# 0.1 Ontology Design Patterns

## 0.1.1 Basics

### Motivation

- It is hard to extract *only useful pieces* of comprehensive higher level ontologies (e.g. foundational ontologies)
- There is need for small ontologies to address each design issue separately
- The ontology should be accompanied with explicit documentation of its design rationales and best reengineering practices
- Therefore, in analogy to software design patterns there are **ontology design patterns**

## 0.1.2 Ontology design pattern catalogues

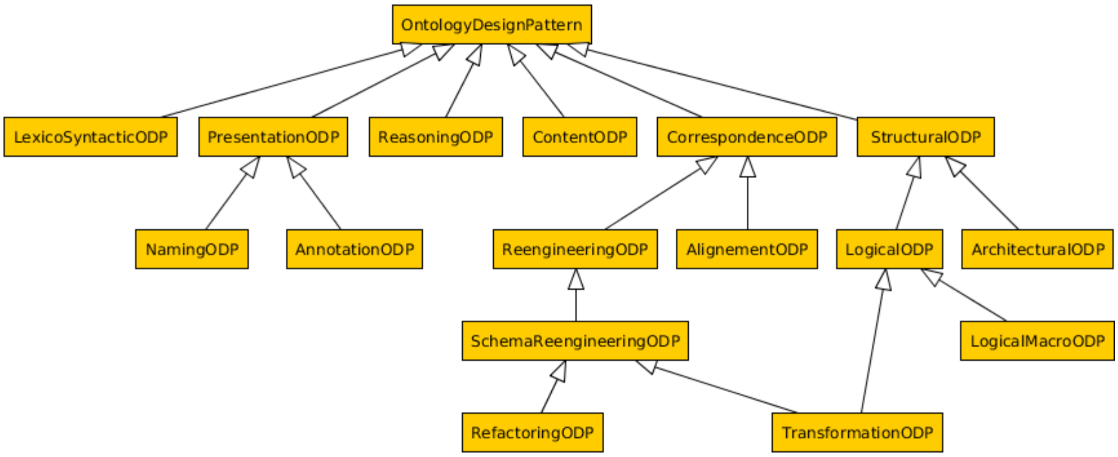
### Overview of ontology design pattern catalogues

Most known public ODP catalogues are :

- **ODPs from W3C Semantic Web Best Practices and Deployment Working Group** – contains 4 pattens i.e. n-ary relations, classes as property values, value partitions/sets, simple part-whole relations. (`http://www.w3.org/2001/sw/BestPractices`)
- **ODPs from the University of Manchester** – contains 17 patterns devided into groups *extension ODPs* (solutions to by-bas the limitations of OWL such as n-ary relations), *good practice ODPs* (making robust and cleaner design e.g. value partitions), *domain modelling ODPs* (solutions for concrete modeling problems in biology). (`http://www.gong.manchester.ac.uk/odp/html`)
- **ODPs from ontologydesignpatterns.org** – contains over 100 patterns categorized into 6 groups of patterns hosted on Semantic Web portal dedicated to ODPs providing review process for creation of certified patterns. (`http://ontologydesignpatterns.org`)

## 0.1.3 Types of ontology design patterns

### Classification of ODPs (1)



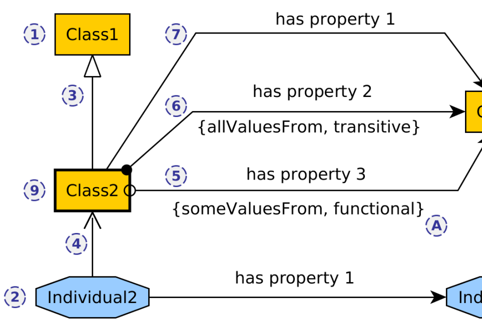

Classification of ODPs according to ontologydesignpatterns.org portal (ODP portal)

