

# Pattern-based ontology design

Maciej Janik

credit:

**Aldo Gangemi, Valentina Presutti**

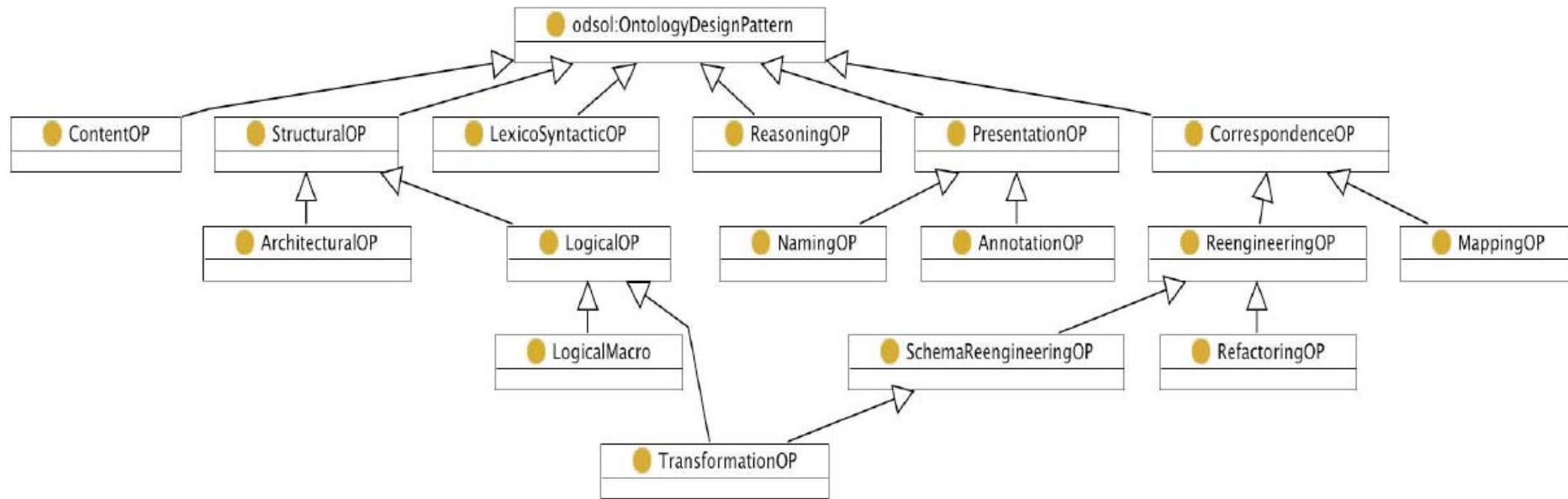


Semantic Web  
2009-07-03

- OWL gives us **logical language constructs**, but **does not give us any guidelines** on how to use them in order to solve our tasks.
  - ◆ E.g. modeling something as an individual, a class, or an object property can be quite arbitrary
- ... OWL is not enough for building a good ontology and we cannot ask all web users either to learn logic, or to study ontology design
- Reusable solutions are described as **Ontology Design Patterns**, which help reducing arbitrariness without asking for sophisticated skills ...
- ... provided that tools are built for any user.

An ontology design pattern is a reusable solution to a recurrent modeling problem

- Pattern-based ontology design is the activity of searching, selecting, and composing different patterns
  - ◆ Logical,
  - ◆ Reasoning,
  - ◆ Architectural,
  - ◆ Naming,
  - ◆ Correspondence,
  - ◆ Reengineering,
  - ◆ Content.
- Common framework to understand modeling choices (the "solution space") with respect to task- and domain-oriented requirements (the "problem space")
- **<http://www.ontologydesignpatterns.org>**



- There are
  - ◆ Ontology Design Patterns
  - ◆ not really Ontology Patterns (indifferent)
  - ◆ Ontology Design Anti-Patterns (AntiOPs)

- Naming OPs are conventions on how to create names for namespaces, files, and ontology elements in general (classes, properties, etc.).
- Naming OPs are good practices that boost ontology readability and understanding by humans, by supporting homogeneity in naming procedures.
- Example
  - ◆ Class names – singular, capital letter
  - ◆ Human readable names – not only numeric values
  - ◆ Use of suffixes or prefixes

