Introduction to the Semantic Web

Lecture 10: Ontology Design Patterns

Moritz Blum

Semantic Computing Group, Bielefeld University, Germany

Table of contents

Motivation for Pattern & Template Based Ontology Engineering

Problems in current ontology desin processes

Types of ontologies:

- top-level (upper) ontologies
- mid-level ontologies
- application ontologies

In ontology engineering, we always want to reuse existing ontologies if possible.

Problems in current ontology desin processes

Ways to reuse ontological resources:

- ontologies as wholes
- **syntactic/semantic ontology modules**

Problems in reusing existing ontolgies

Problems:

- ontologies must match both domain and task
- ontologies have a lifecycle: design, implementation, evaluation, fixed, exploited, reused
- size and complexity of existing ontologies is usually too large to allow the reuse
- lack of criteria which guide designers in reusing parts of existing ontologies
- no supporting tools to assist designers

$\rightarrow$ Developing an ontology from scratch is often the fastest way.

Why is OWL not enough?

OWL provided logical language constructs, but no guidelines on how to use them. E.g. modeling something as an individual, a class, or an object property can be quite arbitrary.

Modular Ontologies

To overcome these problems: small/modular ontologies with explicit documentation of design rationals, and best engineering practices. They can be understood as building blocks.

CCO - Common Core Ontologies

CCO in general

- adopts the modular approach: different ontologies are responsible for different levels of granularity or different domains
- eleven ontologies that aim to represent and integrate taxonomies of generic classes and relations across some general domains
- encourages domain ontologies to not only model data, but also, more importantly, the entities in the world that data refers to

Foundation of CCO

CCO is implemented using OWL, which allows one to define

- object properties - hierarchies of classes and relationships
- individuals - instances of classes
- individuals - instances of classes
- data properties - data values
- class expressions and class expression axioms - assert relationships between classes
- assertions between properties - object or data subproperties, reflexivity, symmetry, and transitivity
- assertions between classes and object properties - e.g. domain and range restrictions
- useful annotation for classes, individuals, and relationships

CCO Design Decisions

Minimal Asserted Class Axiom Expressions - explicit connections between classes, other than subclass relationships, are not typically asserted

Minimal Object Properties - the number of relationships (in OWL, object properties) between classes or individuals is kept at a minimum

Upper-Level Semantic Framework

CCO is designed as a mid-level extension of Basic Formal Ontology (BFO).

CCO Overview

1. **Information Entity Ontology**
2. **Agent Ontology**
3. Quality Ontology
4. Event Ontology
5. Artifact Ontology
6. Time Ontology
7. Geospatial Ontology
8. **Units of Measure Ontology**
9. Currency Unit Ontology
10. Extended Relation Ontology
11. Modal Relation Ontology

CCO Information Entity Ontology

This ontology is about modeling (1) the content of some piece of information, and (2) the expressions of that content in some medium.

Allows to connect pieces of data (e.g., the values in a data table) to entities in the real world (e.g., books, documents, severs, databases) and for tracking the provenance of data.

```
INFORMATION CONTENT ENTITY cco:is_about ENTITY .
ENTITY cco:is_subject_of INFORMATION CONTENT ENTITY .
```

