Semantic Web Ontology Design

Valentina Presutti and Eva Blomqvist
Lecture 3 @ Corso Dottorato 2011
Dipartimento di Scienze dell’Informazione
Bologna, Italy

Computational ontologies

- Ontologies as (software) components, expressed and managed in standard W3C languages like RDF, OWL, RIF, SPARQL
- Ontology design is the core aspect
- Quality is associated with good design
Quality

- Three quality dimensions: Structural-Content-Sustainability
  - *Content* is the primary dimension
- Content compliance spans Coverage-Task-SelfExplanation
  - *Task* is the immediately measurable aspect
  - Quality is not maximal and abstract, but bound to context
  - Partial orders of problems and reusable solutions (locality)
  - Good practices (history)
- Empirical methods for evaluation (measurability)

What is ontology design? 1/2

- Computational Ontologies are artifacts
  - Have a structure (linguistic, logical, etc.)
  - Their function is to “encode” a description of the world (actual, possible, counterfactual, impossible, desired, etc.) for some purpose
What is ontology design? 2/2

• Ontologies must match both domain and task
  – Allow the description of the entities (“domain”) whose attributes and relations are concerned because of some purpose
    • social events and agents as entities that are considered in a legal case
    • research topics as entities that are dealt with by a project, worked on by academic staff, and can be topics of documents
  – Serve a purpose (“task”)
    • finding entities that are considered in a same legal case
    • finding people that work on a same topic
    • matching project topics to staff competencies, time left, available funds, etc.

Standard languages help

• Transform all in RDF, or even OWL
  – Cf. Triplify initiative
  – Datasets extracted from heterogeneous sources, and triplified
  – Semantics depends on intended task of data and relations used for linking
• Then search/visualize RDF data, or make integrating applications
Knowledge search over the semantic web

Ontology Design Patterns

Valentina Presutti and Eva Blomqvist

Lecture 3 @ Corso Dottorato 2011

Dipartimento di Scienze dell’Informazione
Bologna, Italy
Outline

• Motivation for ontology design patterns
• Types of ontology design patterns
• Catalogue of ODPs
• Content ODPs
• Logical ODPs

Two kinds of ontologies

• Coverage-oriented ontologies
  – They cover the terminology/metadata/textual corpora/folksonomies ... that fit a specific domain [big reengineering problem - exploited for annotation, retrieval, etc.]
• Task-oriented ontologies
  – They are able to give a structure to a knowledge base that can be used to answer competency questions [big design and reuse problem - exploited for automated reasoning and querying]
• Currently
  – 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 we can do with OWL

• ... (maybe) we can check the consistency, classify, and query all this knowledge
• this is great, but ...
• ... remember the Scarlet example
  – City subClassOf Country
• Logical consistency is not the main problem
  – e.g. owl:sameAs can be wrongly used and still we have consistency
• Why OWL is not enough?
When to use owl:Individual, owl:Class, owl:ObjectProperty, owl:DatatypeProperty?

- OWL gives us logical language constructs, but does not give us any guidelines on how to use them in order to solve our tasks.
- E.g. modeling something as an individual, a class, or an object property can be quite arbitrary

New problems arising on the Web...

- cf. Semantic Web Interest Group post May 27th, 2008 by Zille Huma:
  “I have been wondering for sometime now that why isn’t it a popular trend to store standard activities of a domain in the ontology and not only the concepts, e.g., for the tourism domain, ontologies normally contain concepts like Tourist, Resort, etc. but I have not so far come across an ontology that also contains the standard activities like searchResort, bookHotel, etc. Why is it so? What support is provided in the ontology languages to model the standard activities of the domain as well?”