### Classification of ODPs (2)

- **Content ODP** – represents domain-specific pattern
- **Structural ODP** – is structure to solve architectural and logical issues of OWL ontologies
- **Correspondence ODP** – is used for reengineering and mappings
- **Reasoning ODP** – is typical reasoning procedure
- **Presentation ODP** – relates to usability of ontology from user perspective
- **Lexico-Syntactic ODP** – is linguistic structure/schema that allow to generalize and extract some conclusions about the meaning they express

## 0.1.4 Selected ontology design patterns

### Diagramming conventions within selected ODPs




The figure shows diagramming conventions that will be used in subsequent slides for selected ODPs. *Squares* (1,9) – classes; *octagons* (2) – individuals; *closed hollow arrows* (3) – `rdfs:subClassOf` or `rdfs:subPropertyOf` relations; *opened arrows* (4) – `rdf:type` relations; *semi-closed solid arrows* (5,6,7) – having origin of the arrow decorated by: a) *hollow blob* (5) – existential restrictions of the class at the origin of the arrow, b) *solid blob* (6) – universal restriction of the class at the origin of the arrow, c) *no decoration* (7) – domain and range axioms of the property if used with classes, facts if used with individuals; *solid/dashed arrows* (3,4,5,6,7)/(8) – asserted/inferred axioms, respectively; *normal/bold edges of a square* (1)/(9) – the classes represented by the square are defined partially/completely with restrictions and other relevant axioms defined in the figure, respectively; *texts within* {} *brackets* (A) – additional information about restriction or property represented by the arrow, someValuesFrom/allValuesFrom information is already represented with arrow having solid/hollow decoration of the origin, thus may be omitted.