- Annotation OPs provide annotation properties or annotation property schemas that are meant to improve the understandability of ontologies and their elements
- Example:
  - ◆ Use of RDF Schema labels and comments (crucial for manual selection and evaluation)
  - ◆ Each class and property should be annotated with meaningful labels (also: different language)
  - ◆ Each ontology and ontology element should be annotated with the rationale they are based on using `rdfs:comment`

- Correspondence OPs include
  - ◆ Reengineering OP,
  - ◆ Mapping OP.
- *Reengineering OPs* provide designers with solutions to the problem of **transforming** a conceptual model, which can even be a non-ontological resource, into a new ontology.
- *Mapping OPs* are patterns for **creating semantic associations** between two existing ontologies.

- Reengineering OPs are transformation rules applied in order to create a new ontology (target model) starting from elements of a source model
- The target model is an ontology, while the source model can be either an ontology, or a non-ontological resource
  - ◆ e.g., a thesaurus concept, a data model pattern, a UML model, a linguistic structure, etc.
- Two types:
  - ◆ Schema reengineering OPs are rules for transforming a non-OWL DL metamodel into an OWL DL ontology
  - ◆ Refactoring OPs provide designers with rules for transforming, i.e. “refactoring”, an existing OWL DL “source” ontology into a new OWL DL “target” ontology
    - e.g. a guideline to reengineer a piece of an OWL ontology in presence of a requirement change, as when moving from individuals to classes, or from object properties to classes.

- Example:
  - ◆ kos2skosABox

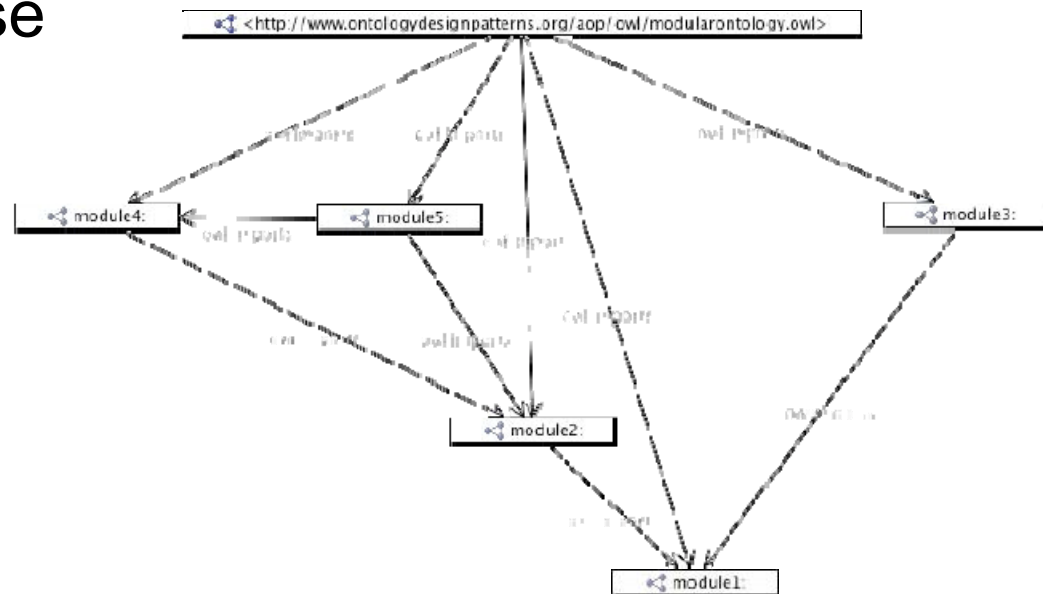
KOS  $\mapsto$  skos:ConceptSchema (2.1)

Descriptor  $\mapsto$  skos:Concept (2.2)

Broader Term  $\mapsto$  skos:broader (2.3)

Related Term  $\mapsto$  skos:related (2.4)

- Architectural OPs affect the overall shape of the ontology: their aim is to constrain ‘how the ontology should look like’
- Architectural OPs emerged as design choices motivated by specific needs
  - ◆ e.g., computational complexity constraints.
- They are useful as reference documentation for those initially approaching the design of an ontology



- Architectural OPs can be of two types: internal APs and external APs
  - ◆ Internal APs are defined in terms of collections of Logical OPs that have to be exclusively employed when designing an ontology
    - e.g., an OWL species, or the varieties of description logics:  
<http://www.cs.man.ac.uk/~ezolin/dl/>
  - ◆ External APs are defined in terms of meta-level constructs
    - e.g., the modular architecture consists of an ontology network, where the involved ontologies play the role of modules. The modules are connected by the import operation.