  - (1) “searching resorts is a type of functionality required for this kind of services”
    - owl:Class(searchResort) rdfs:subClassOf(Functionality)
  - (2) “a functionality for searching resorts is implemented in our web service”
    - owl:Individual(searchResort) rdf:type(Functionality)
  - (3) “who has been searching for what resorts in our web service?”
    - owl:ObjectProperty(searchResort) rdfs:domain(Customer) rdfs:range(Resort)
  - (4) “how many users have been using our resort searching functionality?”
    - owl:DatatypeProperty(searchResort) rdfs:domain(Customer) rdfs:range(xsd:boolean)
Solutions?

• ... OWL is not enough for building a good ontology, and we cannot ask all web users either to learn logic, or to study ontology design
• Reusable solutions are described here as Ontology Design Patterns, which help reducing arbitrariness without asking for sophisticated skills ...
• ... provided that tools are built for any user 😊

A well-designed ontology ...

• Obeys to “capital questions“:
  – What are we talking about?
  – Why do we want to talk about it?
  – Where to find reusable knowledge?
  – Do we have the resources to maintain it?
• Whats, whys and wheres constitute the Problem Space of an ontology project
• Ontology designers need to find solutions from a Solution Space
• Matching problems to solutions is not trivial
From the lessons learnt ...

• Small ontologies with *explicit* documentation of *design rationales*
  – components supported by specific functionalities
    • selection, matching, composition, etc.
  – implemented in repositories, registries, catalogues, open discussion and evaluation forums, and in new-generation ontology design tools
    • ontologydesignpattern.org
    • ODP and Watson APIs
    • NeOn ODP Plugin
    • etc.

Ontology Design Patterns

*An ontology design pattern is a reusable successful solution to a recurrent modeling problem*
Logical Ontology Design Patterns

Types of ODPs: Structural ODPs
Logical ODPs

• **Definition**

A Logical ODP is a formal expression, whose only parts are expressions from a logical vocabulary e.g., OWL DL, that solves a problem of expressivity

• Logical ODPs are independent from a specific domain of interest
  – i.e. they are content-independent

Logical ODPs

• A Logical ODP describes a formal expression that can be *exemplified, morphed, and instantiated* in order to solve a domain modelling issue

• `owl:Class:_:x rdfs:subClassOf owl:Restriction:_:y`

• *Inflammation rdfs:subClassOf* (localizedIn *some BodyPart*)

• *Colitis rdfs:subClassOf* (localizedIn *some Colon*)

• *John’s_colitis localizedIn John’s_colon*
Logical macros

• Logical macros provide a shortcut to model a recurrent intuitive logical expression
• Example:
  • Formally: R some Thing and R only C
  • Things that R, R at least one thing that is only C
  • Carnivore animals eat only animals + Carnivore animals eat some (at least one) animals

Transformation Ontology Design Patterns
N-ary relation

- Chad Smith was the drum player of Red Hot Chili Peppers when they recorded their album Stadium Arcadium from September 2004 to December 2005.
- A person plays a certain role in a band during an album recording, taking place during a certain time interval

- PlaySituation(Person, MusicianRole, Band, Album, TimeInterval)
Let’s remove the domain

• NaryRelation hasKey[relation1, relation2, relation3, relation4]

Transitive Reduction

• I want to represent that a car is composed of several parts
  – part of – transitive property
• I also want to represent that each part can have “direct” components
  – e.g. the turbine is a component of the engine
• The turbine is a component of the engine, hence it is part of the car, but not its “direct” component
Direct components in a car

Transitive reduction

aProperty rdf:type owl:TransitiveProperty
anotherProperty rdfs:subPropertyOf aProperty

anotherProperty does not inherit transitivity, but:

entity1 anotherProperty entity2
implies
entity1 aProperty entity2
Exhaustive partition

• A class with a number of sub-classes, all disjoint with each other
  – its individuals can belong only to one of its sub-classes
  – it is equivalent to the union of its subclasses
• The “Better Cars” sells only three products: cars, motorcycles and dolls.