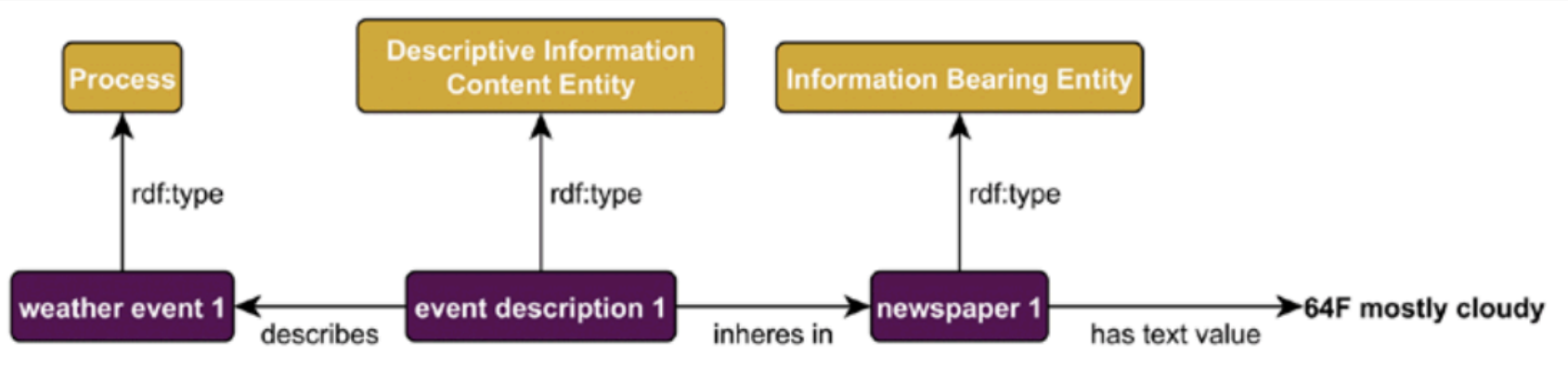
CCO Information Entity Ontology

The *is_about* relation divides into three subproperties, each of which represents a different relation between information content and what that information is about, namely:

- describes - reports and representations
- prescribes - plans and artifact specifications
- designates - names and identifiers

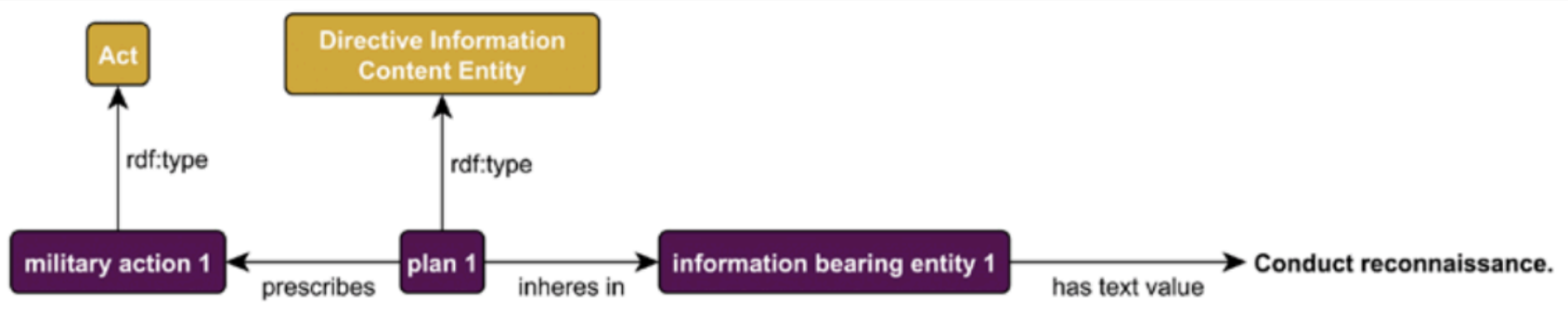
CCO Information Entity Ontology: describes relation

The *"describes"* relation is used for information such as reports and representations (images). An example would be the content of a newspaper describing a weather event:



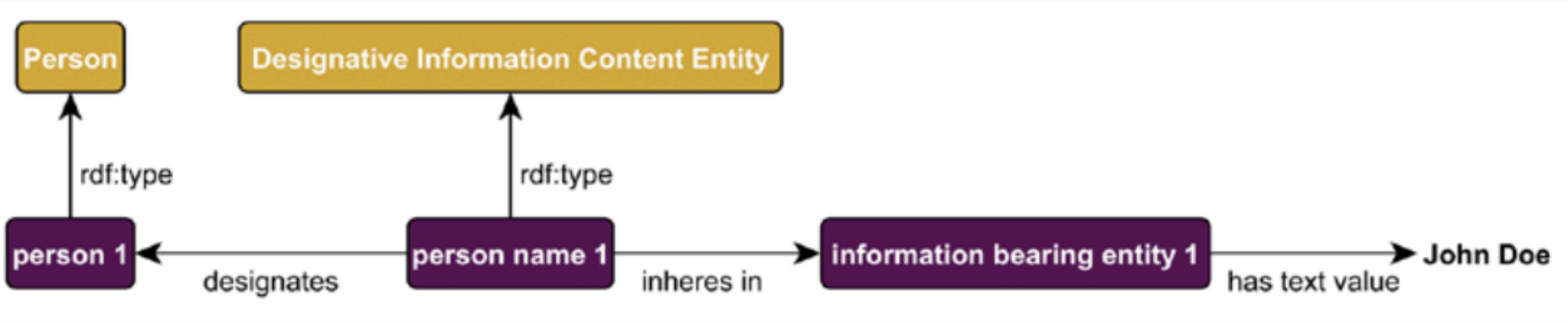
CCO Information Entity Ontology: prescribes relation

