Natural Language Processing

Lecture 6
Knowledge Representation for Computer Processing - Part 2

Ontology

Ontologies aim at capturing knowledge about the world explicitly for a specific purpose or task. In most cases, ontologies are actually more modest and aim to capture the knowledge about a certain domain (e.g., medicine), a specific phenomenon (e.g., the function of the human heart), or even a certain situation or event (e.g., a football match).

The reasons for developing an ontology:

- To share a common understanding of the structure of information among people or software agents;
- To enable the reuse of domain knowledge;
- To make domain assumptions explicit;
- To separate domain knowledge from operational knowledge;
- To analyze domain knowledge

Ontology

Potential applications for ontologies:

- Data integration;
- Content and functionality description at a “semantic level”;
- Providing relevant background knowledge for knowledge-intensive tasks

Ontology in philosophy

Ontology as a branch of philosophy is the science of what is, of the kinds and structures of objects, properties, events, processes and relations in every area of reality.

In the fundamental work described in the Categories, Aristotle introduced the basic ontological notions we use today: **universals** (also known as classes or concepts) and **particulars** (also known as instances, individuals, or objects).

In order to come up with a taxonomical organization of things, Aristotle proposed a “method of division” according to which a general category (a genus) is specialized into several (subgenera or species) by means of differentiae, i.e., properties on which they differ from each other.

Ontology in computer science

Ontologies are logical theories describing some aspect of reality.

1. “An explicit specification of a conceptualization” (*Gruber*, 1993);
2. “An explicit account or representation of [some part of] a conceptualization” (*Uschold*, 1996);
3. “A logical theory accounting for the intended meaning of a formal vocabulary, i.e., its ontological commitment to a particular conceptualization of the world” (*Guarino*, 1998);
4. “A formal explicit description of concepts in a domain of discourse” (*Noy and McGuinness*, 2001);
5. “A formal, explicit specification of a shared conceptualization” (*Studer et al.*, 1998)

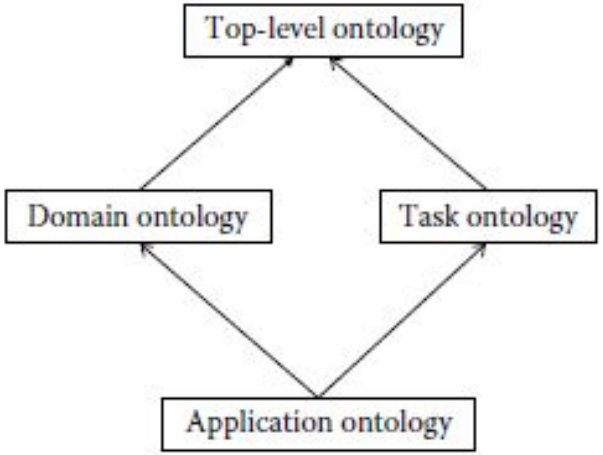
Anatomy of an Ontology

Prototypical axiomatic constructs:

- Subclass axioms;
- Non-taxonomic relations;
- Domain and range restrictions (on relation);
- Cardinality constraints;
- Part-of relations;
- Disjointness

Types of Ontologies (*Guarino*, 1998)

