y \text{ hasSibling}(x,y) \vee \text{hasSibling}(y,x)$  }

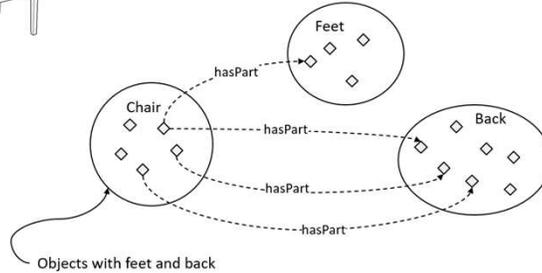
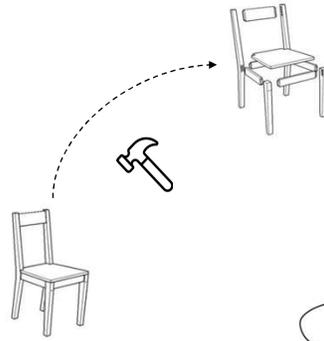


It is then possible to define classes, sets of individuals, according to the relationships linking individuals. So, in this example, a person is not defined according to his nature but according to his relationships. It is then possible to propose different definitions of the Person class. These definitions are not necessarily equivalent.

## 2. Principles



Change your way of thinking



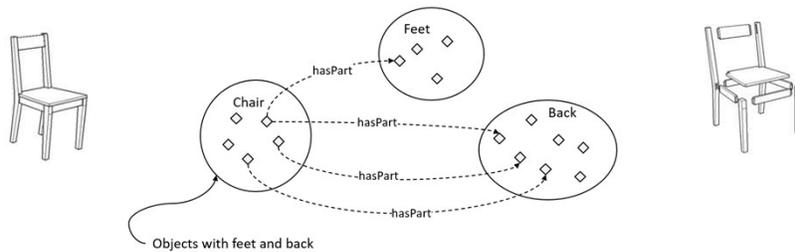
This approach to definition requires changing the way we think about things. A thing is no longer a whole, but a set of related entities. Some domains are well-suited to this approach. For example, a chair is an individual linked to its parts, its backrest, its legs, and so on.

## 2. Principles



### 4) Property restriction A means to define classes of individuals

a) **Existential Restrictions:** describes (anonymous) classes of individuals that participate in *at least one* (some) relationship along a specified property to individuals that are members of a specified class.



b) **Universal Restrictions:** describes (anonymous) classes of individuals that for a given property *only* (only) have relationships along this property to individuals that are members of a specified class (all values of the property must be of a certain type)

c) **Has value:** at least one of the values of the property is a certain value

A class is then defined as a set of property restrictions, that is, restrictions on the objects that can be linked by the property. Restricting the hasPart relationship to the Back Class allows to define the class of individuals which have a Back of which the Chair Class is a subset.

### 3. Example

#### The Krater Ontology

"The term '**krater**' suggests a mixing-vessel (compare Greek *kerannumi* - to mix), and we know that the wine served at the symposium was mixed with water.



**Column-krater:** Named for its column-like handles, the column-krater is first known from Corinthian examples dated to the late seventh century. It is regularly produced by Athenian potters from the first half of the sixth-century until the third quarter of the fifth. It seems from graffiti on Athenian red-figure examples that the vessel was referred to as *Korinthios* or *Korinthiourges*.



**Volute-krater:** The volute-krater is named after its handles. The François Vase is a famous and early example, but the typical Athenian form occurs only later in the sixth century, with the handles tightly curled so that they look like the volutes on Ionic columns. The shape is also found in metal. Over the course of the fifth and fourth centuries, examples become slimmer, and Apulian volute-kraters from South Italy are particularly elaborate.



**Calyx-krater:** The handles of the calyx-krater are placed low down on the body, at what is termed the *cul*. Their upward curling form lends the shape an appearance reminiscent of the calyx of a flower, hence the name. The earliest known example was possibly made by Exekias in the third quarter of the sixth century. It continues to be produced, mainly in red-figure, becoming more elongated over the course of the fifth and fourth centuries.



**Bell-krater:** The latest of the four krater-types, it first occurs in the early fifth century, and is not found decorated in black-figure. It is named for its bell-like shape, perhaps originating in wood. It has small horizontal upturned handles just over halfway up the body. Some do not have a foot, and earlier examples may have lugs for handles.