The *"prescribes"* relation is used for information such as plans and artifact specifications. An example would be the content of a plan for some military operation:



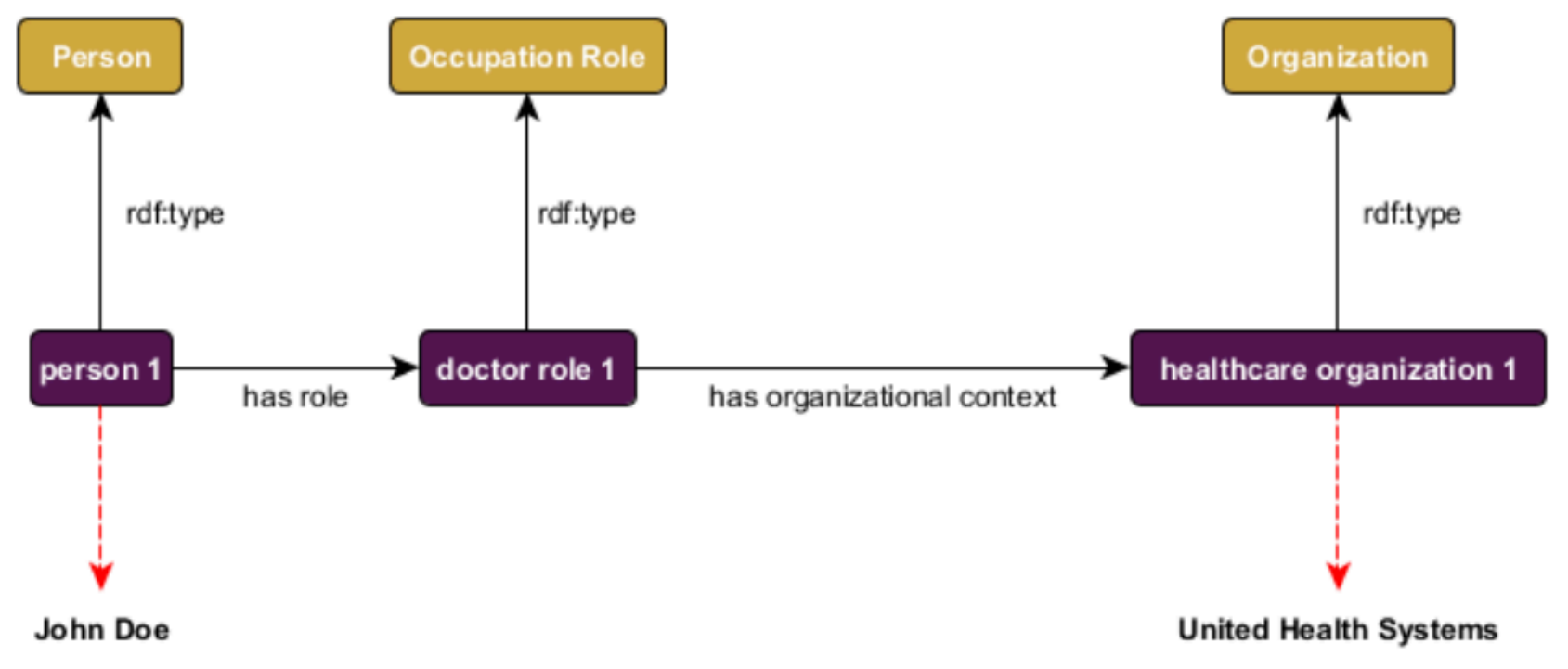
CCO Information Entity Ontology: designates relation

The *"designates"* relation is used for information such as names and other identifiers. An example would be the content of a proper name or an ID number:



CCO Agent Ontology

Can be used to describe agents, their qualities, and the roles they have in various contexts.



CCO Unity of Measure Ontology

Extends from the Information Entity Ontology class MEASUREMENT UNIT.

- several subclasses of measurement unit (e.g. MEASUREMENT UNIT OF AREA, MEASUREMENT UNIT OF ENERGY, and MEASUREMENT UNIT OF LENGTH)
- Specific measurement units are instances of these classes (e.g. Acre, Horsepower, and Kilometer).