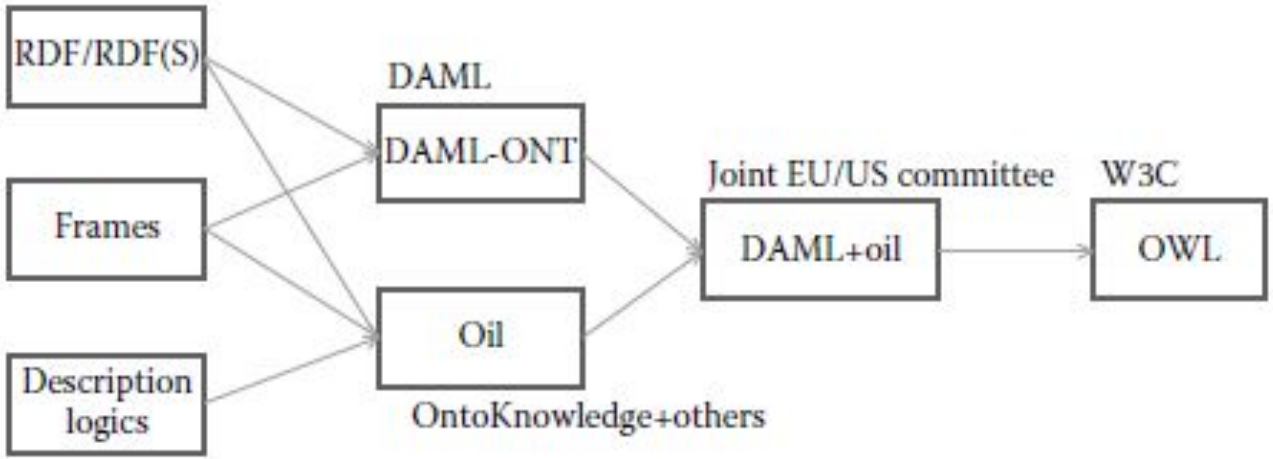
- **Top-level ontologies**: describing very general concepts related to time, space, matter, etc;
- **Domain ontologies**: describing the vocabulary in a specific domain;
- **Task ontologies**: describing the vocabulary of a generic task or activity, such as the task of selling something;
- **Application ontologies**: describing concepts that are often specializations of domain and task ontologies.



Types of Ontologies (*Uschold*, 1996)

- **Degree of formality**: ranging from *highly informal* (e.g., glossaries), *structured informal* over *semiformal* to *rigorously formal*;
- **Purpose**: mentioning different purposes such as (1) *communication between people*, (2) *interoperability* (between systems), where the ontology is used as an interchange format and (3) *knowledge reuse* across systems, as well as (in software engineering) (4) verification by *automated* consistency checks and assistance in the process of requirements specification;
- **Subject matter**: distinguishing (similarly to Guarino) between *domain ontologies*, *task*, *method or problem-solving ontologies*, and *meta-ontologies*, i.e., ontologies describing the vocabulary and semantics of a knowledge representation language.

Ontology Languages



Evolution of ontology languages

Ontology Languages

RDF: The Resource Description Framework essentially provides the vocabulary to express triples—i.e., constructs of the shape (*subject*,*predicate*,*object*)—which allow to provide elementary statements like the fact that Berlin is the capital of Germany, i.e., (*Berlin*,*capitalOf*,*Germany*);

RDF Schema allows one to introduce classes as well as to arrange them taxonomically via the *rdfs:subClassOf* property. In RDF Schema we could, for example, state that the class church is a subclass of building, i.e., (*Church*,*rdfs:subClassOf*,*Building*), thus specifying that all churches are buildings.

Ontology Languages

OWL *Full*: In addition to allowing one to define subclasses (as in RDFS),OWL allows to define complex classes, i.e., the class of *Medieval Cities* as the intersection between the class *City* and *Medieval*;

OWL *DL* was designed in order to have a computationally decidable OWL fragment. It restricts OWL Full by fostering a clear separation between classes and individuals (no individual can be a class at the same time);

OWL *Lite* is a subset of OWL DL removing several sources of complexity for practical reasoning algorithms

Ontology Engineering

Principles:

1. **Clarity**: An ontology should clearly state the meaning of the defined vocabulary. The definition should be objective and independent of computational aspects. Ontologies should be formal in the sense that definitions should be stated in the form of logical axioms (if possible), aiming for complete definitions in terms of necessary and sufficient conditions. All definitions in the ontology should be documented in natural language;
2. **Coherence**: ontologies should be coherent in the sense that the axioms contained therein should be logically consistent;

Ontology Engineering

3. **Extensibility**: An ontology should be designed in a general way such that it anticipates further usage that it was not conceived for;

4. **Minimal encoding bias**: The specification of the ontology should be as independent as possible of issues concerning its encoding in any particular ontology language;

5. **Minimal ontological commitment**: An ontology should make as few assumptions as necessary about the world being modeled, thus allowing other parties to reuse the ontology for their purposes.