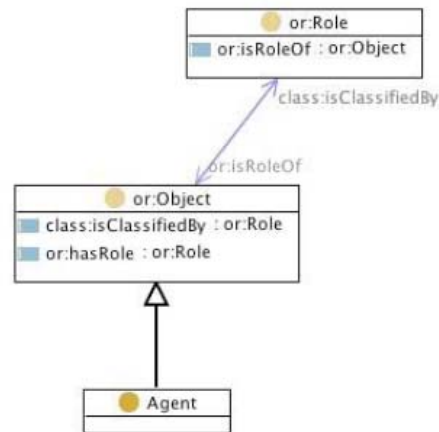
- A Logical OP is a formal expression, whose only parts are expressions from a logical vocabulary e.g., OWL DL, that solves a problem of expressivity
- Logical OPs are independent from a specific domain of interest – they are content-independent
- Logical OPs depend on the expressivity of the logical formalism that is used for representation
  - ◆ They help to solve design problems where the primitives of the representation language do not directly support certain logical constructs
- They can be of two types: *logical macros*, and *transformation patterns*

- Logical macros provide a shortcut to model a recurrent intuitive logical expression
  
- Example:
  - ◆ the macro:  $\forall R.C$
  - ◆ colloquially means “every R must be a C”
  - ◆ formally:  $\exists R. \top \sqcap R.C$
  - ◆ in OWL:
    - the combination of an `owl:allValuesFrom` restriction with an `owl:someValuesFrom` restriction.

- CPs encode conceptual, rather than logical design patterns.
  - ◆ Logical OPs solve design problems independently of a particular conceptualization
  - ◆ CPs are patterns for solving design problems for the domain classes and properties that populate an ontology, therefore they address content problems

- CPs are instantiations of Logical OPs (or of compositions of Logical OPs)
  - ◆ they have an explicit non-logical vocabulary for a specific domain of interest - **they are content-dependent**
  
- Modeling problems solved by CPs have two components: *domain* and *requirements*.
  - ◆ one domain - many requirements
  - ◆ same requirement - different domains
  - ◆ typical way of capturing requirements is by means of competency questions

- (Small) ontology morphing
  - ◆ “being a part of something at some time”
- Downward subsumption of at least one element
  - ◆ “being a component of a system at some time”
- Mostly graphs of classes and properties that are self-connected through axioms (subClassOf, equivalentClass, domain, range, disjointFrom)
  - ◆ ObjectProperty(component domain(System))
- Usually there is an underlying n-ary relation (sometimes polymorphic)
  - ▶  $\forall c((\varphi(c) \wedge \exists p(\Psi(p) \wedge \rho(c,p))) \rightarrow \chi(c))$
  - ▶ SubClassOf
  - ▶ ((intersectionOf
  - ▶ owl:Class: $\varphi$
  - ▶ (restriction(owl:ObjectProperty: $\rho$  someValuesFrom(owl:Class: $\Psi$ ))))
  - ▶ owl:Class: $\chi$ )



## Elements

The **AgentRole** Content OP locally defines the following ontology elements:

### Agent (owl:Class)

Any agentive **Object**, either physical, or social.

[Agent page](#)

## Reviews about AgentRole

The are no reviews.

Go back to the [List of Content OP proposals](#)

The **time indexed person role** CP allows to represent temporariness of roles played by persons. It can be generalized for including objects or, alternatively the **n-ary classification** CP can be specialized in order to obtain the same expressivity.

The elements of this Content OP are added with the elements of its components and/or the elements of the Content OPs it is a specialization of.