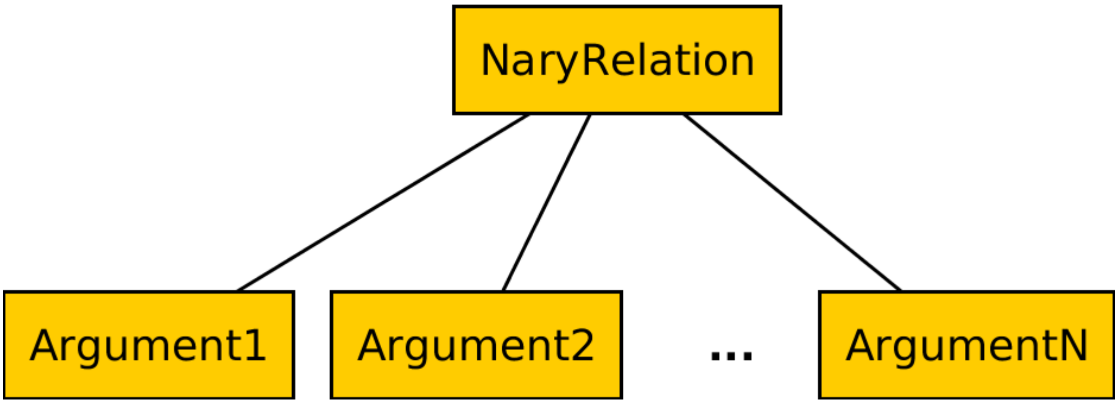



Figure 0.1: Pattern 1

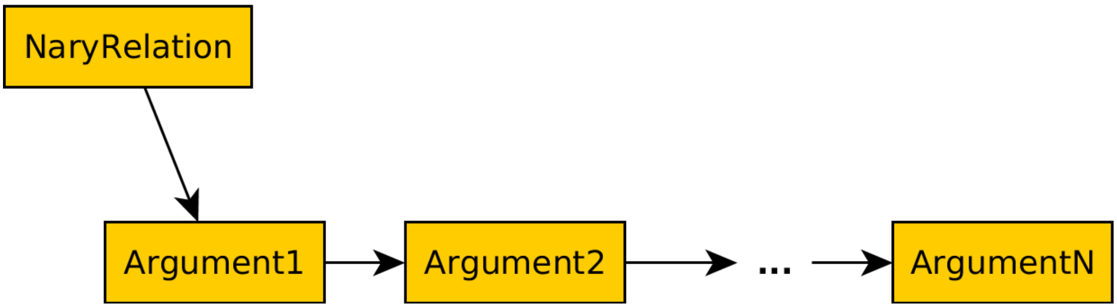



Figure 0.2: Pattern 2

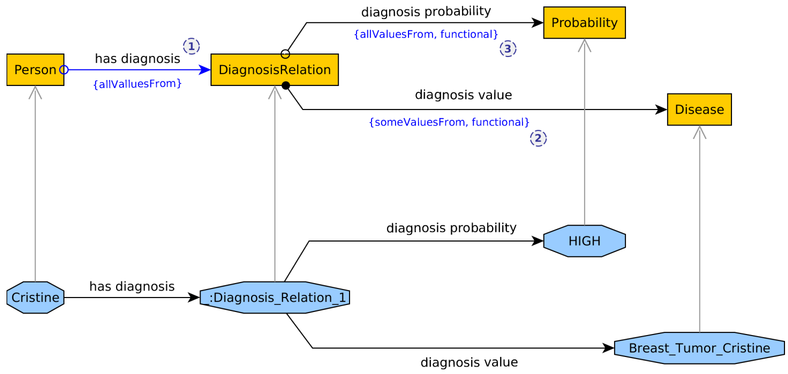
### N-ary relations – general patterns

Logical ODPs that solves issue of representing n-ary relations in OWL which has native support only for binary relations.

- most common representation of n-ary relations
- possible restrictions per argument (e.g. type for each argument, cardinality of each argument type)
- possibility to define required/optional arguments of the relation
- determining owner of relations by direction of the object properties

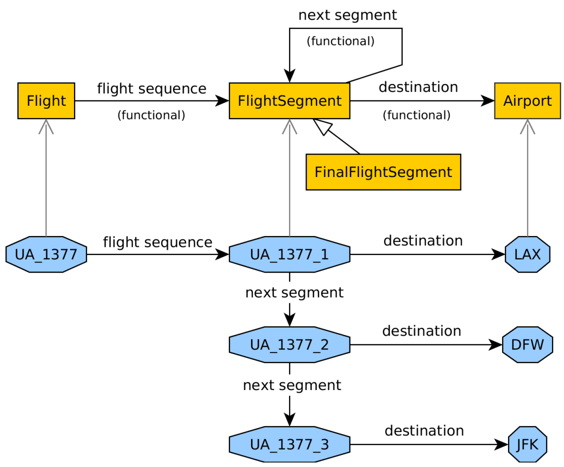
- ordering of dynamic number of arguments
- argument types are content specific instead of generic ones as it is in case of generic list ODP

### N-ary relations – pattern 1 example




The figure demonstrate use of *nary-relations ODP pattern 1* for representation of ternary relation – medical diagnosis of disease (expressed by class `DiagnosisRelation`). The ownership of the relation is captured by direction of `has_diagnosis` (1). Each diagnosis is obliged to have some diagnosis value (2), while the diagnosis probability is understood as additional parameter of the relation which is only obliged to have correct type (3) if the value exists. Similarly to diagnosis probability, `Person` is not obliged to have some diagnosis (1).

## N-ary relations – pattern 2 example




The figure demonstrate use of *nary-relations ODP pattern 2* for representation of n-ary relation that have dynamic number of parameters where ordering of the parameters matters. It represents flight as ordered sequence of flight segments that point to airport destinations.