Ontology Engineering

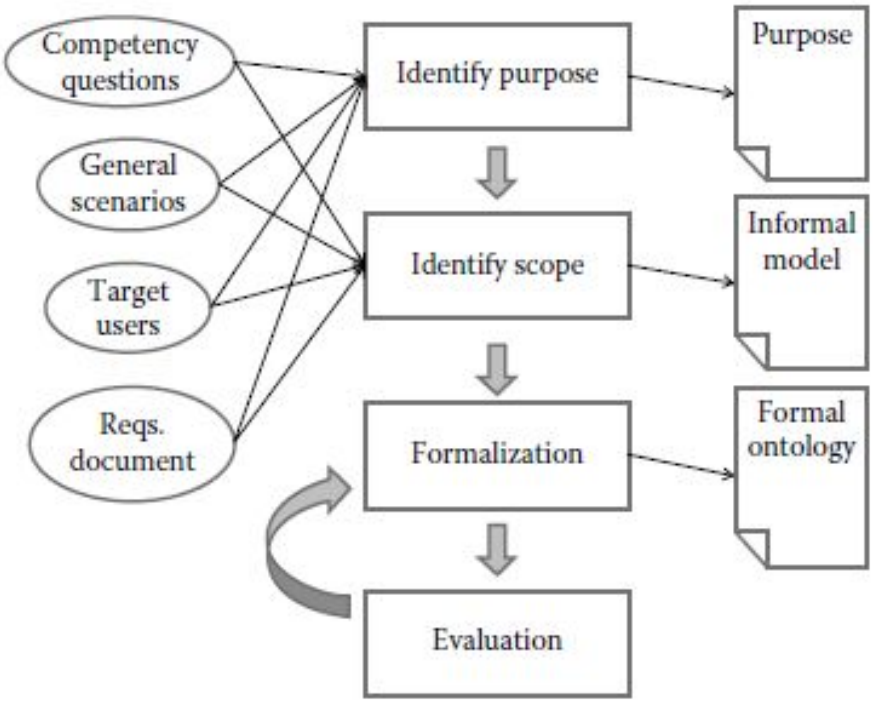
Building an ontology:

1. Identification of the purpose of the ontology;
2. Building the ontology, consisting of the following substeps:
    a. Ontology capture (at the so-called knowledge level);
    b. Ontology coding;
    c. Integration of existing ontologies;
3. Evaluation;
4. Documentation

Ontology Engineering

- Defining the purpose and scope of an ontology beforehand is an important step of all methodologies;
- Every ontology engineering project should follow a clearly defined methodology. This decreases the risk of failure;
- Many methodologies are certainly “underspecified” in the sense that they do not spell out how to reach consensus or how to decide on certain modeling issues. Thus, while they support the overall process (macro-level), they certainly do not support the design at the micro-level.

Ontology Engineering



Ontology Reuse

Types of “ontology reuse”:

- **Fusion or Merging**: The process of unifying various ontologies (typically on similar or related domains) into a new ontology. Hereby, the contribution of each source ontology might not be clearly identifiable;
- **Composition or Integration**: The process of selecting parts of certain ontologies (typically on different domains) and integrating them into a new ontology in a modular fashion as clearly identifiable submodules drawn from other ontologies.