Exhaustive partition: meta-level description

• Sibling partition classes are disjoint with each other
• ExhaustivePartitionClass is the union of its PartitionClass
Transformation ODPs

- N-ary relation
  - Allows to represent relations with more than two arguments
- Transitive reduction
  - Allows to create transitive closure over one or more non-transitive property
- Exhaustive partition
  - Allows to represent classes that have a set of disjoint subclasses constituting a complete partition of the class

Types of ODPs
Content ODPs

• CPs encode conceptual, rather than logical design patterns.
  – Logical ODPs solve design problems independently of a particular conceptualization
  – CPs are patterns for solving design problems for the domain classes and properties that populate an ontology, therefore they address content problems
Content ODPs (CPs) 2/3

• CPs are instantiations of Logical ODPs (or of compositions of Logical ODPs), featuring a non-empty signature
  – Hence, they have an explicit non-logical vocabulary for a specific domain of interest, i.e. they are content-dependent

Content ODPs (CPs) 3/3

• Modeling problems solved by CPs have two components: domain and requirements.
  – A same domain can have many requirements (e.g. different scenarios in a clinical information context)
  – A same requirement can be found in different domains (e.g. different domains with a same “expert finding” scenario)
• A typical way of capturing requirements is by means of competency questions
Catalogues of CPs 1/2

- Content ODPs are collected and described in catalogues and comply to a common presentation template.
- The ontologydesignpatterns.org initiative maintains a repository of CPs and a semantic wiki for their description, discussion, evaluation, certification, etc.

Catalogues of CPs 2/2
Pragmatic characteristics of CPs

- Domain-dependent
  - Expressed with a domain-specific (non-logical) vocabulary
- Requirement-covering
  - Solve domain modeling problems (expressible as use-cases, tasks or “competency questions”), at a typical maximum size (cf. blink)
- Reasoning-relevant components
  - Allow some form of inference (minimal axiomatization, e.g. not an isolated class)
- Cognitively-relevant components
  - Catch relevant core notions of a domain and the related expertise -- blink knowledge
- Linguistically-relevant components
  - Are lexically grounded, e.g. they match linguistic frames, or at least a domain terminology
- Examples:
  - PartOf, Participation, Plan, LegalNorm, LegalFact, SalesOrder, ResearchTopic, LegalContract, Inflammation, MedicalGuideline, GeneOntologyTop, Situation, TimeInterval, etc.

Generic ontology requirements

<table>
<thead>
<tr>
<th>Generic Competency Questions</th>
<th>Specific Modeling Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who does what, when, where?</td>
<td>Production reports schedule</td>
</tr>
<tr>
<td>What objects take part in a certain event?</td>
<td>Resource allocation, biochemical pathways</td>
</tr>
<tr>
<td>What are the parts of something?</td>
<td>Component schema, warehouse management</td>
</tr>
<tr>
<td>What's an object made of?</td>
<td>Using and local composition, e.g. for safety pump</td>
</tr>
<tr>
<td>What's the place of something?</td>
<td>Geographic systems, resource allocation</td>
</tr>
<tr>
<td>What's the true value of something?</td>
<td>Dynamic knowledge bases</td>
</tr>
<tr>
<td>What ontological method is being used?</td>
<td>Instructions, enterprise know-how database</td>
</tr>
<tr>
<td>Which technique should be executed in order to achieve a goal?</td>
<td>Planning, workflow management</td>
</tr>
<tr>
<td>How are these objects linked to a certain rule?</td>
<td>Control systems, legal reasoning services</td>
</tr>
<tr>
<td>What is the degree of this artifact?</td>
<td>System description</td>
</tr>
<tr>
<td>How is this object built?</td>
<td>System description, quality trace</td>
</tr>
<tr>
<td>What's the design of this artifact?</td>
<td>Project assistants, catalogues</td>
</tr>
<tr>
<td>How must this artifact behave?</td>
<td>Logistic systems, physical models</td>
</tr>
<tr>
<td>What's your role in the transformation?</td>
<td>Assembly diagrams, planning organisational models</td>
</tr>
<tr>
<td>What information is about how a certain object is?</td>
<td>Information and content modelling, computational models, subject directories</td>
</tr>
<tr>
<td>What argumentation model are you adopting for negotiating an agreement?</td>
<td>Cooperation systems</td>
</tr>
<tr>
<td>What's the degree of confidence you give to this axiom?</td>
<td>Ontology engineering tools</td>
</tr>
</tbody>
</table>
Examples of CPs