## AgentRole

**Submitted by** [ValentinaPresutti](#)

**Name** agent role

**Also Known As**

**Intent** To represent agents and the roles they play.

**Domains** [Management, Organization, Scheduling](#)

**Competency** which agent does play this role?, what is the role that played by that agent?

**Questions**

**Reusable OWL** <http://www.ontologydesignpatterns.org/cp/owl/agentrole.owl>

**Building Block**

**Consequences** This CP allows designers to make assertions on roles played by agents without involving the agents that play that roles, and vice versa. It does not allow to express temporariness of roles.

**Scenarios** She greeted us all in her various roles of mother, friend, and daughter.

**Known Uses**

**Web**

**References**

**Other**

**References**

**Examples (OWL files)** <http://www.ontologydesignpatterns.org/cp/examples/agentrole/ex1.owl>

**Extracted From** <http://www.loa-cnr.it/ontologies/DUL.owl>

**Reengineered**

**From**

**Has**

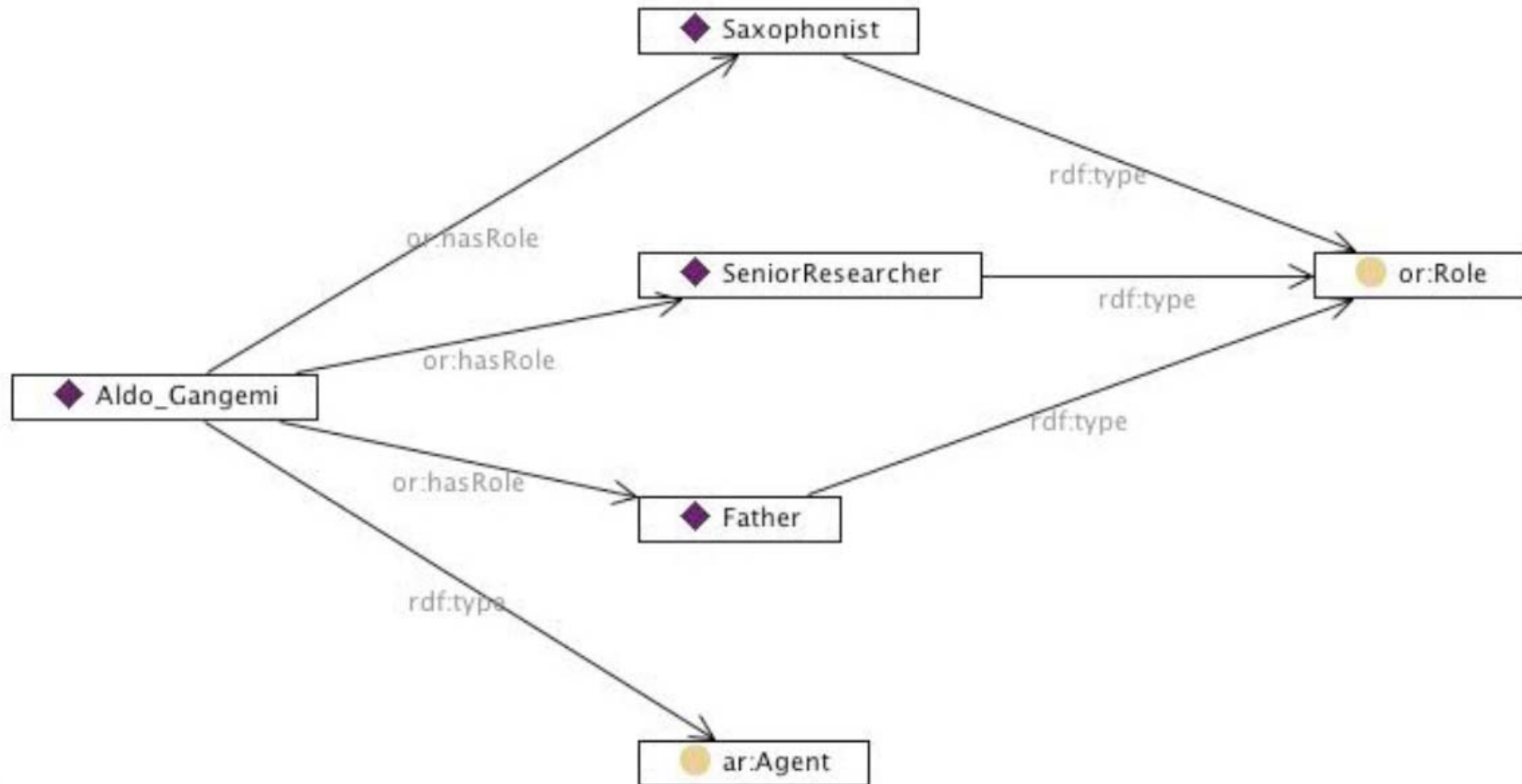