Ontology Reuse

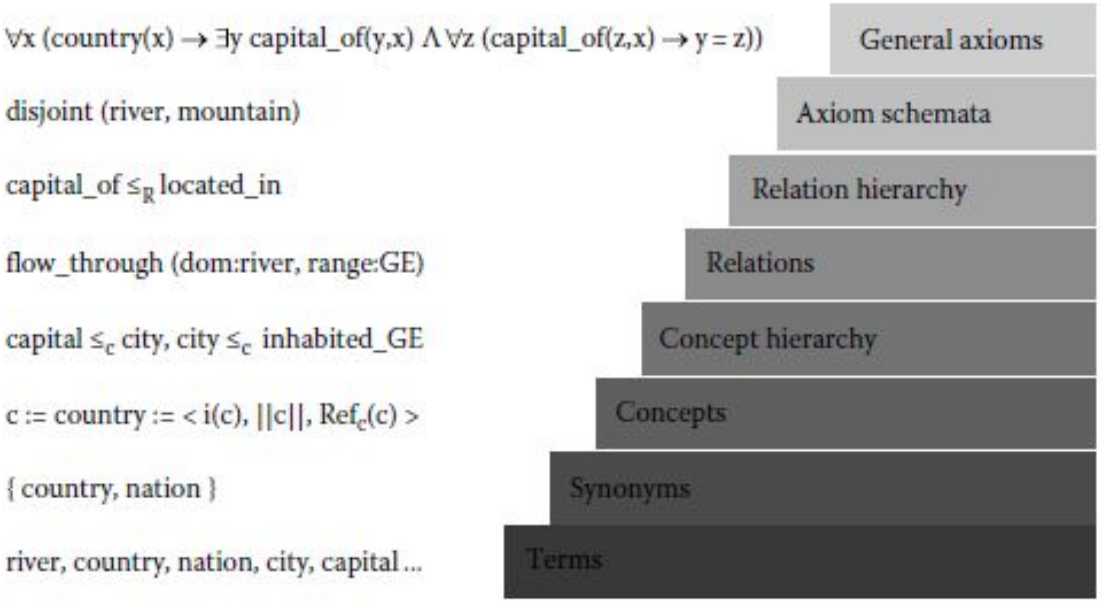
Methods for the alignment of ontology elements from different ontologies:

1. **structure-based methods** that compute semantic similarity relative to the structure of the ontology (similar classes share similar parents, sisters, etc.);
2. **instance-based methods** that compute similarity based on the overlap in terms of instances for any given pair of classes (or properties);
3. **lexical methods** that build on string distance between the names of classes/properties or, in more sophisticated versions, on the similarity of linguistic contexts of class/property instances (which is only tractable if a semantically annotated textual data set is available, e.g., in the case of ontology-based information extraction).

Engineering Paradigms

- **The logico-philosophical/mathematical paradigm**: borrowing directly from the ideas of Aristotle in the sense that the different properties (differentiae) of things are responsible for introducing ontological distinctions in terms of subclasses;
- **The cognitive paradigm**: making ontological distinctions on the basis of what is intuitively perceived as different without the need for an explicit and formal specification of the features on which classes differ;
- **The linguistic approach**: typically assumes that an ontology needs to reflect linguistic distinctions;
- **The computer science approach**;
- **The domain specialist approach**: these ontologies are typically organized for browsing or classifying resources to facilitate retrieval

Ontology Learning



Ontology learning "layer cake"

Ontology Learning

Terms are linguistic realizations of domain-specific concepts and hence a required input for all of the other layers higher up in the layer cake.

The **synonymy** level addresses the acquisition of semantic variants, i.e., words as well as more complex terms that have similar meanings and can be mutually exchanged within semantically similar contexts.

Concept learning should provide an intensional definition of the concept, an extensional set of concept instances, and a set of linguistic realizations for the concept.

Ontology Learning: Concept Hierarchy

Hearst patterns (Hearst, 1992):

- Such NP as {NP,}* {or | and} NP (e.g., "*religious buildings such as churches and mosques*")
- NP {, NP}* {,} {and | or} other NP (e.g., "*senators, governors, and other politicians*")
- NP {,} including {NP,}* {or | and} NP (e.g., "*green vegetables, including beans and spinach*")
- NP {,} especially {NP,}* {or | and} NP (e.g., "*office furniture, especially desks and bookcases*")

Ontology Learning: Concept Hierarchy

Additional patterns:

- **Exception**: NP except for NP (e.g., "*European countries except for Spain*");
- **Apposition**: NP, the capital of NP (e.g., "*London, the capital of England*");
- **Definites**: the PN hotel (e.g., "*the Hilton hotel*");
- **Copula**: NP is a NP (e.g., "*Brazil is a South American country*");
- **Doubly anchored hyponym patterns** as suggested by Kozareva et al. (2008), (e.g., *presidents such as Clinton and PN*)

Ontology Learning

Relation extraction depends rather on the syntagmatic aspect of language, i.e., the extent in which more or less complex linguistic units are connected within a sentence, paragraph, or more extensive discourse.