Let us take for example the definition in Protégé of the ontology of Greek vases, and more precisely, the ontology of kraters as defined in the Beazley archive. A krater is an ancient Greek vessel used to mix water and wine. There are different types of kraters: column, volute, calyx, bell craters, etc. These types of kraters are distinguished by the type of their handles or their shape.

## 4. Ontology Building



Metrics	
Axiom	0
Logical axiom count	0
Declaration axioms count	0
Class count	0
Object property count	0
Data property count	0
Individual count	0
Annotation Property count	0

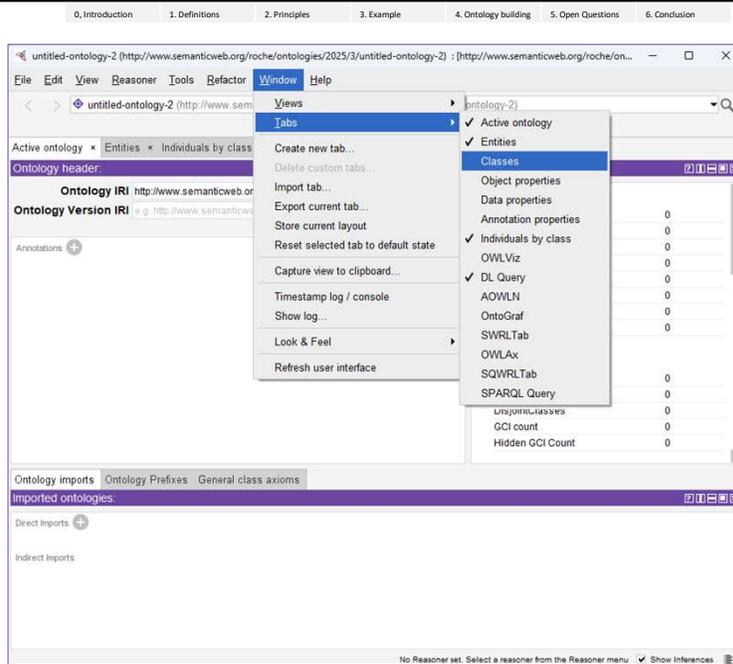
  

Class axioms	
SubClassOf	0
EquivalentClasses	0
DisjointClasses	0
GCI count	0
Hidden GCI Count	0



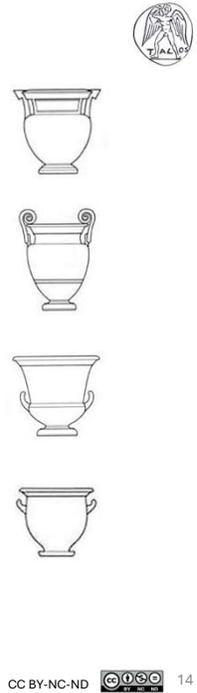
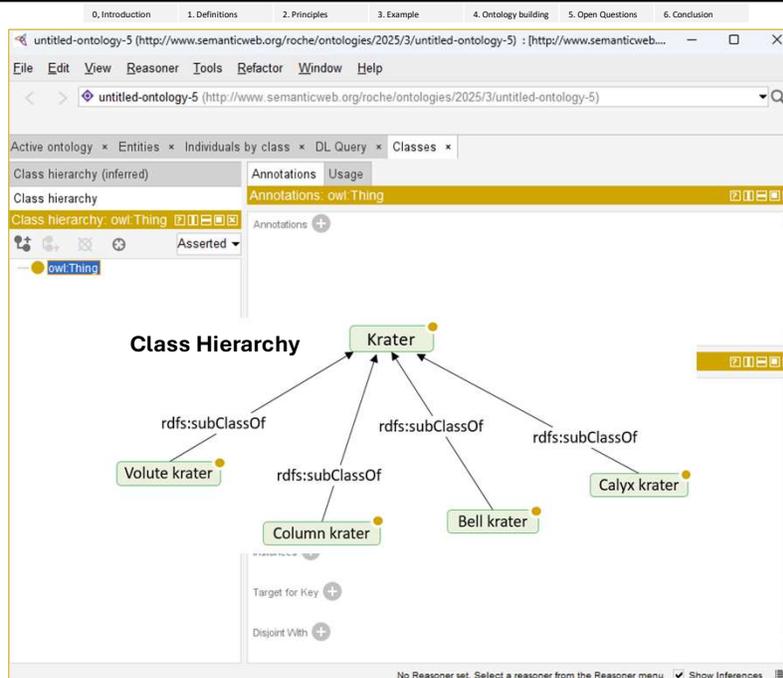
When Protégé is opened, a window is displayed with different tabs. Each tab is divided into views. The first tab named 'Active ontology' lists information about the ontology in use such as the number of classes, individuals, properties, etc.