Important: a measurement of some phenomenon can be expressed multiple ways, e.g. the length of an object can be measured in inch or in meter.

CCO Unity of Measure Ontology Example

CCO Conclusion

Goal of CCO: to represent entities for a wide array of domains. However, they can not capture the level of detail or specificity needed for users to annotate all the data within their domains. $\rightarrow$ Users can develop domain-level ontologies.

To date, numerous domain-level extensions of the CCO have been developed in, e.g. Aircraft, ethnicity, food, medicine, physics, military, transportation

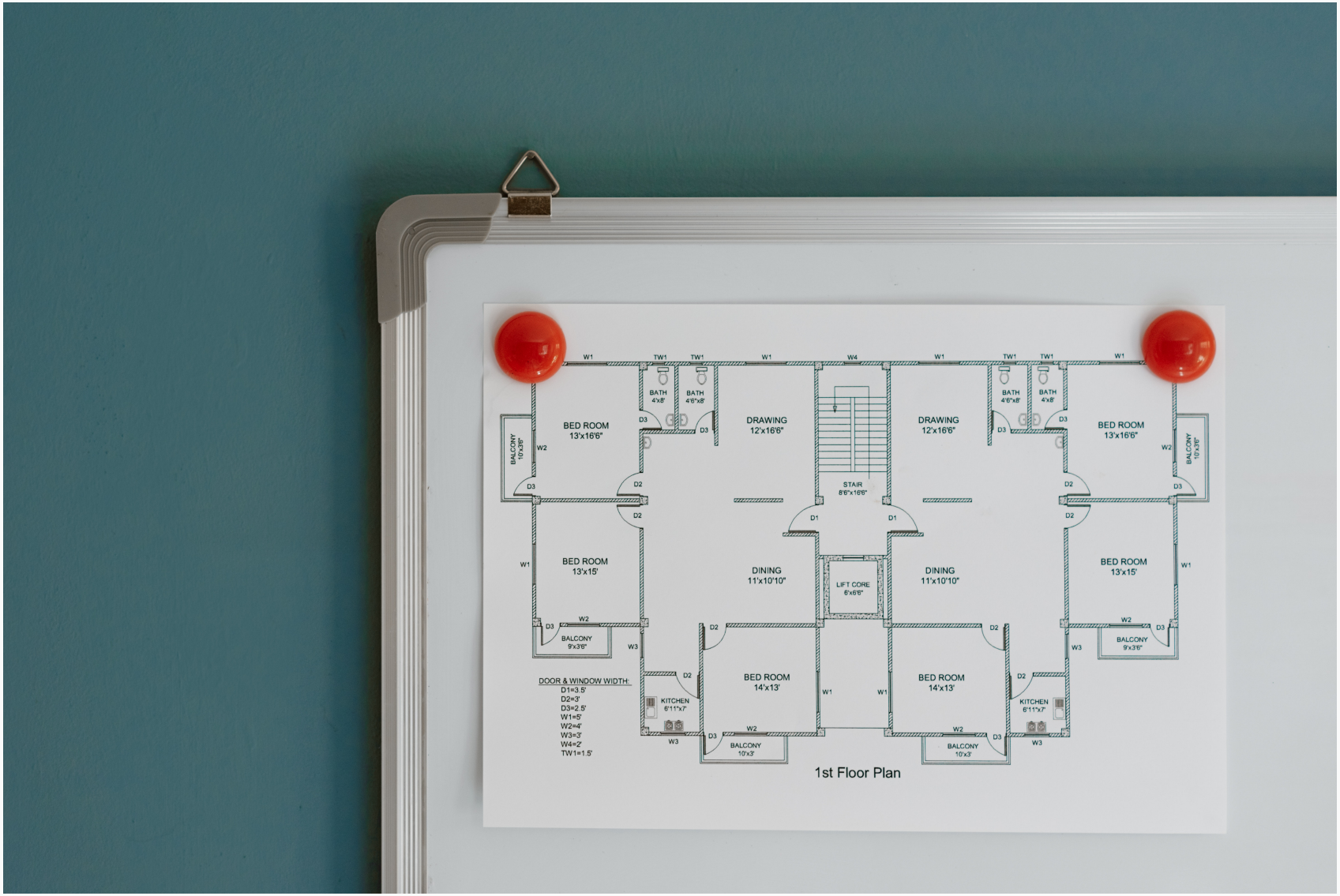
ODP - Ontology Design Patterns

Teaser

Ontologies encode a description of some world, for some purpose.