The task of **learning general axioms and axiom schemata** has not attracted much attention in the ontology learning community so far.

OWL: Classes and Instances

In general **classes** are used to group **individuals** that have something in common in order to refer to them. Hence, classes essentially represent sets of individuals.

Example:

Class: *women*

Instance: *Mary*

OWL: Class Hierarchies

A subclass relationship between two classes A and B can be specified, if the phrase “every A is a B” makes sense and is correct.

Example: *Woman* is subclass of *Person*

Equivalent classes: *Person* and *Human*

OWL: Class Disjointness

“Incompatibility relationship” between classes is referred to as (class) **disjointness**.

Example: *Man* and *Woman*

OWL: Object Properties

The entities describing in which way the individuals are related, are called **properties**.

Example: John **has wife** Mary.

Negative property is also possible: e.g. **doesn’t have wife**

OWL: Property Hierarchies

Whenever *B* is known to be *A*'s wife, it is also known to be *A*'s spouse

OWL: Domain and Range Restrictions

Domain restriction: hasWife - Man

Range restriction: hasWife - Woman

The information that Sasha is related to Hillary via the property *hasWife*, a reasoner would be able to infer that Sasha is a man and Hillary a woman.

OWL: Complex Classes

The **intersection** of two classes consists of exactly those individuals which are instances of both classes: the class *Mother* consists of exactly those objects which are instances of both *Woman* and *Parent*.

The **union** of two classes contains every individual which is contained in at least one of these classes. Therefore we could characterize the class of all parents as the union of the classes *Mother* and *Father*.

The **complement** of a class corresponds to logical negation: it consists of exactly those objects which are not members of the class itself: e.g. *Parent* and *Childless Person*.

OWL: Property Characteristics

Sometimes one property can be obtained by taking another property and changing its direction, i.e. **inverting** it: the property *hasParent* can be defined as the inverse property of *hasChild*.

The property is called **symmetric** when the direction of a property doesn't matter: the property *hasSpouse* relates *A* with *B* exactly if it relates *B* with *A*.

A property can be **asymmetric** meaning that if it connects *A* with *B* it never connects *B* with *A*: the property *hasChild*.

Two properties are **disjoint** if there are no two individuals that are interlinked by both properties: parent-child marriages cannot occur.

OWL: Property Characteristics

Properties can be **reflexive**: such a property relates everything to itself: everybody has himself as a relative.

Properties can be **irreflexive**, meaning that no individual can be related to itself by such a role: nobody can be his own parent.

The property is called **functional** if every individual can be linked by this property to at most one other individual: *hasHusband*.

A **transitive** property interlinks two individuals *A* and *C* whenever it interlinks *A* with *B* and *B* with *C* for some individual *B*: i*sAncestor*.