## 4. Ontology Building



The definition of classes is done using a dedicated tab that must be created, as illustrated in the figure of slide 13

## 4. Ontology Building



The ontology of kraters is a set of classes organized in a hierarchy according to the set relationship of inclusion: column krater is a subclass of the class krater.

0. Introduction 1. Definitions 2. Principles 3. Example 4. Ontology building 5. Open Questions 6. Conclusion

## 4. Ontology Building

Class hierarchy (inferred) Annotations Usage

Class hierarchy Annotations: owl:Thing

Class hierarchy: owl:Thing Annotations +

owl:Thing

Asserted

Create a new Class

Name: Krater

IRI: http://www.semanticweb.org/roche/ontologies/2025/3/untitled-ontology-5#Krater

New entity options...

OK Annuler

Class Hierarchy

Krater

subClass Of (Anonymous Ancestor)

stances +

target for Key +

isjoint With +

rdfs:subClassOf

rdfs:subClassOf

rdfs:subClassOf

rdfs:subClassOf

Volute krater

Column krater

Bell krater

Calyx krater

No Reasoner set. Select a reasoner from the Reasoner menu Show Inferences

TALOS ERA Chair AI for SSH – Project n° 101087269 "Protégé" Christophe Roche CC BY-NC-ND 15

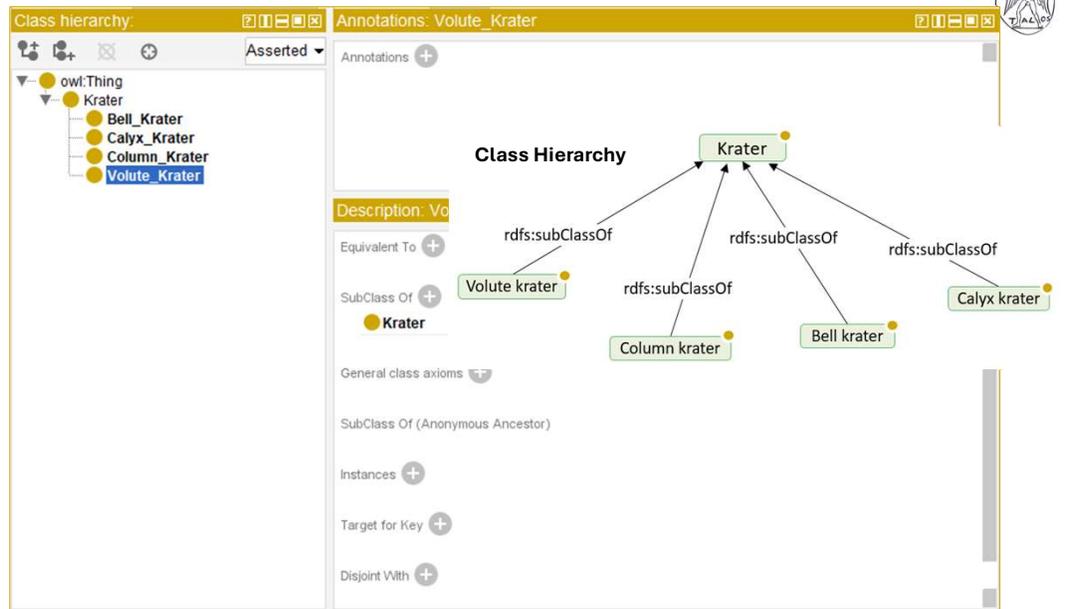
Every class is a subclass of the Thing root class, which groups all objects. The creation of the Krater class is therefore done from the Thing root class.

