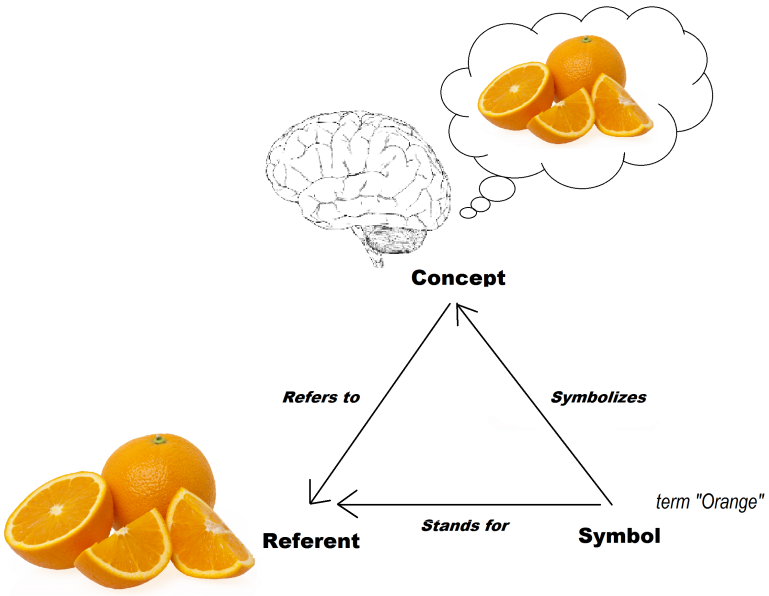
# 1 Managing Semantic Data

## 1.1 Introduction to Ontologies and Ontology Engineering

### 1.1.1 Ontology Basics

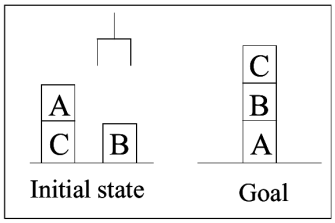
#### Understanding ontology

#### What is concept?




The meaning triangle according to Ogden&Richards, 1969

#### What is conceptualization?




**Conceptualization-1**

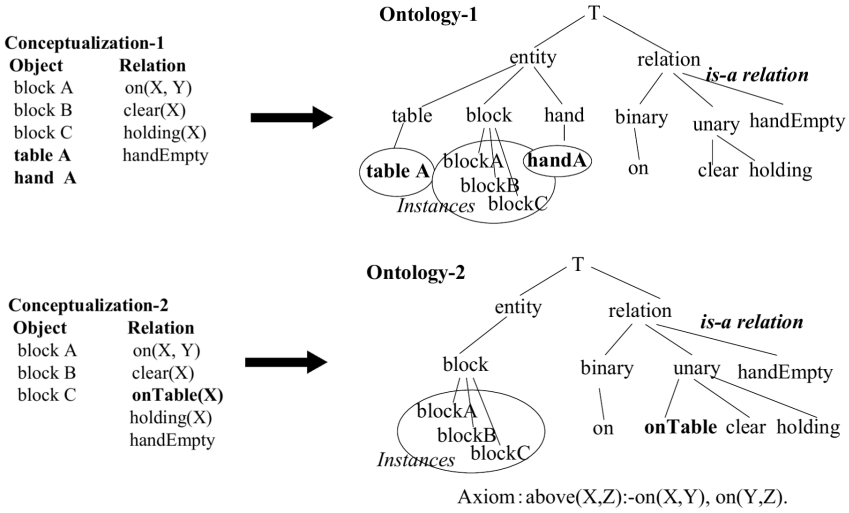
| Object | Relation |
|---|---|
| block A | on(X, Y) |
| block B | clear(X) |
| block C | holding(X) |
| **table A** | handEmpty |
| **hand A** | |

**Conceptualization-2**

| Object | Relation |
|---|---|
| block A | on(X, Y) |
| block B | clear(X) |
| block C | **onTable(X)** |
| | holding(X) |
| | handEmpty |

The block world problem and its two different conceptualizations taken from [**mizoguchi2003part**]. Objects **table A** and **hand A** does not exist in **Conceptualization-2** as they are substituted by **onTable** relation.