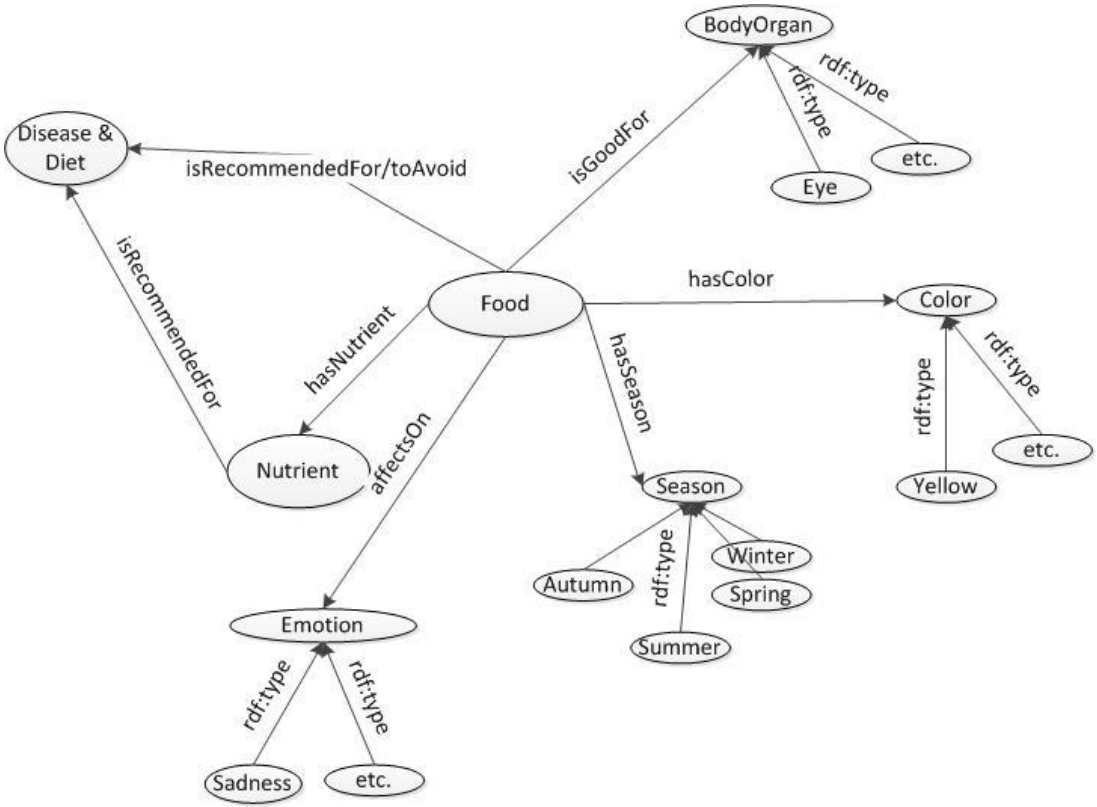
Software

https://www.w3.org/TR/2009/REC-owl2-primer-20091027/ - OWL Documentation

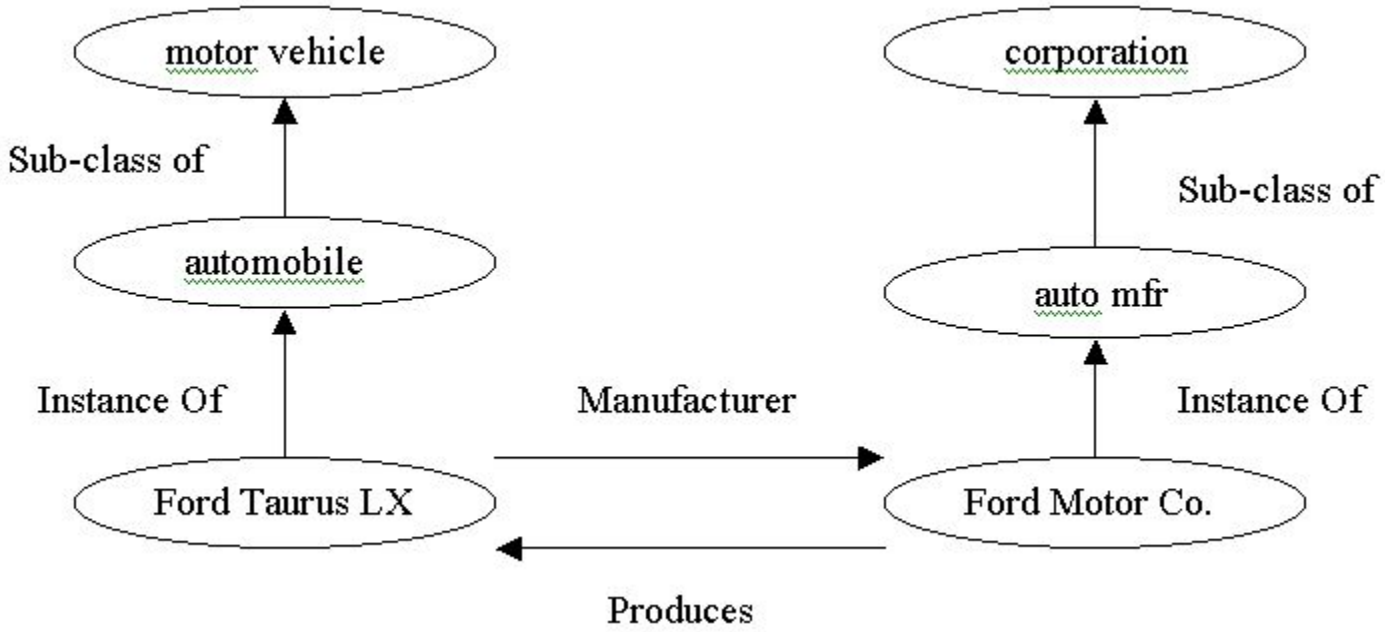
http://protege.stanford.edu/ - Protege: ontology editor and framework for building intelligent systems