## 4. Ontology Building: Class Hierarchy

The screenshot displays the Protégé ontology editor interface. On the left, the 'Class Hierarchy' view shows a tree structure where 'Krater' is the parent class, and 'Volute krater', 'Column krater', 'Bell krater', and 'Calyx krater' are its subclasses. The 'Krater' class is highlighted. In the center, a 'Create a new Class' dialog box is open, showing the name 'Bell Krater' and the IRI 'http://www.semanticweb.org/roche/ontologies/2025/3/untitled-ontology-6#Bell\_Krater'. The dialog has 'OK' and 'Annuler' buttons. On the right, the 'Annotations' panel shows the 'Krater' class with a list of annotations, including 'owl:Thing'. The Protégé logo is visible on the left side of the interface. At the bottom, there is a footer with the text 'TALOS ERA Chair AI for SSH - Project n° 101087269', 'Protégé Christophe Roche', and 'CC BY-NC-ND' with a Creative Commons license icon.

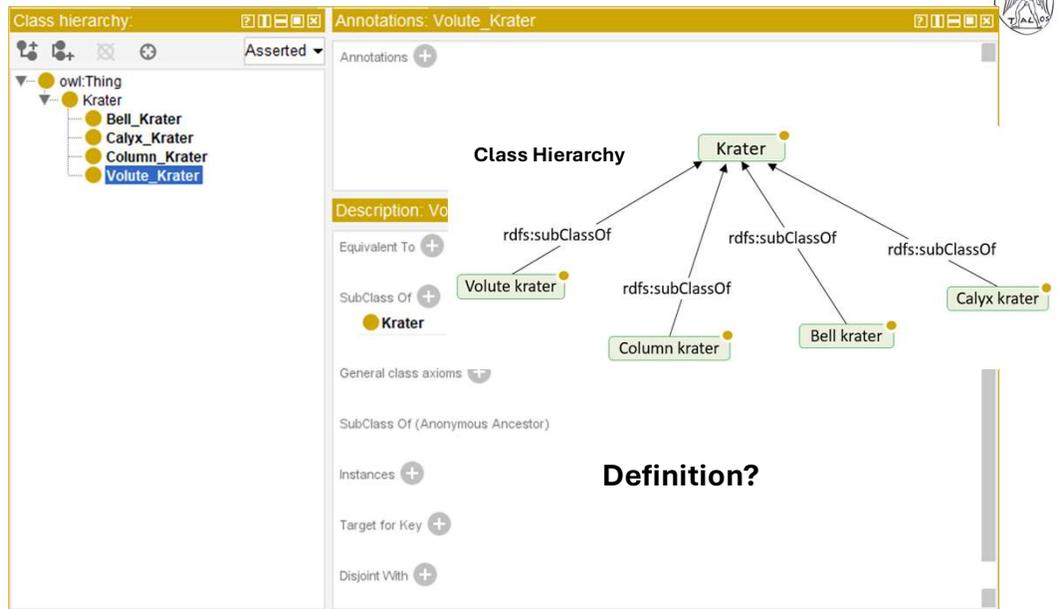
The different types of Kraters are subclasses of the Krater class. They are created in a similar way to what we have just seen.

## 4. Ontology Building



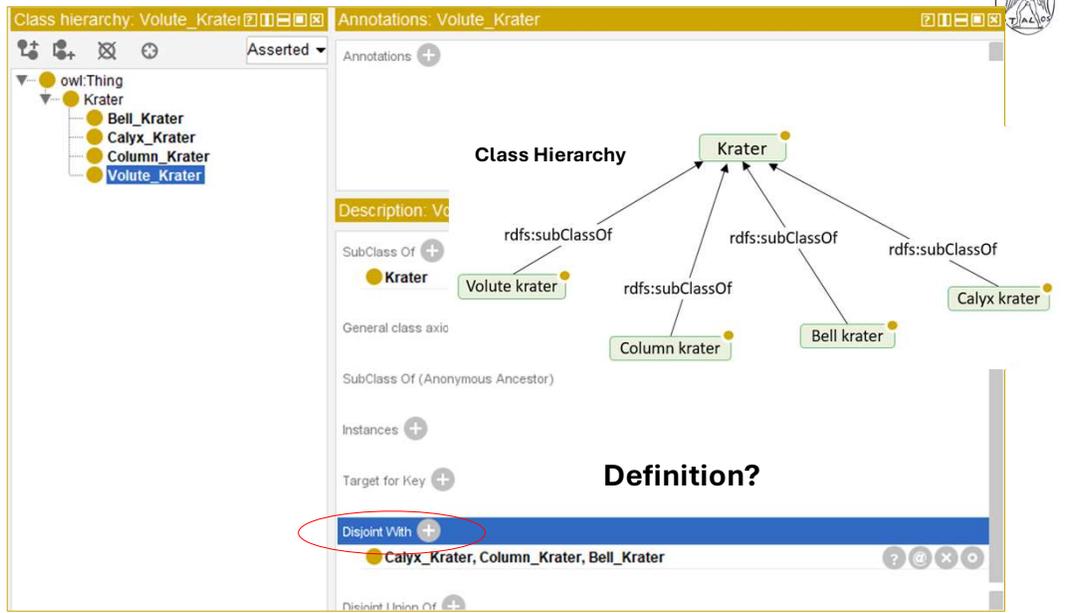
The result is the class hierarchy of slide 17