## Value partitions and value sets – general patterns

Value partition and value set ODP is able to represent a *feature* of some entity (sometimes also referred as “quality”, “attribute”, “characteristic”, or “modifier” of the entity). There are two ways basic ways to represent the feature:

- *values as sets of individuals*
- no possibility of further sub-partitioning

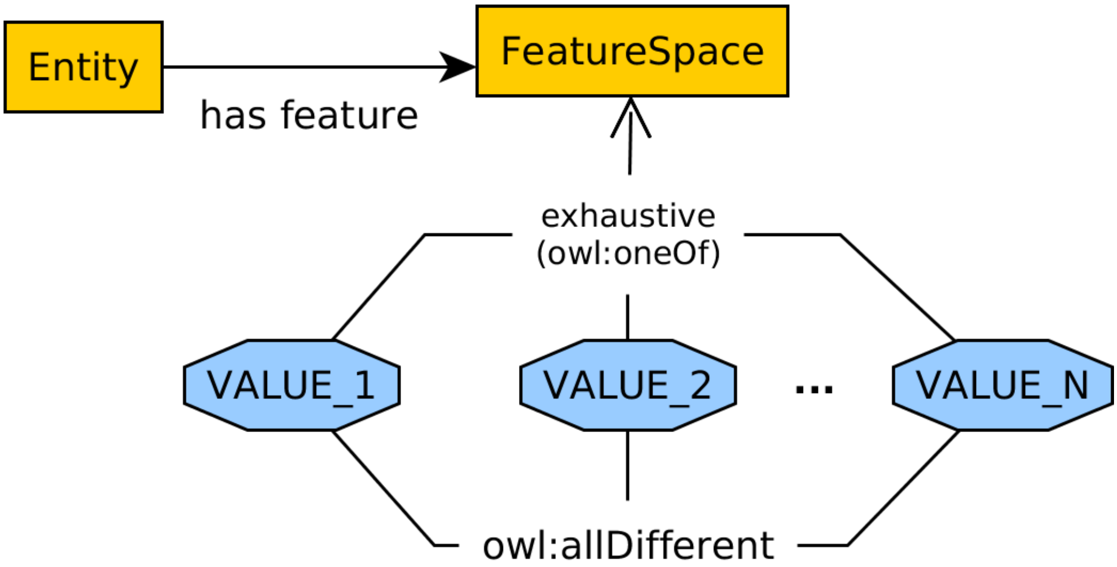
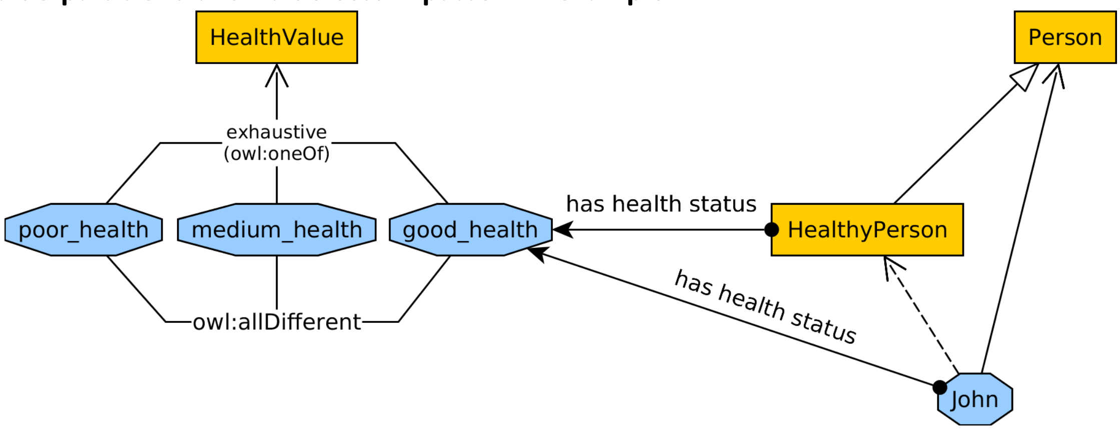



Figure 0.3: Pattern 1

- no alternative partitioning of same feature space
- straightforward with database matching

- *values as subclasses partitioning a "feature"*
- possible sub-partitioning and alternative partitioning
- some people consider it less intuitive

**Value partitions and value sets – pattern 1 example**




The figure represents feature "health status of a person" by using feature space `HealthValue` as set of concrete values `poor_health`, `medium_health`, `good_health`.