http://www.academia.edu/7891589/Protege_Tutorial - Protege Tutorial

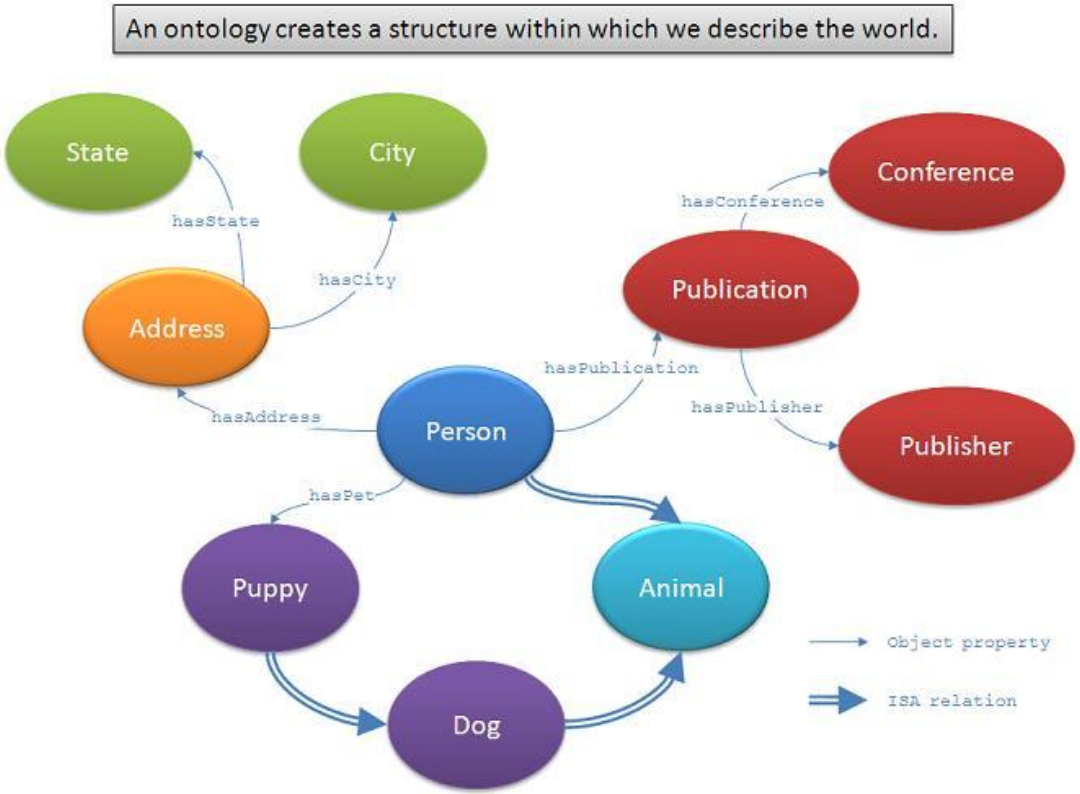
Ontology: Example



Ontology: Example



Ontology: Example



Thank you for your attention!