**Components**

**Specialization** [Submissions:Objectrole](#)

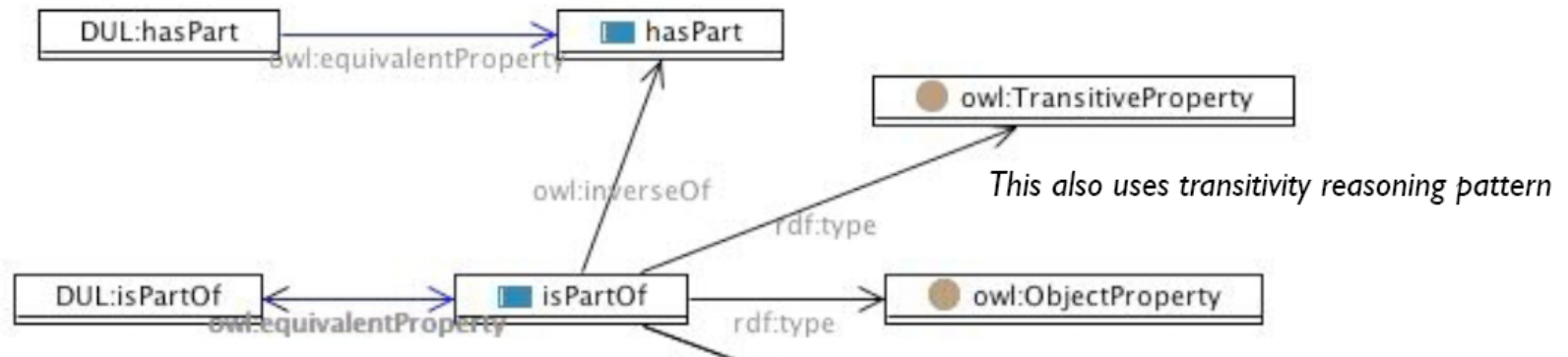
**Of**

**Related CPs**

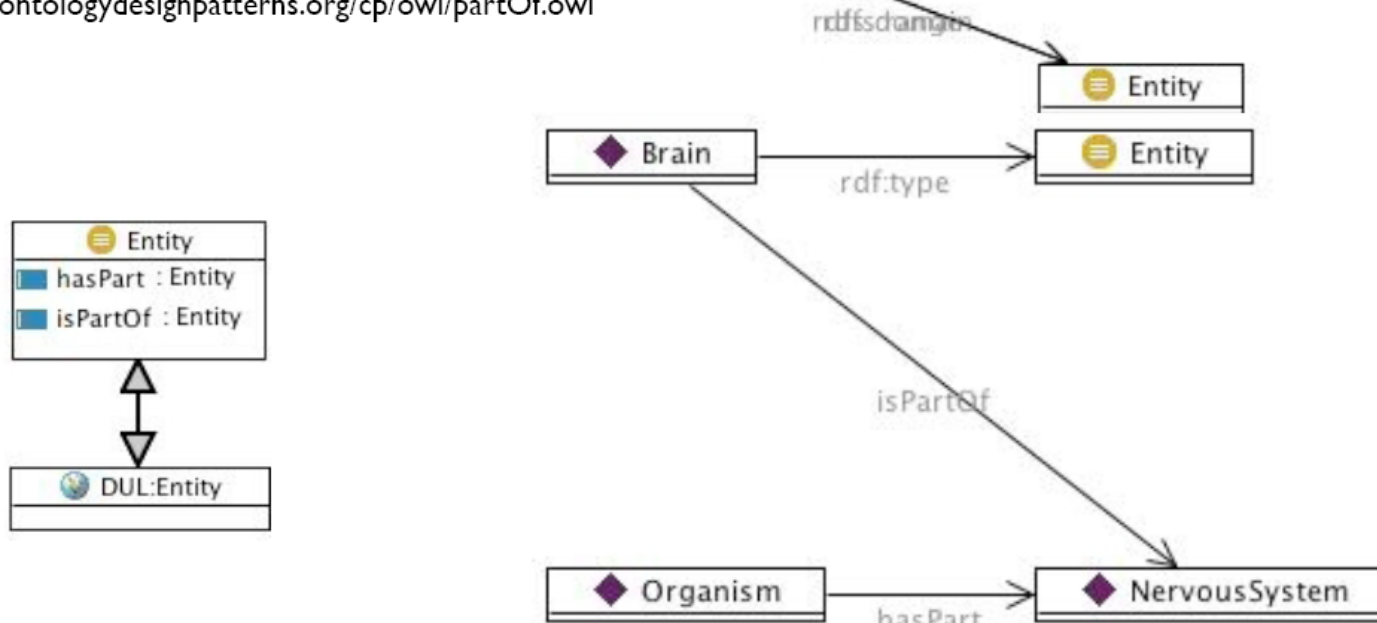
# Example: Agent role - instantiation



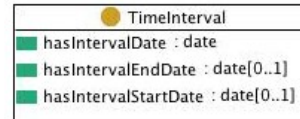
# Example: Part pattern



Cf. <http://www.ontologydesignpatterns.org/cp/owl/partOf.owl>



## Diagram




## Elements

The *TimeInterval* Content OP locally defines the following ontology elements:

 **Time Interval** (owl:Class)


Any region in a dimensional space that represents time.

[TimeInterval page](#)

 **has interval date** (owl:DatatypeProperty)


A datatype property that encodes values from `xsd:date` for a time interval; a same time interval can have more than one `xsd:date` value: begin date, end date, date at which the interval holds, as well as dates expressed in different formats: `xsd:gYear`, `xsd:dateTime`, etc.

[hasIntervalDate page](#)

 **has interval start date** (owl:DatatypeProperty)

The start date of a [time interval](#).

[hasIntervalStartDate page](#)

 **has interval end date** (owl:DatatypeProperty)

The end date of a [time interval](#).

[hasIntervalEndDate page](#)

This Content OP can be composed with other Content OPs when temporal aspects need to be represented.

## TimeInterval

**Submitted by:** [ValentinaPresutti](#)

**Name:** time interval

**Also Known As:**

**Intent:** To represent time intervals.

**Domains:** [Time](#)

**Competency Questions:** What is the end time of this interval?, What is the starting time of this interval?, What is the date of this time interval?

**Reusable OWL Building Block:** <http://www.ontologydesignpatterns.org/cp/owl/timeinterval.owl> (6)

**Consequences:** The dates of the time interval are not part of the domain of discourse, they are datatype values. If there is the need of reasoning about dates this Content OP should be used in composition with the [region](#) Content OP.

