Semantic Web Technologies
Ontology Engineering
Semantic Web – Architecture

here be dragons...

Semantic Web
Technologies
(This lecture)

Technical
Foundations

Query: SPARQL

Ontology: OWL

Rules: RIF

Schema: RDF-S

Data Interchange: RDF

Data Interchange: XML

URI

Unicode

User Interface and Applications

Trust

Proof

Unifying Logic

Cryptography

Berners-Lee (2009): Semantic Web and Linked Data
Ontology Engineering

• Ontologies are a key ingredient for the Semantic Web
• How we have built ontologies so far
  – Read the requirements
  – Pick a starting point at random
  – Start playing around in Protégé
  – Trial and error driven
• That was rather "Ontology Hacking" than "Ontology Engineering"
Ontology Engineering

• How to build ontologies?
  – Methodologies

• Hot to build *good* ontologies?
  – Best Practices
  – Design Patterns
  – Anti Patterns
  – Top Level Ontologies
Warning

- Today's lecture contains a massive amount of philosophy (for computer scientists)
Methodologies

• Known, e.g., from Software Engineering

http://geekandpoke.typepad.com/geekandpoke/2012/01/simply-explained-dp.html
Methontology (Fernández et al., 1997)

Gómez-Pérez et al. (2004): Ontological Engineering
Methontology (Fernández et al., 1997)

• Step by step from less to more formal ontologies
• Stepping back is allowed
• Documentation is produced along the way

• Glossary
  – Terms, descriptions, synonyms, antonyms
• Taxonomy
  – Sub class relations
• Ad hoc binary relations
  – a.k.a. ObjectProperties
• Concept dictionary
  – contains: terms, descriptions, relations, instances (optional)
Methontology (Fernández et al., 1997)

- Concept dictionary (example)

<table>
<thead>
<tr>
<th>Concept name</th>
<th>Class attributes</th>
<th>Instance attributes</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA7462</td>
<td>--</td>
<td>--</td>
<td>same Flight as</td>
</tr>
<tr>
<td>American Airlines Flight</td>
<td>company Name</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>British Airways Flight</td>
<td>company Name</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Five-star Hotel Flight</td>
<td>number of Stars</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Flight</td>
<td>--</td>
<td>--</td>
<td>same Flight as</td>
</tr>
<tr>
<td>Location</td>
<td>--</td>
<td>name size</td>
<td>is Arrival Place of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>is Departure Place of</td>
</tr>
<tr>
<td>Lodging</td>
<td>--</td>
<td>price of Standard Room</td>
<td>placed in</td>
</tr>
<tr>
<td>Travel</td>
<td>--</td>
<td>arrival Date</td>
<td>arrival Place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>company Name</td>
<td>departure Place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>departure Date</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>return Fare</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>single Fare</td>
<td></td>
</tr>
<tr>
<td>Travel Package</td>
<td>--</td>
<td>budget final Price</td>
<td>arrival Place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name</td>
<td>departure Place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>number of Days</td>
<td>accommodated in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>travel Restrictions</td>
<td>travels in</td>
</tr>
<tr>
<td>USA Location</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Gómez-Pérez et al. (2004): Ontological Engineering
Building Good Ontologies

• Real example SNOMED (a medical ontology)
  
  Finger partOf Hand .
  Hand partOf Arm .
  partOf a owl:TransitiveProperty .
  Surgery rdfs:subClassOf Treatment .
  onBodyPart rdfs:domain Treatment .
  onBodyPart owl:propertyChain (onBodyPart, partOf) .

• This allows for inferences such as
  − An operation of the finger is also an operation of the hand
    (and an operation of the arm).

• So far, so good...
  
  Amputation subClassOf Surgery .
OntoClean

• A collection of analysis methods and tests
  – Does my class hierarchy make sense?

• Developed ~2000-2004 by Nicola Guarino and Chris Welty
  – Based on philosophical foundations
Rigidity

• Consider the following task:
  – *Build an ontology for public transport*
  – "*Passengers can be people and animals."*

• How do you like this solution?
Rigidity

- OntoClean distinguishes *rigid* and *non-rigid* classes
  - If an entity belongs to a rigid class, this holds once and for all
    - i.e.: if the entity does not belong to that class anymore, it ceases to exist
  - This does not hold for non-rigid classes
- Examples for rigid classes
  - Person, mountain, company
- Examples for non-rigid classes
  - Student, stock company, town
  - Caterpillar and butterfly
Rigidity in OntoClean

- OntoClean rule
  - Rigid classes must not be subclasses of non-rigid classes

- Assume that
  - `:peter a :Person .`
  - From that, we conclude that `:peter a :Passenger .`
  - This is probably unwanted
Rigidity in OntoClean

- Improved solution

Diagram:
- Animal
- Person
- Passenger
  - hasRole: Animal
  - hasRole: Person
Rigidity in OntoClean

• Other typical rigidity problems
  – PhysicalObject > Animal
    • An entity may die and thus be no longer an animal
      – If we consider “living” as necessary for animals
    • The physical object (i.e., the body), however, still exists
Identity

• How do we know that two entities are the same
  – Some classes have criteria for identity
    • Immatriculation number of students
    • Tax number for citizens and companies
    • Country codes
    • ...