## Ontology based on conceptualization




Ontologies created based on **Conceptualization-1** and **Conceptualization-2** taken from [**mizoguchi2003part**].

## Basics

- **Conceptualization** is set of objects which an observer thinks that they exist in target world (world of interest). It provides backbone of the conceptual structure of such world.
- **Ontology** is "explicit specification of conceptualization" [**gruberOntology**].

- It contains hierarchically organized structure of concepts and relations between them. Such structure defines meaning of objects appearing in the target world.
- It is type of KOS where each term from controlled vocabulary is put into some specific place within its complex structure.
- It is declarative description of fundamental understanding of the target world.

**Note**

In case when OWL2 is used to represent an ontology – OWL classes represents concepts, OWL individuals represent concrete objects, and combination of OWL classes and OWL properties represent relations of the target world. **From now on we will use OWL2 terminology whenever it is possible**.

### Using ontology in the most correct way

### But what conceptualizations are the best to use with ontology?

- There aren't the best conceptualizations! There are only conceptualization that fit the best for the given use of the ontology and its formal language capabilities.
- On the other hand, ontological engineering is currently viewed as a challenge to enabling *knowledge sharing* and *reuse* which other fields of AI failed to realize. In this sense we will call ontology more "**ontological**" or more "**ontologically correct**" if it suits better for this purpose.
- To create correct ontologies first we need to understand fundamental issues of building ontologies such as distinction between *is-a* and *part-of* relations, distinction between *classes*, *instances* and is-a relation etc.

### Importance of is-a and part-of relation

- Is-a hierarchy of classes allows us to use terms with different specificity. But can we count or identify objects using hierarchy properly? Consider query that we want to ask an ontology :
- **How many blue things are in this room?**
  - Blue table has many blue things on it.
  - Property "blue" does not allows us to count objects.
  - **To answer this query we need proper categorization of things.**
  - We need to understand what are object *wholes* and what are object *parts*.
- **How many tables are in this room? How many furniture are in this room?**
  - Classes can be organized into hierarchies according to different properties.

  - **To answer this query we need proper way of class organization.**
  - We need to understand how is *identity* of an object propagated through the use of *is-a* links.

### Proper representation of is-a hierarchy

- *is-a hierarchy* should not be simple classification of classes, but rather represent inheritance of *essential property* of each class.
- correct is-a hierarchy reveals the intrinsic structure of the target world which help people to understand in-depth the class without its definition.
- ad-hoc classification are usually purpose-dependent thus less shareable. "Correct ontology" should not be knowledge base for problem solving but rather foundation of knowledge base for various purposes.
- is-a hierarchies should be shareable and stable backbone of the knowledge structure.
- with proper is-a hierarchy we can safely propagate properties of super-class to properties of its sub-classes

### Proper representation of part-of relations

- *part-of relation* is used to represent a thing as a whole which is composed of few parts and is usually transitive.
- There are at least 5 types of *part-of relations* [**mizoguchi2004part**] :
  - Functional *part-of*, where contribution of the part to the whole is functional (e.g. wheel is *part-of* bike)
  - Qualification *part-of*, where instances of the relation must have qualification/-role in order to become part of whole. (e.g. husband *part-of* married-couple)
  - Spatial/temporal relation *part-of*, where instances of the relation must satisfy spatial/temporal constraints to a become part of the whole. (e.g. tree *part-of* forest, mountain part-of mountains)
  - Staff *part-of*, where the whole is stuff. (e.g. a piece of pie *part-of* pie)
  - Material *part-of*, where instances are materials of the whole. (e.g. glass *part-of* cup)

### Note

Although *part-of relation* is essential for building ontologies, OWL2 does not provide vocabulary to represent it. Thus, semantics of *part-of relation* must be defined within the ontology.