Content Pattern
(generic TBox)

Content Pattern
(specific TBox)
Operations

• CP creation and reuse relies on a set of operations:
  – Import
  – specialization
  – composition

Sample Specialization

• A content pattern CP₂ specializes CP₁ if at least one ontology element of CP₂ is subsumed by an ontology element of CP₁
  – i.e., either by rdfs:subClassOf or rdfs:subPropertyOf
Composition

- The composition operation relates two CPs and results into a new ontology

- The resulting ontology is composed of the union of the ontology elements and axioms from the two CPs, plus the axioms (e.g., disjointness, equivalence, etc.) that are added in order to link the CPs

- The composition of CP1 and CP2 consists of creating a semantic association between CP1 and CP2 by adding at least one new axiom, which involves ontology elements from both CP1 and CP2

- Typically, also new elements (“expansion”) are added when composing

Sample composition

The resulting ontology is composed of the union of the ontology elements and axioms from the two CPs, plus the axioms (e.g., disjointness, equivalence, etc.) that are added in order to link the CPs
General Content ODPs

- Classification
  - Roles of objects
- Containment
  - Part-whole relationships
  - Membership
- Information and its realizations
- Time and Places
- Situation
- Description
Roles of objects

• Objects can play different roles in different situations
• Depending on the constraints given by the requirements, modeling of objects and their roles can be addressed differently
• Do we want to represent properties of roles?
• Do we want to classify objects based on their roles?
• Do we want to assert facts about roles?

Roles of objects

• A beer mug used as vase
• Books used as table’s legs
• A sax player (person)
• A song writer (person)
Roles as classes

• An object and its roles are related through the rdf:type property
• rdf:type relations can be either asserted or inferred through classification
• In order to automatically classify individuals in a certain class the ontology has to define appropriate axioms
Roles as classes

- Consequences
  - Low expressivity
  - Roles are described at TBox level
  - Class taxonomy is bigger - a class for each role
  - Class taxonomy is entangled - multi-typing
  - ABox is smaller – same individual, several (role) types
  - Automatic classification of individuals through rdfs:subClassOf inheritance – with proper axioms
  - Roles cannot be indexed in terms of space and time
  - Facts about roles cannot be expressed e.g. “Roles in UniBo can be student, professor, researcher”, “Valentina is teacher for KMDM course”
  - Queries: ?x a SongWriter
- General CQs
  - What objects have a (role) type?

Roles as individuals
Roles as individuals

- An object and its roles are related through domain-specific relations
- Relations between an object and its roles have to be asserted
- Automatic inference of relations between an object and its roles can be obtained through property sub-sumption

Roles as individuals

- Consequences
  - Expressivity is improved
  - Roles are described at ABox level
  - Class taxonomy is smaller – roles are individuals
  - ABox is bigger
  - Facts on roles can be asserted
  - Roles can be indexed in terms of time and space - through n-ary relations
  - N-ary relations are needed for relating an object to its role with respect to some other object e.g. Valentina is teacher for KMDM course
    - kmdm_teacher involvesPerson Valentina
    - kmdm_teacher involvesRole teacher
    - kmdm_teacher involvesCourse KMDM
    - Valentina hasRole teacher
  - Roles do not type objects, no automatic classification of objects
  - Queries: ?x hasRole ?y; ?x a Role
  - General CQs
    - What roles has an object? What objects have a role?
Roles as properties