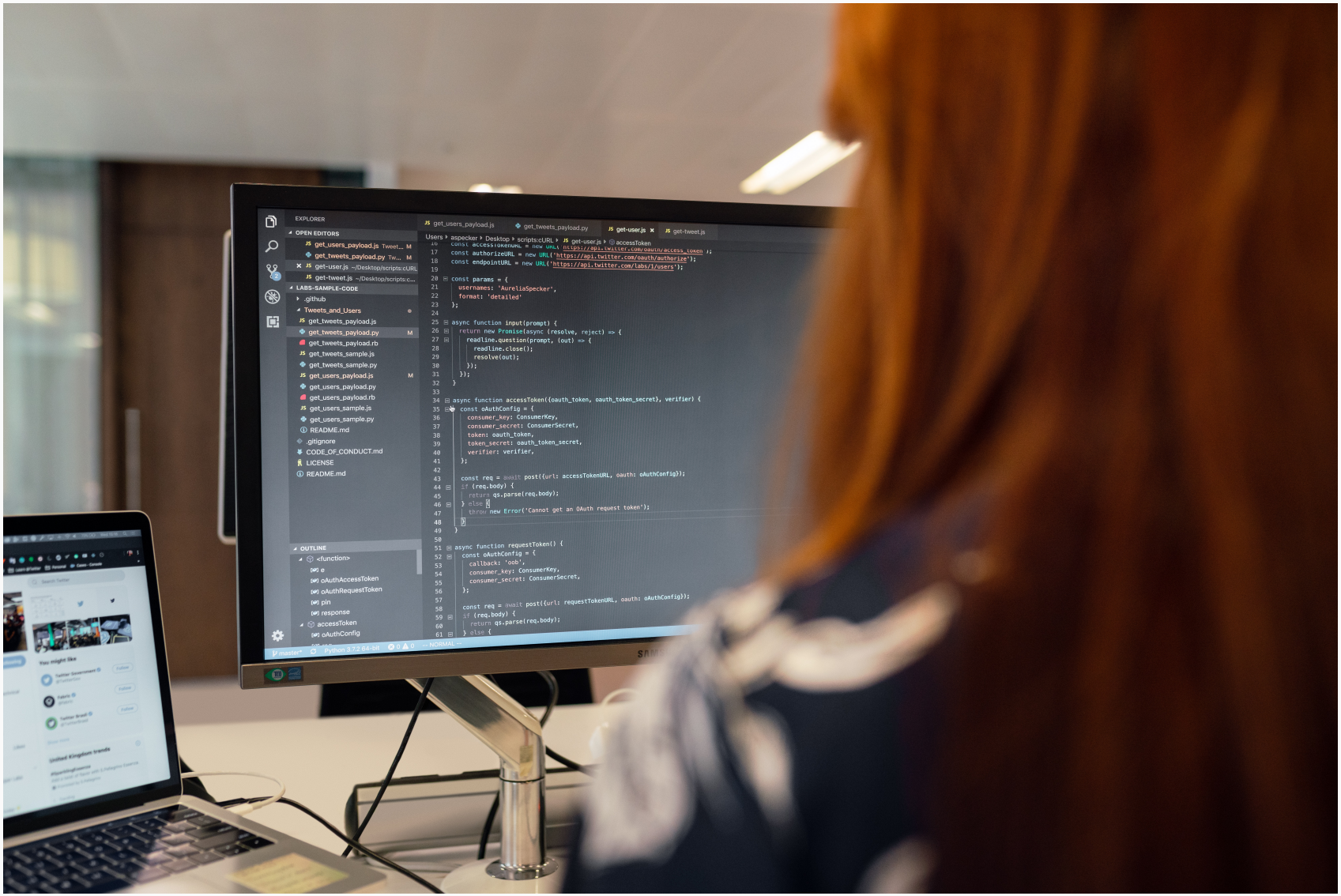
If there exist classes of problems, we can use this in our favor!

Background: Architectural Design Patterns



Architecture

Background: Software Design Patterns



Software

Ontology Design Patterns

ODPs are modularized ontologies = ontology that individual users can easily adapt to their own use-cases, while still preserving relations with other versions of the ontology, that is, keeping it interoperable with other ontologies.