Identity

• Consider the following task:
  – *Build an ontology for recording working times*
  – "*Time intervals are specific durations. A duration may be 1h, 2h, etc., a time interval may be Monday, 1-2pm, or Tuesday, 3-5pm."*

• How do you like this solution?
Identity

• Let us look at some instances
  – :1h a :Duration . :2h a :Duration . . .

• Obviously, there are more instances of *Interval* than there are instances of *Duration*

• What does that mean?
Identity

• Since the subclass cannot be larger than the superclass, there must be instances that are the same

• Probably, we would expect a mapping such as
  – :Mo10-11 owl:sameAs :1h .
  – :Mo11-12 owl:sameAs :1h .

• From that, we conclude that
  – :Mo10-11 owl:sameAs :Mo11-12 .

• Do we really want that to hold?
Identity

- We have to extend our ontology
- When are two durations the same?
  - If their length is the same
  - \texttt{:1h owl:sameAs :60Min}.

<table>
<thead>
<tr>
<th>Identity</th>
<th>Duration</th>
<th>Interval</th>
<th>Length</th>
<th>Start time</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g., 1h, 2h</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e.g., Monday, 1-2pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Identity

- We have to extend our ontology
- When are two intervals the same?
  - If they have the same length and the same start time
  - :Mo13-14 owl:sameAs :Mo1pm-2pm.
Identity in OntoClean

• Observation:
  – The identity criteria are of the two classes are different

• OntoClean rule:
  – If \( p \) is a subclass of \( q \),
    then \( p \) must not have identity criteria that \( q \) does not have
Identity in OntoClean

- Improved solution:
  - Replace subclass relation by another relation

- e.g., 1h, 2h

- e.g., Monday, 1-2pm

- Duration has duration Interval

- Length

- Start time
Identity in OntoClean

• Other typical problems
  – GeographicalObject > Country
    • Geographical objects and countries have different identity criteria
    • Geographical object: position/polygon
    • Country: government, constitution
    • OntoClean enforces a separation of the geographic and the social construct of a “country”
  – Book > Book edition
    • Book: Title, author
    • Book edition: ISBN, or title and author plus number of the edition
  – Book > Book copy
    • Book: ISBN
    • Book copy: inventory number
Unity

• For some classes, entities can be decomposed into instances of the same class
  – We call them “anti unity classes”

• Examples:
  – An amount of waters into two amount of waters
  – A group into two sub groups

• Other classes only have “whole” instances → “unity classes”
  – e.g., people, cities

• For "whole" individuals, there is always a relation unambiguously relating a part to the whole
  – e.g., relating a body part to a person
Unity

• Assume that we defined

- Amount of Matter
  - Organic Matter
    - Animal
  - Anorganic Matter
    - Plant
Unity

• Let us further assume that we defined*:
  – if we add two amounts of the same type of matter, the result is a larger amount of that type of matter

\[
\text{C} \text{ rdfs:subClassof AmountOfMatter} \\
\land \quad \text{m1 a C . m2 a C . m3 hasPart m1, m2 .} \\
\rightarrow \quad \text{m3 a C .}
\]

*pretending this was possible in OWL, or using rules such as SWRL
• This leads to the following conclusion:

:fluffi a :Animal .
:schnuffi a :Animal .
:SetOfPetersPets hasPart :fluffi, :schnuffi .