- The semantics of “having a role” is embedded in the name of a property
- Typically properties conveying a role information are verbs
- Objects are not explicitly related to their roles, they are related to other things through a property expressing an action they perform, a role they play
- Most common pattern in the web of data for modeling roles
Roles as properties

- Consequences
  - Smaller taxonomy of classes
  - Bigger taxonomy of properties – a property for each role
  - Simpler graph of data – one triple for “Valentina is teacher for KMDM course”
    - Valentina teaches KMDM
  - Roles cannot be indexed in terms of space and time
  - Semantics of roles is implicit (embedded in a property name)
  - Facts about roles cannot be expressed
    - Queries: ?x teaches ?y
- General CQs
  - Who did something?

Roles of objects

- The three solutions differ in expressivity, simplicity, and CQs
  - Simplest is roles as properties
  - Most expressive is roles as individuals
  - Least expressive is roles as classes
- Each of them has pros and cons
- The choice depends on requirements
- What about combining them?
Combining roles as instances with roles as classes

• A class Role
• A class for each Role e.g. SaxPlayer
• A property restriction on classes representing roles, for automatic classification

Combining roles as instances with roles as classes

• In this example John_Coltrane is a Person
• He has the role of sax_player
• The property restriction on SaxPlayer allows to classify John_Coltrane as a SaxPlayer
...and add roles as properties

- Note the restriction on property writerOf

Indexing roles in terms of time and space
Indexing roles in terms of time and space

Content ODPs for roles of objects

- **Object-Role**
  - OWL pattern representing roles as individuals
  - [http://ontologydesignpatterns.org/cp/owl/dul/objectrole.owl](http://ontologydesignpatterns.org/cp/owl/dul/objectrole.owl)

- **Classes as roles**
  - Sample pattern (template) representing roles as classes
  - [http://ontologydesignpatterns.org/cp/owl/classesasrole.owl](http://ontologydesignpatterns.org/cp/owl/classesasrole.owl)

- **Time-place-indexed-object-role**
  - N-ary relation representing an objects, the roles it plays at a certain date in a certain place
  - [http://www.ontologydesignpatterns.org/cp/owl/dul/timeplaceindexedobjectrole.owl](http://www.ontologydesignpatterns.org/cp/owl/dul/timeplaceindexedobjectrole.owl)
Object classification

- The object-role relation can be generalized
- An object is classified by a concept
  - Project hasStatus Status
    - IKS hasStatus active
  - Person hasRole Role
    - John_Coltrane hasRole sax_player
  - Object hasType Type
    - MacBookPro hasType laptop
  - Color hasParameter Parameter
    - Black hasParameter positive
  - Document hasTopic Topic
  - Action addresses Task
    - email_to_partners addresses meeting_notification

Object classification

- The discussion on modeling objects and their roles holds for any classification relation
- The general pattern is called “classification”
Parthood

- Objects can have different parts
- Parthood relation is intuitively similar to set inclusion
- The parthood relation is
  
  - Reflexive
    \[ \text{ex:obj1 partOf ex:obj1} \]
  
  - Anti-symmetric
    \[ \text{ex:obj1 partOf ex:obj2} \land \text{ex:obj2 part of ex:obj1} \rightarrow \text{ex:obj1 owl:sameAs ex:obj2} \]
  
  - Transitive
    \[ \text{ex:obj1 partOf ex:obj2} \land \text{ex:obj2 part of ex:obj3} \rightarrow \text{ex:obj1 partOf ex:obj3} \]

Parthood modeling

- Depending on the constraints given by the requirements, modeling of objects and their parts can be addressed differently
- Do we want to distinguish parts of an object at different points in time?
- Do we want to distinguish parts of a whole from its direct parts?
Parthood examples

- The CPU is part of the motherboard
- The motherboard is part of the computer
Parthood examples

- The CPU is direct part of the motherboard
- The motherboard is part of the computer