Types of Ontology Design Patterns

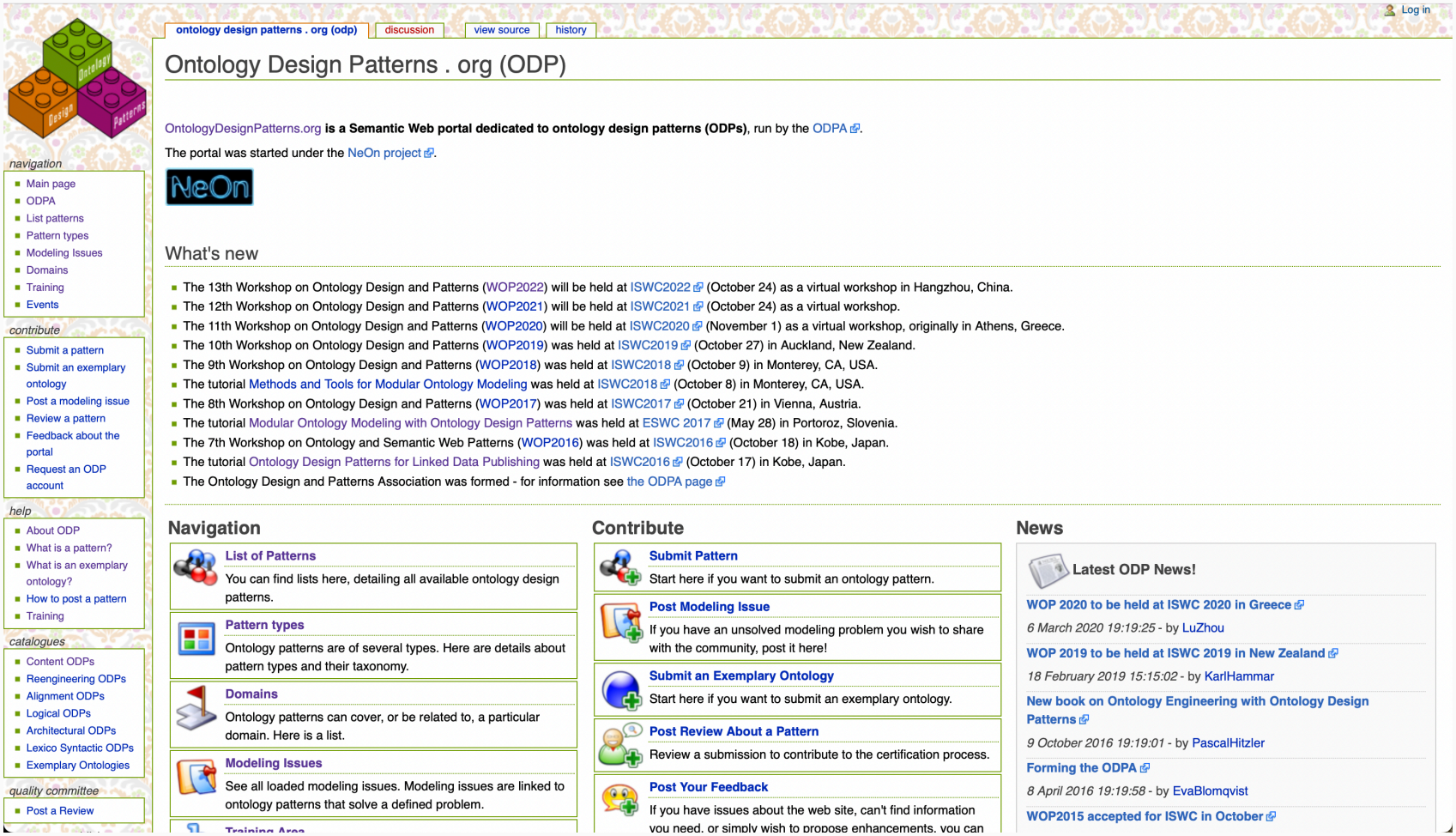
- Content Patterns: domain dependent, language independent
- Logical Patterns: domain independent, related to representation language
- Presentation Patterns: ontology from user perspective, as e.g. naming conventions
- Transformation Patterns: how to transform an ontology in another representation language, e.g. N-ary relation
- more

$\rightarrow$ main building blocks are content patterns and logical patterns

Methodology

Methodology aka how to work with ODPs:
searchable repository ordered by competence questions