### Difference between part-of and is-a relationship

- It does not make sense for two classes be in relation part-of and is-a relation at the same time.
- Consider following axioms in context of a plant:
  1. normal operation *is-a* operation;

     restoration operation *is-a* operation
  2. normal operation *part-of* operation;

     restoration operation *part-of* operation
- **How can we resolve this issue?**

  (Hint : differentiate between event/process and action)

### Difference between instance-of and is-a relationship

- Class represent set of concrete things in target world
- Instance is element of those sets, i.e. they represent undividable elements
- Thus it does not make sense for an ontological entity to be instance and class within ontology at the same time.
- Consider following axioms about cars:

  my personal car *instance-of* Ford;

  Ford *instance-of* car brand
- **How can we resolve this issue?**

  (Hint : differentiate between Ford within both axioms, e.g. who is owner of Ford for each axiom)
- What happens if we represent above axioms within *OWL2* ontology?

### Classification of ontologies

### Different types of ontologies

- There are many classification of ontologies. We will distinguish only four types :
  - **Top-level ontologies** – describe very general concepts such as event, object, action. They are independent of a particular problem or domain.
  - **Domain ontologies** – describe the vocabulary related to a generic domain such as cultural tourism, or medicine.

  - **Task ontologies** – describe the vocabulary related to a generic task or activity such as selling, or diagnosing.
  - **Application ontologies** – describe classes that depend on both a specific task and a specific domain. The classes within ontology often correspond to roles played by domain entities performing some task.

- Domain and task ontologies typical specializes top-level ontologies.
- Application ontologies typically specializes both task and domain ontologies.

#### Study materials

- Tutorial on Ontological Engineering (from Mizoguchi Laboratory) `http://www.ei.sanken.osaka-u.ac.jp/japanese/tutorial-j.html`
  - Part "What is an ontology?" – written by T.Gruber, contains definition of ontology from multiple perspectives
  - Part 1 – details about conceptualizations, definition of ontology, types of ontologies
  - Part 3 – details about "ontologicaly correct" ontologies (part-of, is-a relations etc.)

### 1.1.2 Ontology Engineering

#### Basics

#### Basic terms

- **Ontology Engineering** is research field that covers a set of activities appearing during conceptualization, design, implementation and deployment of ontologies. It includes topics such as philosophy, metaphysics, knowledge representation formalisms, development methodology, knowledge sharing and reuse, knowledge management, systemization of domain knowledge, standardization, evaluation etc.
- **Ontology life cycle** is the specific sequence of activities that are carried out by ontology practitioners for developing an ontology.
- **Ontology life cycle model** is a framework that provides general structure on which activities of the ontology life cycle can be mapped.

#### Ontology Engineering activities

List of Ontology Engineering activities :

- Ontology Conceptualization
- Ontology Integration

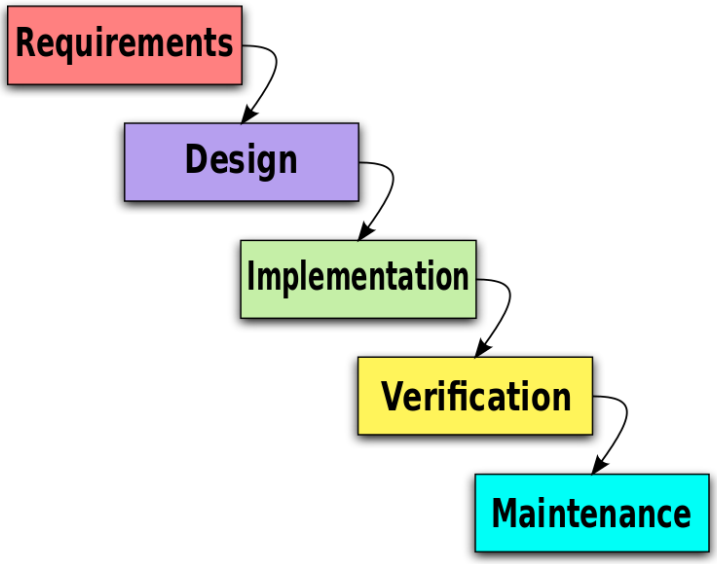
- Ontology Population
- Ontology Elicitation
- Ontology Learning
- Ontology Localization
- Ontology Matching
- Ontology Search
- Ontology Annotation
- Ontology Specialization
- Ontology Diagnosis
- Ontology Repair
- Ontology Modularization
- Ontology Reengineering
- Ontology Reuse
- Ontology Versioning
- Ontology Evaluation
- ...