## 4. Ontology Building



Defining the different types of kraters as subclasses of the class krater does not define them. Indeed, nothing distinguishes, except their name, the different types of kraters. They still need to be defined more precisely.

## 4. Ontology Building

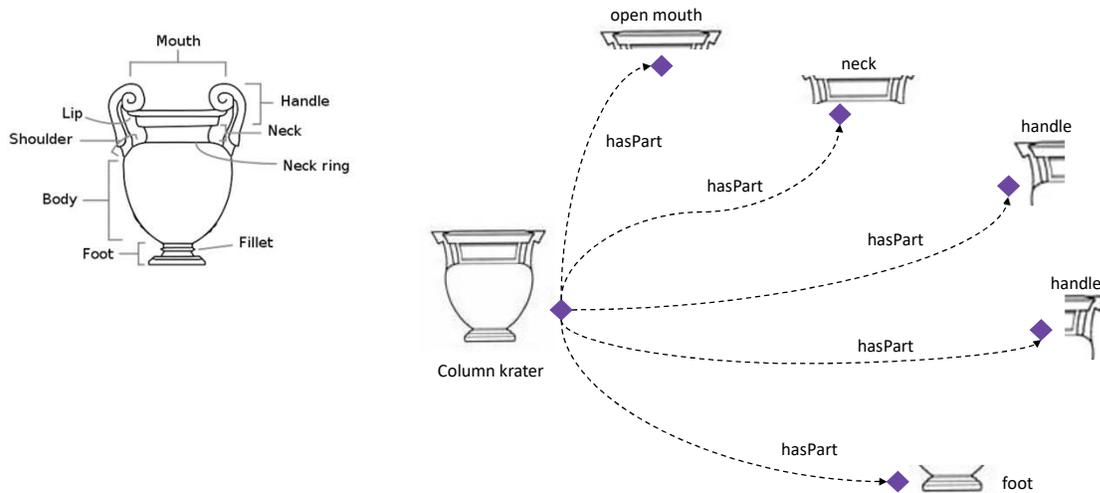


A first property is to declare the different classes of kraters as disjoint: a given krater can only be of one type and therefore belong to only one class. But that still doesn't define what a particular krater is.

## Object Properties



### Relationships between individuals



Since a class is defined on the basis of relations between individuals, a krater must be thought of not as a whole, but as a set of individuals, for example as the set of its parts: A column krater is linked by the property `hasPart` to a foot, a neck, two column-like handles, etc.

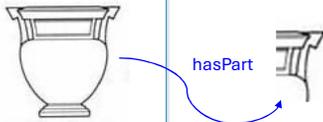
## 4. Ontology Building



The screenshot shows the Protégé ontology editor interface. The left panel, titled "Class hierarchy: Part", displays a tree structure of classes. The "Part" class is highlighted with a red box and contains subclasses: "Foot", "Handle", "Column\_like\_Handle", "Upward\_Curling\_Handle", "Volute\_like\_Handle", "Mouth", "Close\_Mouth", "Open\_Mouth", and "Neck". The right panel, titled "Description: Part", shows a diagram of a "Column krater" with its parts: "open mouth", "neck", "handle", and "foot", connected by "hasPart" relationships.

It is therefore necessary to define a new class, the class of parts, disjoint from the class of kraters. The different parts, Foot, Handle, Neck, etc., are subclasses of the class Part.