Sources for ODPs



http://ontologydesignpatterns.org/

Sources for ODPs

MODL: A Modular Ontology Design Library*

Cogan Shimizu[1], Quinn Hirt[1], and Pascal Hitzler[1,2]

[1] Data Semantics Laboratory, Wright State University, Dayton, OH, USA
[2] Data Semantics Laboratory, Kansas State University, Manhattan, KS, USA

Abstract. Pattern-based, modular ontologies have several beneficial properties that lend themselves to FAIR data practices, especially as it pertains to Interoperability and Reusability. However, developing such ontologies has a high upfront cost, e.g. reusing a pattern is predicated upon being aware of its existence in the first place. Thus, to help overcome these barriers, we have developed MODL: a modular ontology design library. MODL is a curated collection of well-documented ontology design patterns, drawn from a wide variety of interdisciplinary use-cases. In this paper we present MODL as a useful resource for the development of high-quality, modular ontologies, discuss its use, and provide some examples of its contents.

https://github.com/Data-Semantics-Laboratory/modular-ontology-design-library/blob/master

MODL Ontology Design Patterns

Patterns included in the MODL library: Content Patterns and Logical Patterns

Category	**Patterns**
Metapatterns	Explicit Typing Property Reification Stubs
Organization of Data	Aggregation, Bag, Collection Sequence, List Tree
Space, Time, and Movement	Spatiotemporal Extent Spatial Extent Temporal Extent Trajectory Event
Agents and Roles	AgentRole ParticipantRole Name Stub
Description and Details	Quantities and Units Partonymy/Meronymy Provenance Identifier

MODL ODP categories.

Guidelines for ODP documentation in MODL

Guidelines selected based on a community wide survey:

- Schema Diagram
- Example of Pattern Instantiation
- Competency Questions
- Axiomatization
- OWL File
- Pointers to Related Patterns
- Metadata

MODL Schema Diagrams

Simple diagrams are used, as these have shown to work best to get a first understanding of the ODP.

Diagrams can be ambiguous and incomplete visualizations as not always all details can be covered visually.

For lookup:

- rectangular box with solid frame and orange fill: a class
- rectangular box with dashed frame and blue fill: a module, which is described in more detail elsewhere in the document
- rectangular box with dashed frame and purple fill: a set of URIs constituting a controlled vocabulary
- oval with solid frame and yellow fill: a data type
- arrow with white head and no label: a subClass relationship
- arrow with solid tip and label: a relationship (or property) other than a subClass relationship

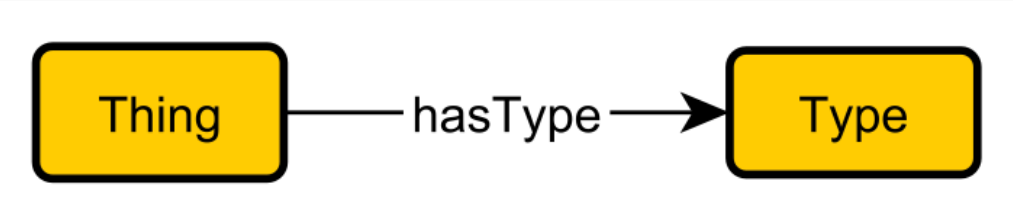
MODL Example: Explicit Typing

This pattern is used when there is a finite, but mutable number of types of a thing.

Competency Questions:

- What is the type of Event?
- Which type of apparatus is that?

MODL Example: Explicit Typing



MODL ODP: Explicit Typing.

Example:

```
ex:Dolphin rdfs:subClassOf ex:Thing .
ex:Mammal rdfs:subClassOf ex:Type .
ex:hasAnimalType rdfs:domain ex:Thing .
ex:hasAnimalType rdfs:range ex:Type .
```