Comprehensive glossary of Ontology Engineering activities can be found at `http://mayor2.dia.fi.upm.es/oeg-upm/files/pdf/NeOnGlossary.pdf`

### Ontology life cycle models

There are multiple models to sequence activities for development of ontology :

- Waterfall Model
- "V" Model
- Incremental Model
- Iterative Model
- Evolutionary Prototyping
- Rapid Throwaway Prototyping
- Spiral Model




### Requirements engineering

### Requirements engineering basics

- **Requirement engineering** is an activity in which we view ontology as black box while trying to describe requirements that the ontology should fulfill.
- Requirements are of two types :
  - **Functional requirements** describing internal structure and content
    - Query results
    - Inferences

    - Error checking
    - ...
  - **Non-functional requirements** describing overall structure, acceptance, guidelines and rules for development etc.
    - Coverage
    - Efficiency
    - Documentation
    - ...

- We will focus here rather on "computational ontologies" (as part of a software system performing some task)

### Requirements engineering – Functional requirements

There are three types of functional requirements that are used within requirement engineering :

1. **Competency question** (CQ) is natural language question that the ontology should be able to answer
   - *Simple lookup queries* – Who are participant of certain event?
   - *Queries expressing inferences and constraints* – If people have children, is a specific person grandparent or not? Is it valid to have 3 parents?
2. **Contextual statement** is an axiom that must hold in the ontology expressed in natural language – Every person has at exactly 2 parents. Grandparent is person who has a child who has a child.
3. **Reasoning (inference) requirement** specify the input and output data for a reasoning task – we need to query directly for all the grandparents

**Note**

Contextual statements and reasoning requirements are used to clarify/remove complex CQs

### Overview of methodologies

### METHONDOLOGY

- **METHONDOLOGY** was developed at Polytechnic University of Madrid and is based on IEEE standards for Developing Software Life Cycle Processes.
- It provides guidelines for
  - **Project management process** – planning, project control, quality control etc.

  - **Ontology development process** – envisioned use of the ontology, explication of the envisioned users, conceptualization of target domain, formalization of ontology, implementation etc.
  - **Support activities** – knowledge acquisition, evaluation, ontology integration, documentation, version management etc.

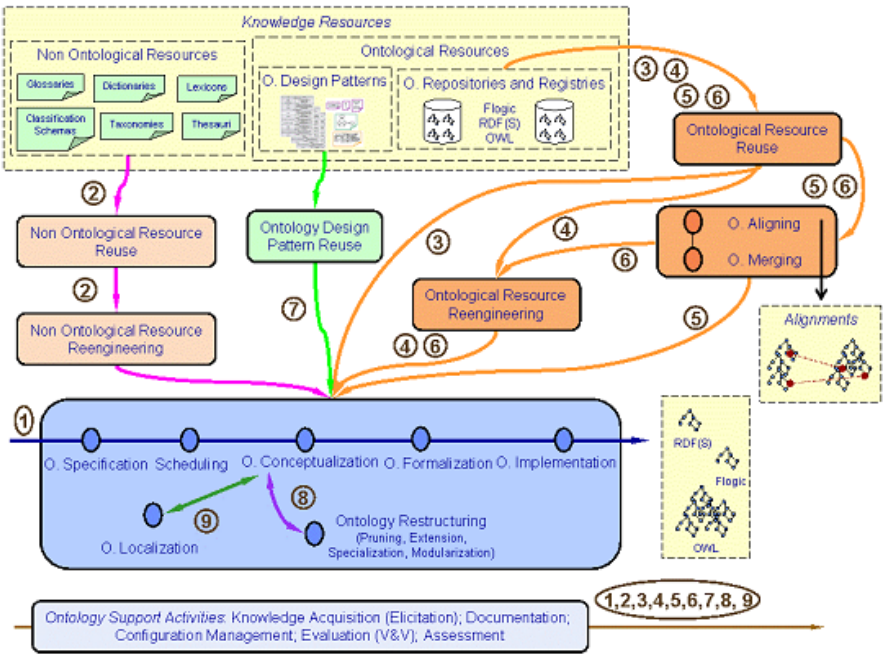
### On-To-Knowledge methodology

- **On-To-Knowledge methodology** was developed in Karlsruhe University.
- It is based on two-loop architecture :
  - **Knowledge process** – normal knowledge use and evolution process
  - **Knowledge meta-process** – methodology for ontology development composed of 5 major steps (with 13 sub-steps)
    * Feasibility study
    * Kickoff
    * Refinement
    * Evaluation
    * Application and evaluation