Part of

- Note that reflexivity and anti-symmetry are not modeled
- Anti-symmetry is not representable in OWL
- Reflexivity would cause the materialization of numerous somehow redundant triples, possibly not desirable in LOD
  - it can be easily added as local axiom
Direct part

- directPartOf does not inherit transitivity
- directPartOf implies part of

Time-indexed parthood

- We need an n-ary relation
- Transitivity for time-indexed parthood is not expressible in OWL
- The range of datatype properties is not set, it depends on local needs
  - Datatypes are all disjoint with each other e.g. xsd:date owl:disjointWith xsd:gYear
Time indexed parthood

• ...and if we want to have time intervals in our domain of discourse?

Time Interval

• hasDate allows us to set any date within the time interval
• The range of the datatype properties is set locally
Membership

- Objects can belong to a collection of objects
- Membership is similar the notion of set membership
- The membership relation is not transitive
  - if an object $O$ belongs to a collection $C$, which in turn belongs to a collection $D$, $O$ is not said to belong to $D$
- The membership relation chained with the parthood relation implies parthood
  - if an object $O$ belongs to a collection $C$, which in turn is part of a collection $D$, $O$ belongs to $D
Membership modeling

- Depending on the constraints given by the requirements, modeling of collections and their members can be addressed differently.
- Do we want to identify members of a collection at a certain point in time?

Membership examples
Membership examples

[Diagram showing membership examples with labels: memberOf and partOf]

5/9/11
Membership examples

Membership

- Note the property chain of memberOf and partOf that implies memberOf
Time indexed membership

• Analogously to parthood, we can have the variant with time intervals

Containment

• Parthood and membership can be generalized as containment relations
• Containment relation represents a general cognitive schema
• A containment schema involves a physical or metaphorical
  – boundary
  – enclosed area or volume, or
  – excluded area or volume
• A containment schema can have additional optional properties, such as
  – transitivity of enclosure (whereby if one object is enclosed by a second, and that by a third, the first is also enclosed by the third)
  – objects inside or outside the boundary
  – protectedness of an enclosed object
  – the restriction of forces inside the enclosure, and
  – the relatively fixed position of an enclosed object
• We model containment as a general transitive relation between objects
Containment

- Time indexed containment and time indexed containment with time intervals are analogous to the previous ones (time indexed membership and parthood with/without time intervals).

Information objects and their realizations

- An information object is a piece of information independently from how it is concretely realized
- Examples of information objects are musical compositions, texts, words, pictures, etc.
- An information realization is a concrete realization of an Information object
- Examples of information realization are the written document containing the text of a law, an mp3 file, etc.
- The distinction between information objects and their realizations is a key requirements in copyright management
Modeling Information Objects

- Depending on the constraints given by the requirements, modeling information objects and their realizations can be addressed differently
- Do we want to temporarily index the realization of an information object?
- Do we want to spatially index the realization of an information object?

Example of information objects and their realizations
Examples of information objects and their realizations

- Pictures of a party are realized in JPG files

Examples of information objects and their realizations

- The realizations of pictures of the party are available on my laptop at the moment I download the attachments to this email
Information Realization

Time and Place indexed information realization
Situation

- A general vocabulary for n-ary relations
- Situation is able to represent reified n-ary relations, by defining a top-level relation for all binary projections of the n-ary relation
- A way somebody conceives a state of affairs, a set of things, a fact
- All time indexed (and place indexed) patterns we have seen so far are (in principle) subclass of Situation
Description

- A Description is represents a conceptualization
- It can be thought also as a 'descriptive context'
- It uses or defines concepts in order to create a view on a 'relational context' (Situation) out of a set of data or observations
- Examples of descriptions are:
  - a Plan of some actions to be executed by agents in a certain way, with certain parameters;
  - a Diagnosis that provides an interpretation for a set of observed entities
Description example

• A situation is a view, consistent with ('satisfying') a description, on a set of entities.
Example of description and situation

- A workflow is a description, its execution is a situation satisfying it