## 4. Ontology Building



The screenshot displays the Protégé ontology editor interface. On the left, the 'Object property hierarchy' pane shows 'owl:topObjectProperty' expanded to 'hasPart'. The main workspace shows a diagram of a vase with a handle, connected by a blue arrow labeled 'hasPart'. On the right, the 'Annotations: hasPart' and 'Description: hasPart' panes are visible. The 'Description: hasPart' pane shows the following configuration:

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

Under 'Domains (intersection)', the class 'Krater' is selected. Under 'Ranges (intersection)', the class 'Part' is selected.

The next step is to set a new property between objects, the hasPart relationship. This relationship links individuals of the krater class, which is the domain of the relationship, to individuals of the class Part, the range of the relationship, which includes all the individuals of its subclasses: Foot, Handle, Neck, etc.

0. Introduction 1. Definitions 2. Principles 3. Example 4. Ontology building 5. Open Questions 6. Conclusion

**Definition of Krater Classes: Property Restrictions**

The screenshot displays the Protégé ontology editor interface. On the left, a class hierarchy shows 'Column\_krater' as a subclass of 'Krater'. The center panel shows the 'Annotations' and 'Description' for 'Column\_krater', including labels in English ('column krater') and French ('cratère à colonnettes'). The right panel, titled 'Column\_krater', is the 'Object restriction creator' window. It shows the 'Restricted property' as 'hasPart' and the 'Restriction filler' as 'Column\_like\_handle'. The 'Restriction type' is set to 'Exactly (exact cardinality)' with a cardinality of 2. A small image of a krater is shown below the class hierarchy.

TALOS ERA Chair AI for SSH – Project n° 101087269 "Protégé" Christophe Roche CC BY-NC-ND

It then remains to specify how the individuals of a class are related to its different parts through the hasPart relation. Thus, the class of column kraters is a subclass of the class of objects linked to exactly 2 column-like handles.

## Definition of Krater Classes: Property Restrictions



Class hierarchy: Column\_krater

- owl:Thing
  - Greek\_God
    - Krater
      - Bell\_krater
      - Calyx\_krater
      - Column\_krater**
      - Volute\_krater
    - Part
      - Foot
      - Handle
        - Column\_like\_handle
        - Upward\_curling\_handle
        - Volute\_like\_handle
      - Lip
      - Mouth
      - Neck

Annotations: Column\_krater

- rdfs:label [language: en] column krater
- rdfs:label [language: fr] cratère à colonnettes

Description: Column\_krater

SubClass Of

- hasPart exactly 2 Column\_like\_handle
- Krater

General class axioms

SubClass Of (Anonymous Ancestor)



Class hierarchy: Volute\_krater

- owl:Thing
  - Greek\_God
    - Krater
      - Bell\_krater
      - Calyx\_krater
      - Column\_krater
      - Volute\_krater**
    - Part
      - Foot
      - Handle
        - Column\_like\_handle
        - Upward\_curling\_handle
        - Volute\_like\_handle
      - Lip
      - Mouth
      - Neck

Annotations: Volute\_krater

- rdfs:label [language: en] volute krater
- skos:definition [language: fr] Krater with a clearly defined neck and volute-like hand

Description: Volute\_krater

Equivalent To

SubClass Of

- hasPart exactly 2 Volute\_like\_handle
- Krater

General class axioms

SubClass Of (Anonymous Ancestor)



This slide illustrates the formal definitions of the column krater and the volute krater classes.

## 4. Ontology Building: Annotating

The screenshot displays the Protégé ontology editor interface. On the left, a class hierarchy shows 'owl:Thing' as the root, with 'Krater' as a child, and 'Bell\_krater', 'Calyx\_krater', 'Column\_krater', and 'Volute\_krater' as sub-classes of 'Krater'. The 'Annotations' tab for 'Bell\_krater' is active, showing a list of annotations with a red arrow pointing to a '+' icon for adding a new annotation. A 'Description: Bell\_krater' dialog box is open, showing a list of properties with 'rdfs:label' selected and highlighted by a red arrow. The dialog also shows a text input field containing 'bell krater', a 'Type' dropdown, and a 'Lang' dropdown set to 'en'. To the right of the dialog, a magnifying glass icon is shown with a blue dashed arrow pointing to the text 'Bell krater' and another blue dashed arrow pointing to the text '"bell krater"@en "cratère en cloche"@fr".