• Do we want that?
Unity in OntoClean

- OntoClean rule:
  - Unity classes may only have unity classes as their subclasses
  - Anti unity classes may only have anti unity classes as their subclasses

- In our example:
  - OrganicMatter is an anti unity class
  - Animal is a unity class
Unity in OntoClean

- Solution (again): replace subclass relation by a different relation
Unity in OntoClean

- Such refactorings may hint at missing classes
Summarizing OntoClean

• A number of tests that can be carried out on ontologies
  – Rigidity, Identity, Unity
  – Reveal possible mismodeling issues
  – Avoid nonsensical reasoning consequences
Ontology Design Patterns

• Origin of the term “design pattern”
  – Christopher Alexander (*1936)
  – Buch "A Pattern Language" (1977)

• Architecture
  – Recurring problems
  – Standard solutions
    • With certain degrees of freedom

• Example
  – Problem: rain falls into the building
  – Solution: roof
    • Degrees of freedom: shed roof, saddle roof, hip roof...
Types of Ontology Design Patterns

• Presentation Patterns
  – e.g., naming conventions

• Logical Patterns
  – Domain independent
  – Always specific to a language (e.g., OWL DL)

• Content Patterns
  – Domain dependent
  – Language independent

• Transformation Patterns
  – e.g., how to transform an ontology from one language to the other
Presentation Patterns

• Typical ontology naming conventions
• Use CamelCase
  – CityInNorthernEurope
• Classes start with capital letters, always use singular nouns
  – City, Country
• Properties start with small letters, use a verb, allow unambiguous reading direction
  – isLocatedIn, isCapitalOf
• Instances start with a capital letter
  – Paris, France
• Provide labels for each class, property, and instance
• ...

11/06/15 Heiko Paulheim
Logical Patterns

• Example: ternary relation
• Statement to express: r(X,Y,Z)
• Pattern:

R a owl:Class .
hasR a owl:ObjectProperty .
rComp1 a owl:ObjectProperty .
rComp2 a owl:ObjectProperty .
X hasR [a R ;
hasComp1 Y ;
hasComp2 Z ] .
Content Pattern

- Example: Roles taken at a time
  - e.g.: Gerhard Schröder was the German chancellor from 1998 to 2005

- Competency Question:
  - Who had a certain role at a given time?

- Specializes
  - ternary relation

![Diagram of the Content Pattern]

- Agent → RoleAtTime → Role
  - RoleAtTime → Time Interval
Anti-Patterns

• Things that should not be done
  – But are often done
  – ...and cause some problems

• Possible causes
  – Not thought about each and every consequence
  – Little/wrong understanding of RDF/OWL principles
Anti Pattern: Rampant Classism

• Typical problem:
  – What should be an instance, what should be class?

Writer

Goethe  Schiller  Lessing
Anti Pattern: Rampant Classism

• This is an extreme case...

  :Goethe rdfs:subClassOf :Writer .
  :Faust rdfs:subClassOf :Drama .

• What can we conclude from that?

• Nothing with a DL reasoner, because this is not proper DL!
Anti Pattern: Rampant Classism

• How to distinguish classes and instances
• For every classes, there must be (one or more) instance(s)
  – What should be instances of Goethe?
  – Are there any sentences like “X is a Goethe”?

• Sub class relations must make sense
  – Pattern: “Every X is a Y”
  – “Every Goethe is a Writer”?
Anti Pattern: Exclusivity

- Given the following specification:
  - *Cities bordering an ocean are coastal cities.*

- Modeled in OWL, e.g.
Anti Pattern: Exclusivity

- In OWL:

```
:CoastalCity
  rdfs:subClassOf :City ;
  owl:equivalentClass [
    a owl:Restriction ;
    owl:onProperty :bordering ;
    owl:someValuesFrom :Ocean ] .
```
Anti Pattern: Exclusivity

• Now with instances:

  :Hamburg a :City .
  :AtlanticOcean a :Ocean .
→ :Hamburg a :CoastalCity .

• So far, so good.

  :Germany a :Country .
  :Germany :bordering :AtlanticOcean .
  :AtlanticOcean a :Ocean .
→ :Germany a :CoastalCity .
→ :Germany a :City .
Anti Pattern: Exclusivity

• What is happening here?
  – Ontology was built *exclusively* for a domain
  – e.g., cities
  – Breaks if used in another context

• Recap: Semantic Web Principles
  – AAA (Anybody can say Anything about Anything)
  – i.e., statements should work in different contexts

• Another example:
  – Every person is married to at most one other person
Anti-Patterns: Exclusivity

• Possible Solution:

```owl
:CoastalCity
  owl:intersectionOf
  ( :City
    [ a owl:Restriction ;
      owl:onProperty :bordering ;
      owl:someValuesFrom :Ocean ] ) .
```
Classification of Ontologies

Top Level Ontologies

• Top Level Ontologies
  – Domain independent
  – Task independent
  – Very general

• Goal
  – Reuse
  – Semantic clarity
  – Modeling guidance (i.e., avoid bad modeling)
  – Interoperability
History