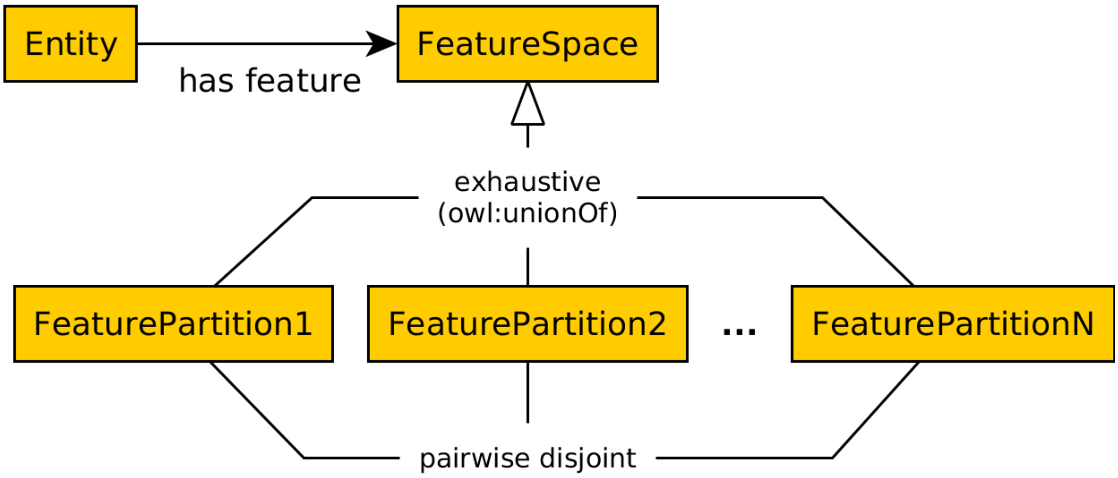
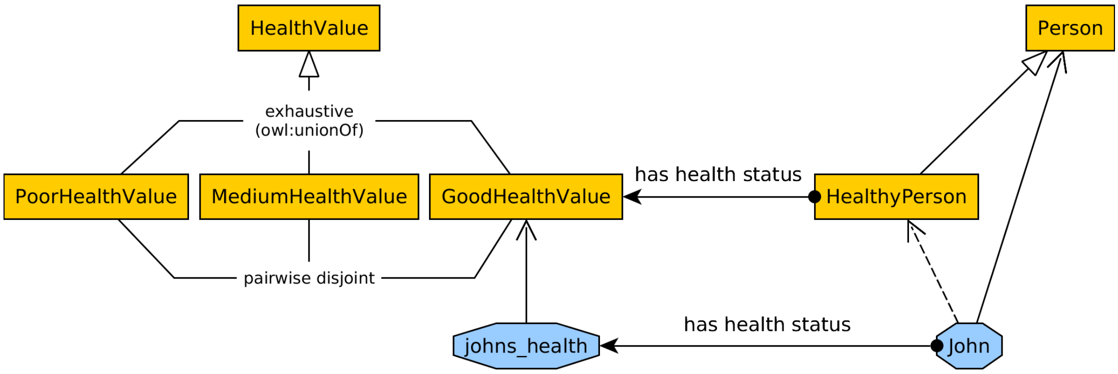