TALOS ERA Chair AI for SSH – Project n° 101087269

"Protégé" Christophe Roche

CC BY-NC-ND 25

It is also possible to associate different information with classes in the form of annotations. For example, associating a term designating the class in different languages. The use of W3C vocabularies facilitates the exchange and interoperability of ontologies. In our example, the terms "bell krater" in English and "cratère en cloche" in French refer through the `rdfs:label` property to the class `Bell_Krater`

## 4. Ontology Building: Annotating



The screenshot displays the Protégé interface with the following components:

- Class hierarchy:** A tree view showing the ontology structure. The hierarchy is: owl:Thing (parent) -> Krater (child) -> Bell\_Krater (child of Krater). Other children of Krater include Calyx\_Krater, Column\_Krater, and Volute\_Krater. Part is another child of owl:Thing.
- Annotations: Bell\_Krater:** A panel showing two annotations:
  - rdfs:label** [language: fr] with the value "cratère en cloche".
  - rdfs:label** [language: en] with the value "bell krater".
- Description: Bell\_Krater:** A panel showing class relationships:
  - Equivalent To:** (empty)
  - SubClass Of:** Krater
  - General class axioms:** (empty)
  - SubClass Of (Anonymous Ancestor):** (empty)
  - Instances:** (empty)



...

## 4. Ontology Building: Annotating

The screenshot shows the Protégé ontology editor interface. On the left, the 'Class hierarchy' panel displays a tree structure starting with 'owl:Thing', followed by 'Krater', and then its subclasses: 'Bell\_Krater', 'Calyx\_Krater', 'Column\_Krater', and 'Volute\_Krater'. The 'Bell\_Krater' class is selected. The main area is divided into two panes: 'Annotations: Bell\_Krater' and 'Description: Bell\_Krater'. The 'Annotations' pane shows three annotations: an 'rdfs:label' in French ('cratère à cloche'), an 'rdfs:label' in English ('bell krater'), and an 'skos:definition' in English describing the object's shape and handles. Below this, an 'rdfs:seeAlso' annotation points to a URL. The 'Description' pane shows 'Krater' as a subclass, 'Calyx\_Krater, Column\_Krater, Volute\_Krater' as disjoint classes, and other class-related information.

The diagram illustrates the annotation of the 'Bell krater' class. A green box labeled 'Bell krater' is connected to three annotations:

- A blue dashed arrow labeled 'rdfs:label!' points to the text '"bell krater"@en' and '"cratère en cloche"@fr'.
- A black dashed arrow labeled 'skos:definition' points to the text '"Krater with a clearly defined neck and volute-like handles."'.
- A black dashed arrow labeled 'rdfs:seeAlso' points to the URL 'https://www.carc.ox.ac.uk/carc/resources/Introduction-to-Greek-Pottery/Shapes/Kraters'.

TALOS ERA Chair AI for SSH – Project n° 101087269 "Protégé" Christophe Roche CC BY-NC-ND 27

It is also possible, using annotations and standardized vocabularies such as SKOS, to associate a definition using the `skos:definition` property and to link a class to external resources using the `rdfs:seeAlso` property

## 4. Ontology Building: Populating



The screenshot shows the Protégé ontology editor interface. The left pane displays the class hierarchy with 'Calyx\_krater' selected. The main area shows the 'Annotations' for the instance 'Beazley-215424', including its label, URIs, and a foaf:depiction. The right pane shows 'Property assertions' for the instance, including 'technique "Red-Figure"@en' and 'fabric "Athenian"@en'. Below the screenshot, there is a small image of a vase and its metadata.

**Vase Number:** 215424  
**Fabric:** ATHENIAN  
**Technique:** RED-FIGURE  
**Shape Name:** KRATER, CALYX  
**Date:** -450 to -400

TALOS ERA Chair AI for SSH – Project n° 101087269  
"Protégé" Christophe Roche  
CC BY-NC-ND 28

A particular vase, for example, the krater reference 215424 in the Beazley archive, will be represented by an individual instance of the calyx krater class. Its attributes, such as the technique used, are represented using data properties, linking the instance not to another object as with object properties, but to data such as a string or numeric value

## 5. Open Questions



Definition: { essential characteristics }

Amphora

*for storing and transport*



*without neck*

Bell\_Krater ::= { *for\_mixing\_wine\_and\_water* , *without\_neck* ,  
*with\_foot* , *with\_open\_mouth* ,  
*with\_upward\_curling\_handles* }

Krater



*for mixing wine and water*



Protégé is an extremely powerful tool. However, some questions remain open concerning the notion of definition and the linguistic dimension associated with an ontology. Considering an object as a set of related entities does not represent all the knowledge that defines a class. If the absence of a part, a bell krater has no neck, can be expressed in the form of a logical property, how can the function of a vase be represented: kraters are for mixing water and wine, while amphorae are for storage and transport?

