Ontology engineering
- The basic process

LINKÖPING
UNIVERSITY

What is an Ontology?

- We consider *computational ontologies*
- In this sense an ontology is an artifact that can be used as a part of a software system
- A **model** of something – whether the real world or some imaginary world it is never an exact reproduction of it
- In other words it is:
  - a descriptive specification of a set of contextual assumptions about a domain of interest
- While it is usually not:
  - a prescriptive specification of the inner structure of ‘true reality’

Why develop an ontology?

- To develop a shared common understanding of the structure and meaning of information
  - among people
  - among software agents
  - between people and software
- To enable standardisation and/or reuse of domain knowledge
  - to avoid “re-inventing the wheel”
  - to introduce standards to allow interoperability
- To make domain assumptions explicit
  - easier to change domain assumptions
  - easier to understand and update legacy data
- To separate domain knowledge from operational knowledge (i.e. separate data from operations in a system)
  - re-use domain and operational knowledge separately (e.g., configuration based on constraints)

What is Ontology Engineering?

- Ontologies are artifacts
  - Have a structure (linguistic and logical)
  - Their function is to “encode” a description of the world (actual, possible, counterfactual, impossible, desired, etc.) for some **purpose**
- Ontologies must match both **domain** and **task**
  - Allow the description of the entities (“domain”) and their attributes and relations, *e.g. cars and their characteristics*
  - Serve a purpose (“task”), e.g. *finding cars that match some customer criteria*
- Ontologies have a lifecycle
  - Created, evaluated, fixed, and exploited just like any artifact, e.g., like software
  - Their lifecycle has some special characteristics regarding:
    - *Data, processes, argumentation, design patterns…*

Two main kinds of ontologies

- Coverage-oriented ontologies
  - They cover the terminology/metadata/textual corpora/ folksonomies … that fit a specific domain
- Task-oriented ontologies
  - They are able to give a structure to a knowledge base that can be used to answer competency questions and do reasoning

- Currently on the Web
  - a mass of heterogeneous data and ontologies, either expressed or portable to RDF (DB lifting, rdf-ized sources, etc.)
  - with generally low quality in some quality dimension/aspect

What is needed for designing ontologies

- Resources (“raw” material) - from domain experts
  - *Reengineering* is key
  - Thesauri2ABox, Lexicon2TBox, Tags2ABox, etc.
  - Texts, interview transcripts etc.
- Formal languages, e.g. RDF(S) and OWL
- Solutions (target configurations for the raw material)
  - *Design patterns*
  - Reusable/standard ontologies
- Methods (production from raw material)
  - *Collaboration workflows*
  - Search, evaluation, selection, reengineering procedures, pattern matching and composition
- Tools
  - Ontology engineering tools (TopBraid Composer, Protégé 4 and 5, WebProtége ...)
  - Management and versioning (github, w3id, Ontoology, ...)
  - API:s and frameworks for using and applying ontologies (Jena, OWL API, various triple stores...)

The overall process

- Project scoping and initialisation
  - Figure out what to do, when, how and why and with what resources
- Project realisation
  - Elicit requirements, formalise the ontology, evaluate and test
- Deployment and maintenance
  - Apply the ontology to your data, and/or in your software system, maintain the ontology over time

The overall process

- Project scoping and initialisation
  - Figure out what to do, when, how and why and with what resources
- **Project realisation**
  - **Elicit requirements, formalise the ontology, evaluate and test**
- Deployment and maintenance
  - Apply the ontology to your data, and/or in your software system, maintain the ontology over time

Requirements engineering

What are “requirements”?

- Viewing an ontology as a black box… what should that box provide?
- Functional requirements
  - Query results?
  - Inferences?
  - Error checking?
  - …
- Non-functional requirements
  - Coverage
  - Efficiency
  - Documentation
  - Changeability – extendibility
  - …

Internal structure, and content

Overall structure, acceptance
→ Guidelines and rules for development

Non-functional Requirements

- Coverage
  - How important is the coverage of the domain? How will the ontology be updated?
- Efficiency
  - What OWL profile to use?
  - Reasoning off-line or online?
  - Query optimization, e.g. not requiring inferences
- Documentation
  - Labels and comments?
  - Naming conventions
- Changeability – extendibility
  - Should future extensions be prepared for?
  - Alignment to online ontologies, standards?

Competency Questions (CQs)

- What do we want to ask the knowledge base?
  - Typical questions/queries

- Can be used as requirements and as the basis for unit tests

- Example
  - Example fact to be represented:
    "Anders works at SAAB training systems"
  - Generalised knowledge, 'instance-free sentence':
    "People work at companies"
  - Potential competency questions:
    - For what company does a certain person work?
    - What persons work at a certain company?

Competency Questions (cont.)

- Requirements of an ontology = competency questions + additional constraints/restrictions and reasoning requirements
  - Additional restrictions (axioms) to be defined on the model, usually restrictions over data
  - Reasoning requirements specify the facts that have to be inferred before they can be retrieved through a query, i.e. they are not explicitly stored in the KB

- Example constraints/restrictions
  - Each flower shop sell at least 2 kinds of flowers.
  - Every flower shop sells some roses.
- Example reasoning requirement
  - The class of “Popular flowers” is the flowers that have been sold more than twice the past week

Trade-off: Software vs. Ontology

- What functionality is going to be put into the software and what is going to be part of the ontology?
  - An OWL reasoner is nothing more than general-purpose code for processing data – why not use specific code in our system instead?
- Ontology pro:s
  - The ontology makes assumptions explicit
    - Ontologies can be published and shared together with the data
  - The ontology can be changed at runtime without changing the code (or with minimal changes)
  - The reasoning procedures are sound and well-defined, and they are reused for all inferences
- Software pro:s
  - More efficient?

Hands-on: CQs