Porphyry, Greek philosopher, ca. 234-305

Figure 1.1 Tree of Porphyry, translated from a version by Peter of Spain (1239)
History

• One of the oldest top level ontologies
  – Aristotle (384-322)
• Four basic categories of existence
### Aristotle's Ontological Square

- Example: „white coffee mugs“

<table>
<thead>
<tr>
<th></th>
<th>not substantial</th>
<th>substantial</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal</td>
<td>Category II</td>
<td>Category III</td>
</tr>
<tr>
<td></td>
<td>the color “white”</td>
<td>the category of white coffee mugs</td>
</tr>
<tr>
<td>particular</td>
<td>Category I</td>
<td>Category IV</td>
</tr>
<tr>
<td></td>
<td>the white color of a particular coffee mug</td>
<td>a particular white coffee mug</td>
</tr>
</tbody>
</table>
Basic Categories for Top Level Ontologies

• Abstract vs. concrete entities

• Abstract entities do neither have a temporal nor a spatial dimension
  – Numbers
  – Units of measure

• Concrete entities do at least have a temporal dimension, i.e., a time span at which they exist
  – Things (books, tables, …)
  – Events (lectures, tournaments, …)
Basic Categories for Top Level Ontologies

• 3D vs. 4D view

• 3D view
  – Things extend in space
  – At every point in time, they are completely present

• 4D view
  – Things extend in time and space
  – At a given point in time, they can also be partially present

• Actual vs. possible entities
  – Actualism: only existing entities are included in an ontology
  – Possibilism: all possible entities are included in an ontology
Basic Categories for Top Level Ontologies

• Colocation
  – Can multiple entities exist in the same place?
• This should be easy...
  – 3D view: no
  – 4D view: yes, but not at the same time
• ...but it is not that trivial
  – Example: a statue and the amount of clay from which it was made
    • Do statues even exist?
      – Or is there only clay in the shape of a statue?
      – ...and if both exist, should they belong to the same category?
  – Another example: a hole in a piece of Swiss cheese
    • Do holes even exist?
      – Or are there only perforated objects?
John Sowa's Top Level Ontology

• An “older” top level ontology (1990s)
• Three distinctions form twelve basic categories
  – Physical vs. Abstract
    • Things that exist in time (and potentially in space)
    • Things that do not
  – Continuant vs. Occurent
    • Things that exist as a whole at each point in time
    • Things that partially exist at each point in time
  – Independent vs. Relative vs. Mediating
    • Things that can exist on their own
    • Things that require other things to exist
    • “Third” things that relate two others
John Sowa's Top Level Ontology

- These three distinctions create twelve basic classes of objects
  - All of them are disjoint

<table>
<thead>
<tr>
<th></th>
<th>Physical</th>
<th>Abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Continuant</td>
<td>Occurent</td>
</tr>
<tr>
<td>Independent</td>
<td>Object</td>
<td>Process</td>
</tr>
<tr>
<td>Relative</td>
<td>Juncture</td>
<td>Participation</td>
</tr>
<tr>
<td>Mediating</td>
<td>Structure</td>
<td>Situation</td>
</tr>
</tbody>
</table>

John F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations (1999)
Think Aloud

• Which categories do those entities belong to?
  – The building B6 23-25
  – Today's Semantic Web Technologies Lecture
  – The semester break between HWS2015 and FSS2016
  – Your motivation to be here today
Think Aloud

• Proposal(!) for a solution:
  – The building B6 23-25
    • Physical, Continuant, Independent → Object
  – Today's Semantic Web Technologies Lecture
    • Physical, Occurent, Independent → Process
  – The semester break between HWS2015 and FSS2016
    • Physical, Occurent, Mediating → Situation
  – Your motivation to be here today
    • Abstract, Occurent, Mediating → Purpose
DOLCE

- *Descriptive Ontology for Linguistic and Cognitive Engineering*

- One of the most well known top level ontologies
  - Developed in the EU WonderWeb project (2002-2004)
  - Strong philosophical foundation

- Modular design
  - Basic ontologies: 37 classes, 70 relations
  - All modules: ~120 classes, ~300 relations
Basic Distinctions in DOLCE

- Particulars, universals, and quantities
- Particulars (think: categories): can have instances
  - “City”, “University”
- Universals (think: individuals): cannot have instances
  - "Mannheim", "Mannheim University"
- Qualities: describe an instance
  - e.g., color of a book, height of a person
  - Are neither particulars nor universals
  - Cannot exist without an instance
DOLCE: Basic Assumptions