### The NeOn Methodology – introduction

- **The NeOn Methodology** [**suarez2010neon**] is scenario-based methodology with emphasis on
  - reuse and reengineering of knowledge aware resources
  - collaborative and argumentative ontology development
  - building of ontology networks, as opposed to custom-building new ontologies from scratch.
- It defines following scenarios
  1. From specification to implementation
  2. Reusing and re-engineering non-ontological resources (NORs)
  3. Reusing ontological resources
  4. Reusing and re-engineering ontological resources
  5. Reusing and merging ontological resources
  6. Reusing, merging and re-engineering ontological resources
  7. Reusing ontology design patterns (ODPs)
  8. Restructuring ontological resources
  9. Localizing ontological resources

### The NeOn Methodology – overall architecture




Available at `http://mayor2.dia.fi.upm.es/oeg-upm/index.php/en/methodologies/59-neon-methodology`

### Other methodologies

- Cyc methodology
- Grüninger and Fox's methodology
- Uschold and King's methodology
- KACTUS methodology
- SENSUS methodology
- DILIGENT methodology
- TOVE methodology
- Activity-First Method (AFM) in Hozo

### Three-layer model of guidelines

An ontology engineering methodology can be composed of three-layers guidelines [**mizoguchi2004tutorial**] :

- **Top-layer** – The whole building process compliant with the conventional software development process (The methodologies described so far includes mostly only this level guidelines and partially the next level)
- **Middle-layer** – Generic constraints and guidelines which specify major steps (AFM is example of a methodology mainly concerned with this level)
- **Bottom-layer** – The most fine-grain guidelines such as those for class identification process, etc. (Ontology development tutorial by Noy [**noy2001ontology**] have few guidelines at this level)

**Note**
Middle-layer and bottom-layer guidelines are essential for novices to develop "correct ontology" as they directly influence the quality of the ontology developed.

### Summary of methodologies

- *Cyc, Uschold and King, Grüninger and Fox, KACTUS, METHONDOLOGY, SENSUS, On-To-Knowledge* methodologies can be used to build ontologies from scratch.
- *METHONDOLOGY, On-To-Knowledge methodology* are very useful in development process management.
- *Uschold and King methodology* is useful for obtaining informal ontology in early phase of development.
- *Activity-First Method methodology* is useful for building task and domain ontologies.

### Selected methodological guidelines

#### Guideline to ontology requirement specification

Based on NeOn methodology ontology requirement specification can be formed from a set of ontological needs as follows :

1. Identify the purpose, scope and implementation language
2. Identify the intended end-users
3. Identify the intended uses
4. Identify functional and non-functional requirements
5. Group requirements (the list of CQs)
6. Validate set of requirements – *if not valid jump to step 4*
7. Prioritize requirements (optional)
8. Extract terminology and its frequency

**Guideline for ontology life cycle models**

Based on assumption about ontology requirements we can use following ontology life cycle models :

- Ontology requirements are known from the beginning
  - Waterfall Model
  - "V" Model
  - Incremental Model
  - Iterative Model
- Ontology requirements are not known from the beginning and can change during the project
  - Evolutionary Prototyping
  - Rapid Throwaway Prototyping
- Uncertainties in the ontology requirements can derive into risks in the project
  - Spiral model

**Guidelines for taxonomy construction direction**

- There are three approaches to construct taxonomies :
  - top-down approach
  - bottom-up approach
  - middle-out approach
- Try to combine the approaches as much as possible!

**Guideline for reusing ontological resource**

- There are multiple ways how to reuse ontological resources :
  - ontologies as wholes
  - ontology modules
  - ontology design patterns
  - ontology statements
- In addition to reuse, reconstruction, re-engineering, and merging can be used.