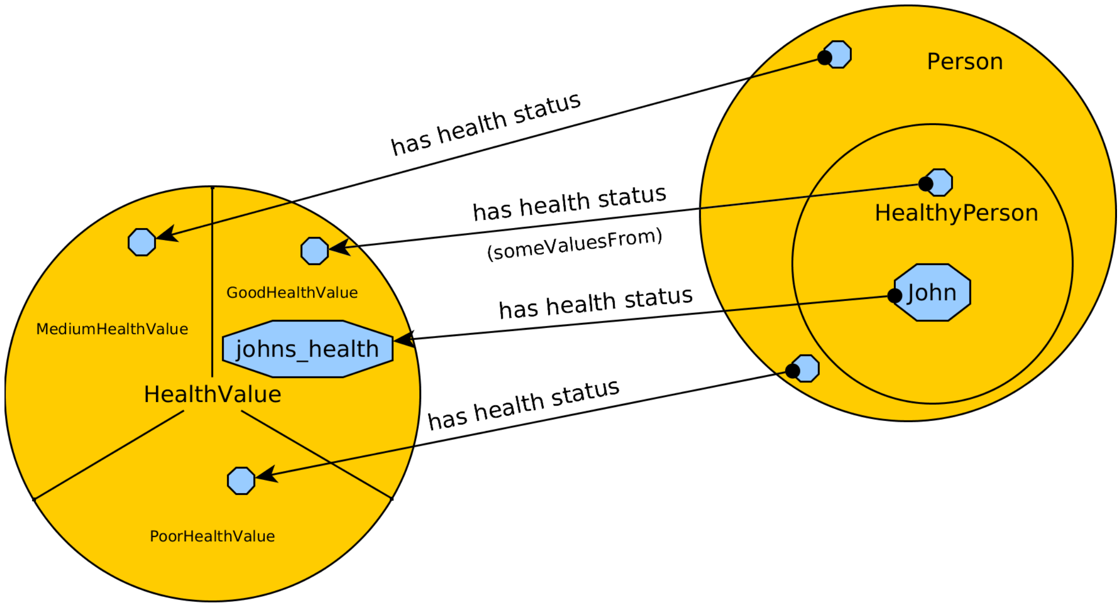

Figure 0.4: Pattern 2

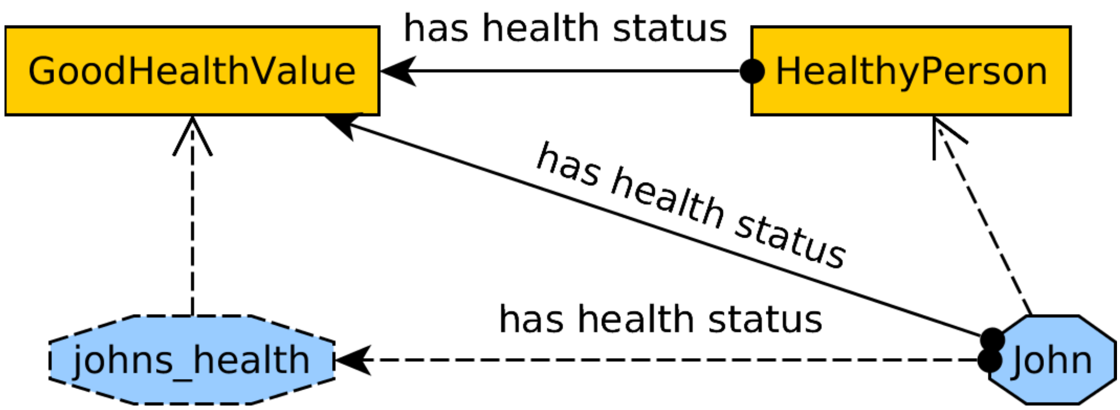
## Value partitions and value sets – pattern 2 example




The figure represents feature "health status of a person" by partitioning feature space `HealthValue` into sub-partitions `PoorHealthValue`, `MediumHealthValue`, `GoodHealthValue`.

## Value partitions and value sets – pattern 2 example

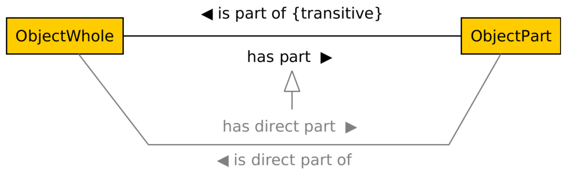



The figure on the left depict previous example in adapted Venn diagram as an alternative diagrammatic format to show partitioning more explicitly. The right part of the figure shows alternative representation of John's health status which is not expressed explicitly but inferred from other axioms