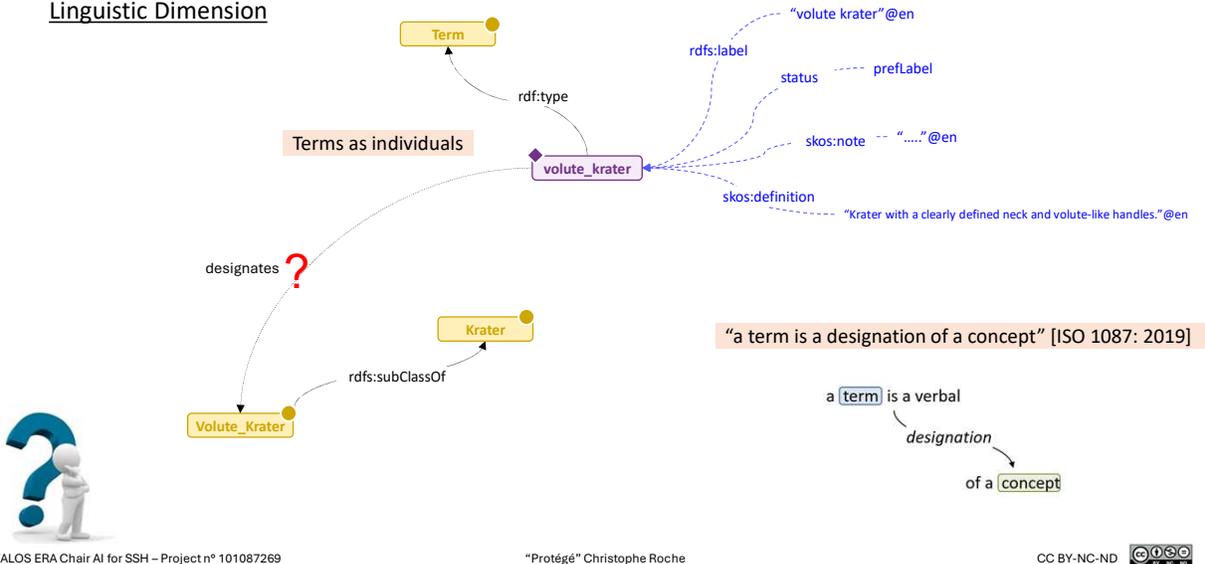
'for storage and transport' and 'for mixing water and wine' are essential characteristics, which contribute to the definition of these classes.

While the representation of essential characteristics in the form of individuals is logically correct, it is not satisfactory from the point of view of the domain knowledge.

## 5. Open Questions



### Linguistic Dimension



Concerning the linguistic dimension, a term, defined as a verbal designation of a concept, cannot be reduced to a simple label attached to a class. The term requires explicit representation in the form of an individual in order to be able to attach information such as their grammatical category (part of speech) or status. Such a representation raises problems whose solutions complicate the ontology

## 6. Conclusion



- ✓ Free environment
- ✓ Large Community
- ✓ Definition based on relations between objects (Class)
- ✓ Description Logic Reasoners
- ✓ W3C Standards compliant



- How to represent essential characteristics (Concept)?
- How to represent the linguistic dimension?
- How to take into account the way of thinking of Experts?

To conclude. Protégé is an extremely powerful open-source tool with a large user community. Compliant with W3C standards, it is based on logical foundations that allow to verify several properties, including the consistency of the ontology. Protégé is based on the notion of class, not concept. A class is defined not according to the nature of its instances, but according to the relationships that its instances have with other objects. These principles are well-suited for organizing objects into a hierarchy of classes. However, they do not always align with the way experts think. Moreover, the essential characteristics underlying concept definitions and the linguistic dimension of ontology raise issues whose solutions in Protégé are not entirely satisfactory..