• A top level ontologies of particulars
  – For both actual and possible entities (possibilistic)
• 4D
  – Some entities may have a temporal dimension
• Colocation
  – Is allowed
  – restriction: not two entities of the same kind at the same spatial and temporal location
    • Not: two statues
    • But: a statue and an amount of clay
Top Hierarchy of DOLCE

- Four pairwise disjoint classes

Endurants vs. Perdurants

• Endurants exist in time
  – Think: things like people, books, ...
    • May also be non-physical: organizations, pieces of information
  – Are always fully present at each point in time during their existence

• Perdurants "happen" in time
  – Think: events and processes
  – Only exist partially at each point in time during their existence
    • i.e., previous and future parts of the perdurant may not (yet|anymore) exist at a given point in time

• Qualities are attached to endurants and perdurants

• Abstracts: numbers, units of measure, etc.
Endurants in DOLCE (1)

Distinguishing Endurants

• Amount of Matter vs. Phyiscal Object
  – Amount of Matter is \textit{mereologically invariant}
  – i.e., a part of an AoM is still an AoM
    • A part of “some water” is still “some water”
    • But a part of a cup is (likely) not a cup
      – cf. unity/anti unity in OntoClean
• Features
  – Cannot exist without a physical endurant
  – e.g., holes, fringes
Endurants in DOLCE (2)

- Physical Endurant
- Arbitrary Sum
- Non-Physical Endurant
- Non-Physical Object
- Mental Object
- Social Object
- Agentive Social Object
- Non-Agentive Social Object
- Social Agent
- Society

Perdurants in DOLCE

Perdurant

Event

Achievement
Accomplishment

Stative

State
Process

Distinguishing Perdurants

• Events vs. Statives
  – The sum of two consecutive statives is a (longer) stative
    • The sum of two times “sitting around” is “sitting around for a longer time”
    • But: the sum of two times “flying to the moon” is not “flying to the moon for a longer time”
Distinguishing Perdurants

• Achievement vs. Accomplishment
  – Achievements non-divisible ("Reaching the border")
  – Accomplishments are divisable ("Going to China")

• State vs. Process
  – States only consist of states of the same type (like “sitting around”)
  – Processes may consist of processes of different types
    – e.g., “studying” consists of “listen to lecture”, “work on project”, “present results”, “write paper”...
Relation of Endurants and Perdurants

• Endurants take part in perdurants
  – Actively (Reader and reading)
  – Passively (Book and reading)
  – DOLCE defines various types of participation

• Endurants only consist of endurants, perdurants only consist of perdurants
  – Books consist of pages, cover, ...
  – Reading consists of perceiving, turning pages, ...
Qualities

• Basic distinction
  – Quality is a property of an entity
  – Quality space is the set of possible values of the quality

• Qualities need entities
  – In general, all particulars can have qualities
  – Qualities only exist as long as the entity exists
Qualities

• Example:
  – Color is a quality
  – RBG is a quality space

• "Two cars have exactly the same color"
  – Every car has got its own quality “color”
  – Both qualities have the same value in the quality space

• Why should each car have its own quality?
  – Qualities only exist as long as the entity they belong to
  – Otherwise, the second car would have no more color once the first car ceases to exist
Other Top Level Ontologies

• SUMO: Suggested Upper Merged Ontology
  – Around 1,000 classes
  – Strong formalization in KIF (Knowledge Interchange Format)

• Cyc: stems from *EnCyClopedi*a
  – Own language (CycL)
  – Top Level and deep general ontology
  – ~250,000 classes
  – OpenCyc: free as OWL and LOD endpoint

• PROTO: PROTo ONtology
  – General top level+ upper level, different domain extensions
  – ~300 classes, ~100 relations
Comparison

- Size: CyC >> SUMO > PROTON > DOLCE
- Level of formalization: SUMO > DOLCE > CyC > PROTON
- Radically different definitions
- Example: time interval
  - In DOLCE: a region (abstract)
  - In SUMO: a quantity (abstract)
  - In PROTON: a happening (~DOLCE:Perdurant)
  - In CyC: e.g., a TemporalThing (~DOLCE:Perdurant)
    and an IntangibleIndividual (~DOLCE:NonPhysicalEndurant)

- Different top level ontologies are, in general, incompatible!
Wrap-Up

• Ontology Engineering: Developing good ontologies
  – Given some utility, e.g., correctness of reasoning
• Methodologies, e.g., Methontology
• OntoClean
  – Systematic debugging of ontologies
• Design Patterns & Anti Patterns
  – Small reusable building blocks
  – Common mistakes to avoid
• Top Level Ontologies
  – Basic categories
  – Help structuring ontologies
Questions?