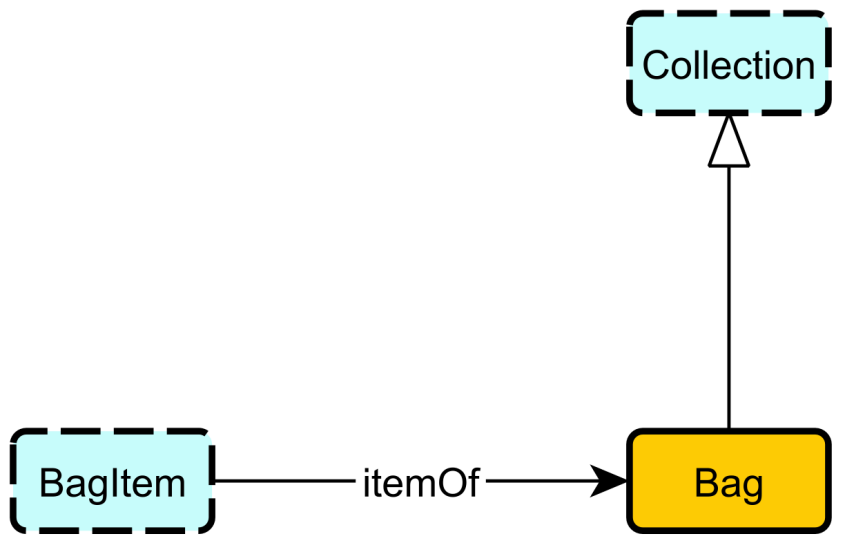
MODL Example: Aggregation, Bag, Collection

The pattern for an Aggregation, Bag, or Collection is relatively simple. The Bag is a type of unordered collection.

Competency Questions:

- What bag is this item an element of?
- What resource does this item refer to?
- What are the items contained in this bag?

MODL Example: Aggregation, Bag, Collection



MODL ODP: Aggregation, Bag, Collection.

Example:

```
ex:SoccerTeam rdfs:subClassOf ex:Bag .
ex:Bag rdfs:subClassOf ex:Collection .
ex:SoccerPlayer rdfs:subClassOf ex:BagItem .
ex:teamMemberOf rdfs:domain ex:SoccerPlayer .
ex:teamMemberOf rdfs:range ex:SoccerTeam .
```

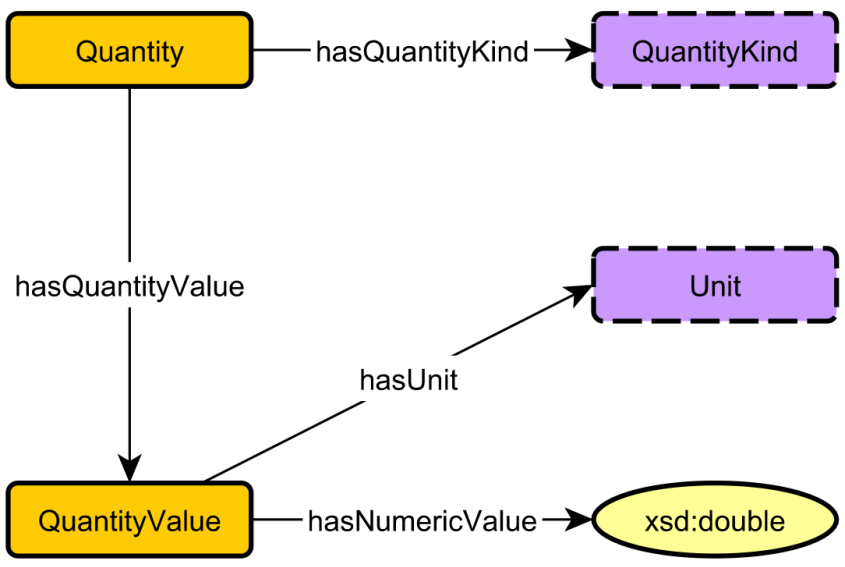
MODL Example: Quantities and Units

This pattern is heavily adapted from QUDT. This pattern allows a developer to express a quantity of some stuff. The nature of quantities is rather complex, due to the fact that there are a multitude of dimensions, unit types, and ways to measure quantities.

Competency Questions:

- How much does an elephant weigh in kilograms?
- How long is Jupiter from the Sun, at its farthest, in furlongs?
- How long ago was the Middle Ages?

MODL Example: Quantities and Units



MODL ODP: Quantities and Units.

Example:

```
ex:Height rdfs:subClassOf ex:Quantity .
ex:HeightValue rdfs:subClassOf ex:QuantityValue .
ex:EiffelTowerHeight rdfs:type ex:Height .
ex:EiffelTowerHeight ex:hasQuantityKind quantitykind:Length .
ex:EiffelTowerHeightValue rdfs:type ex:HeightValue,
                          ex:hasUnit unit:M,
                          ex:hasNumericValue “” 330^^^xsd:integer .
ex:EiffelTowerHeight ex:hasQuantityValue ex:EiffelTowerHeightValue .
```

OTTR - Reasonable Ontology Templates