**Scenarios:** The time interval "January 2008" starts at 2008-01-01 and ends at and ends at 2008-01-31.

**Known Uses:**

**Web References:**

**Other References:**

**Examples (OWL files):** <http://www.ontologydesignpatterns.org/cp/examples/timeinterval/january2008.owl>

**Extracted From:**

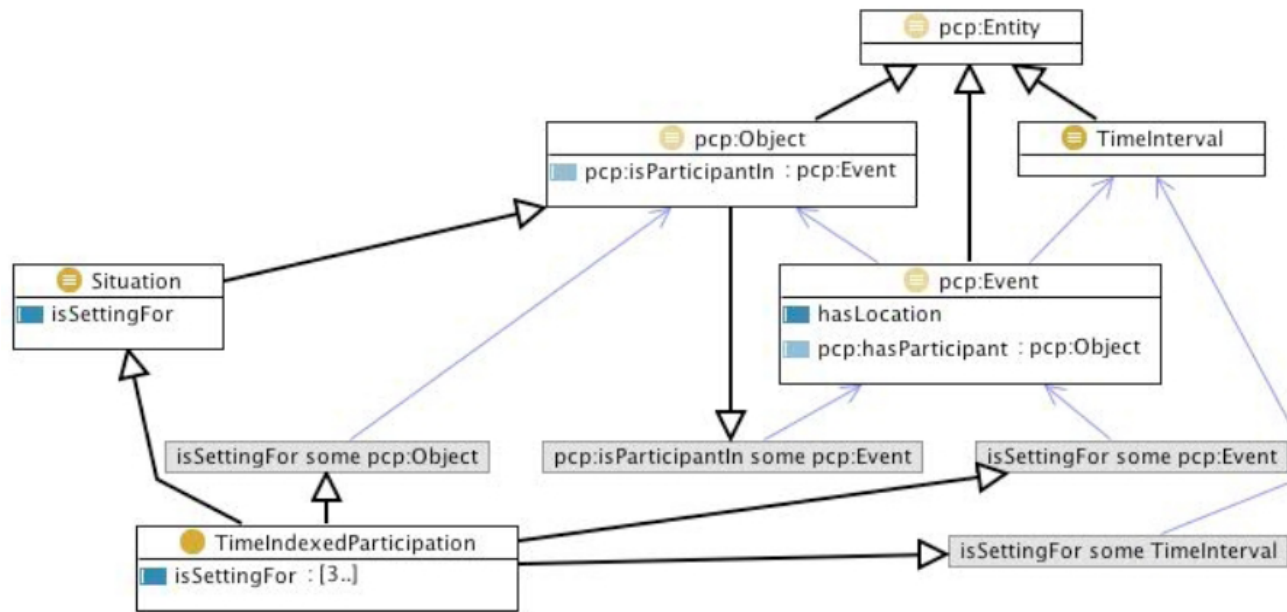
**Reengineered From:**

**Has Components:**

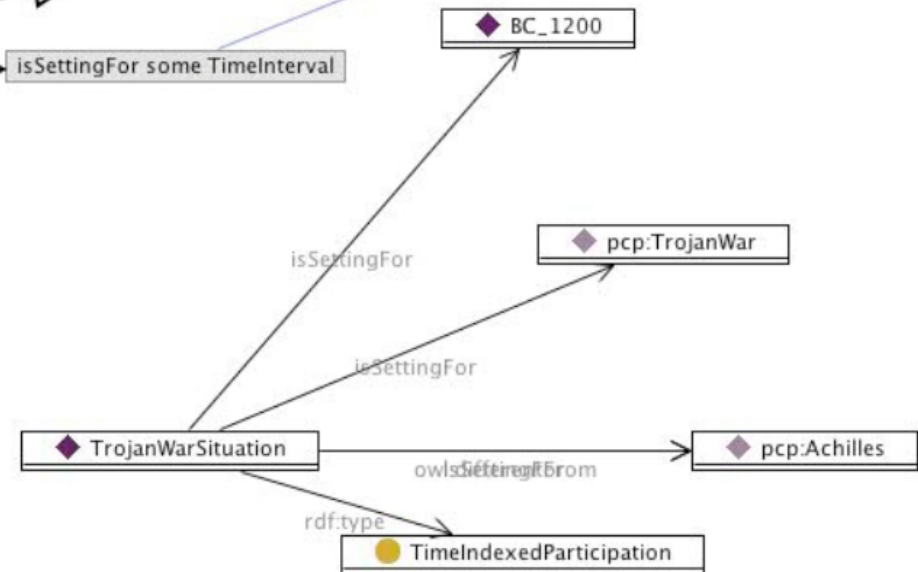
**Specialization Of:**

**Related CPs:**

# Example: Time-indexed participation

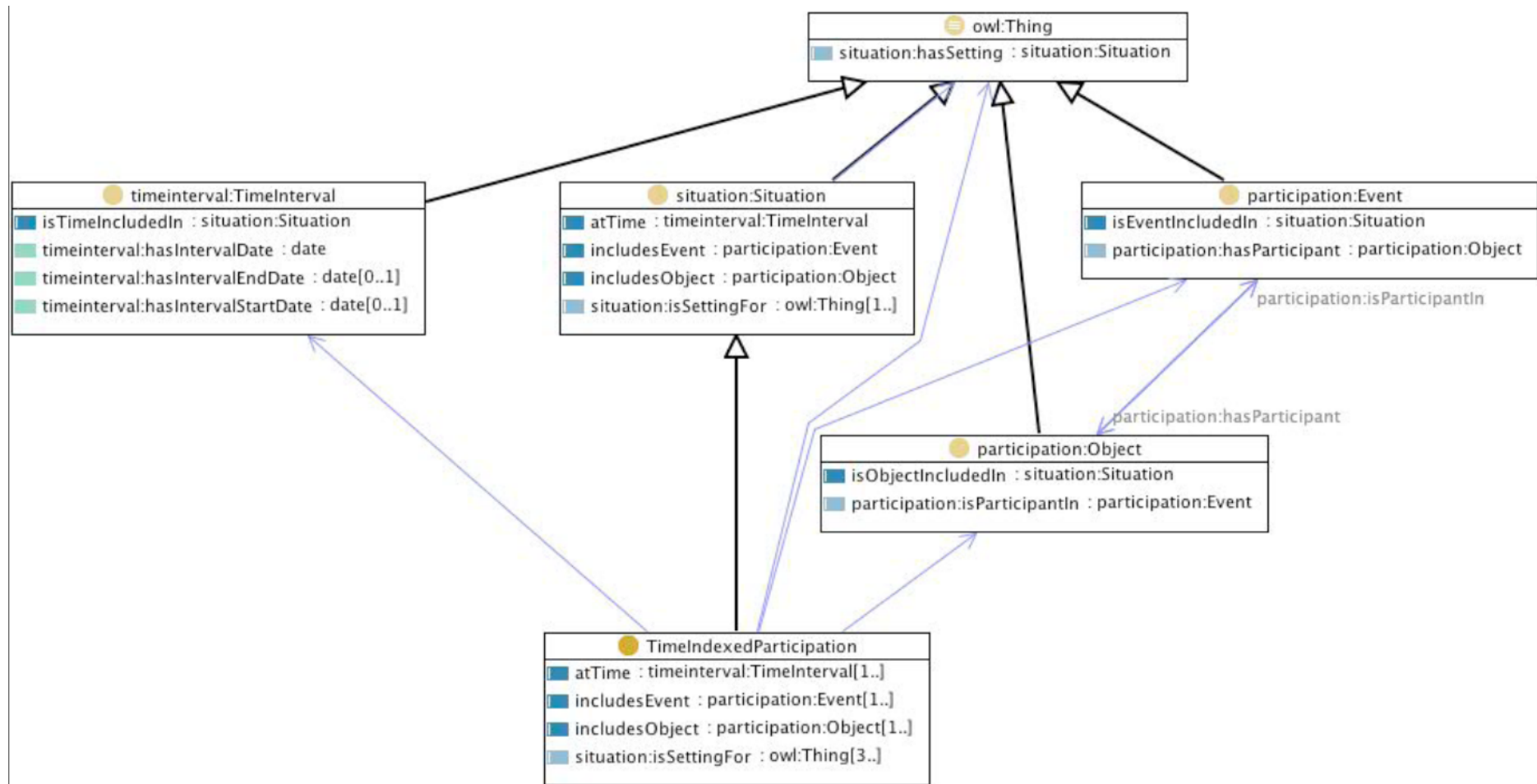


*This also uses N-ary logical pattern*



- Matching pattern to your situation / ontology
  - ◆ Precise or redundant matching
  - ◆ Broader or narrower matching
  - ◆ Partial matching
  
- Content Pattern creation and reuse for specific ontology relies on a set of operations:
  - ◆ import
  - ◆ specialization
  - ◆ generalization (converse of specialization)
  - ◆ composition
  - ◆ expansion
  - ◆ clone

# Example reuse of patterns: composition



- Content ontology design patterns (CPs) come from the experience of ontology engineers in modeling foundational, core, or domain ontologies
- There are four ways of creating CPs, which can be summarized as follows:
  - ◆ Reengineering from patterns expressed in other data models
    - Data model patterns, Lexical Frames, Workflow patterns, Knowledge discovery patterns, etc.
  - ◆ Specialization/Generalization/Composition of other CPs
  - ◆ Extraction from reference ontologies

**Use CPs by** combining **extraction, specialization, generalization, composition,** and **expansion**