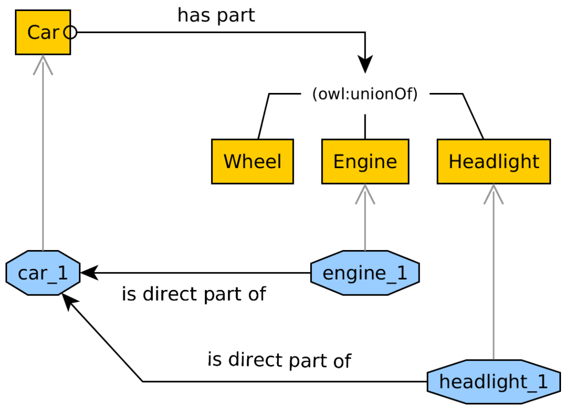
### Part-whole relations – general pattern

Part-whole relation ODP provide us way to represent objects called wholes and their parts.



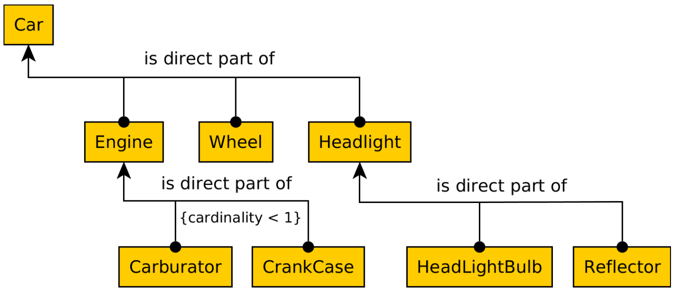

The figure depicts general schema for part-whole relations.

## Part-whole relations – inventory of parts example



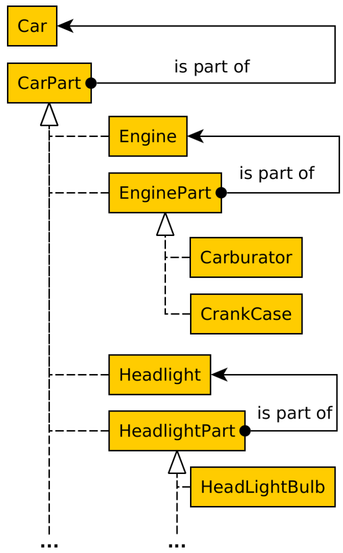

The figure shows how to represent inventory of parts (i.e. parts of concrete objects).

## Part-whole relations – hierarchy of parts example



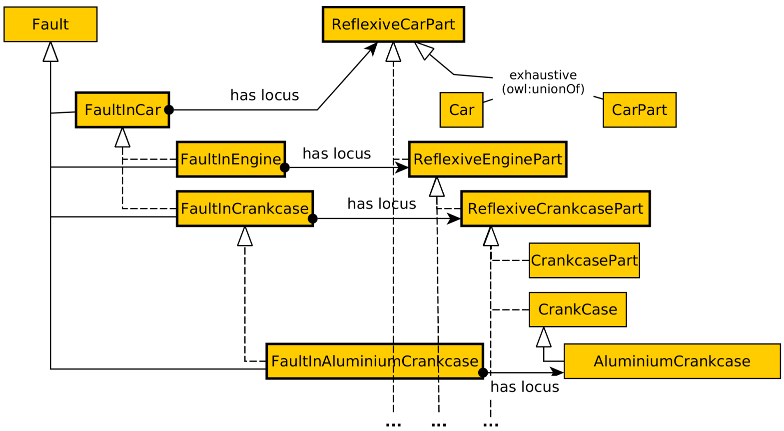

The figure shows how to represent hierarchy of hypothetical parts of wholes).

## Part-whole relations – classes for parts example




The figure shows how to represent classes for parts, so the correct hierarchy of parts is inferred.

## Part-whole relations – faults in parts example




The figure shows how to faults in parts using `has_locus` property.