Pattern-based ontology design

Aldo Gangemi, Valentina Presutti

Semantic Technology Lab
ISTC-CNR, Roma
aldo.gangemi@cnr.it
valentina.presutti@istc.cnr.it
Outline

- Computational Ontologies
- Design applied to coverage-oriented ontologies
- Design applied to task-oriented ontologies
- Ontology Design Patterns
Computational ontologies

- Ontologies as (software) components, expressed and managed in standard W3C languages like RDF, OWL, RIF, SPARQL, Fresnel, etc.
- Ontology design is the core aspect
- Quality is associated with good design
  - STLab people study from 2004-5: “A formal framework for ontology evaluation and selection”
  - Three dimensions: Structural-Content-Sustainability
  - Content is the primary dimension
  - Content compliance spans Coverage-Task-SelfExplanation
  - Task is the primary dimension
- 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?

- Computational Ontologies are **artifacts**
  - Have a structure (linguistic, “taxonomical”, 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”) whose attributes and relations are concerned by some purpose
    - e.g. 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 topic of documents, events, etc.
    - Serve a purpose (“task”), e.g. finding elements that are considered in a same case, finding people that work on a same topic, matching project topics to staff competencies, time left, available funds, etc.
- Ontologies have a lifecycle
  - Are created, evaluated, fixed, and exploited just like any artifact
  - Their lifecycle has some original characteristics regarding:
    - Data, Project and Workflow types, Argumentation structures, Design patterns
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 one or more of the dimensions
Design in the C-ODO key

- Watson, Swoogle, Oyster, etc.
- Linking Open Data
- odp-web
- Ontology-related data
- NTK, TopBraid, etc
- Collaborative Ontology Design
- Ontology project execution
- Design solution
- Collaborative procedure
- Argumentation session
- Design action
- Ontology-related data input
- Ontology-related data output
- Collaborative Protégé
- Semantic Wikis
- Biological ODPs on sourceforge
- odp-web
- W3C OEP
- pattern support tools
- evaluation and selection tools
- reengineering tools
- Collaborative Ontology Design Components
- eXtreme Design (XD) - NeOn Tutorial - Grenoble January 2009
Design in the C-ODO key

- Watson, Swoogle, Oyster, etc.
- Linking Open Data
- odp-web
- Ontology-related data (input)
- Ontology project execution
- NTK, TopBraid, etc (output)
- Collaborative Protégé
- Semantic Wikis
- Collaborative procedure
- Argumentation session
- Cicero
- Collaborative Ontology Design Components

- Biological ODPs on sourceforge
- odp-web
- W3C OEP

- pattern support tools
- evaluation and selection tools
- reengineering tools

- eXtreme Design (XD) - NeOn Tutorial - Grenoble January 2009

- Linking Open Data
- Evaluation and selection tools
- Reengineering tools
- Collaborative Ontology Design Components
Outline

- Computational Ontologies
- Design applied to coverage-oriented ontologies
- Design applied to task-oriented ontologies
- Ontology Design Patterns
Ontology-related data

- Informal vs. formal
  - Text corpora
  - Folksonomies (tag sets, directories, topic trees, subject indexes, infoboxes)
  - Lexica (dictionaries, wordnets, terminologies, nomenclatures)
  - Knowledge organization systems (thesauri, classification schemes)
  - Frames, semantic networks
  - DB schemas
  - Linked Open Data datasets
  - (Computational) ontologies
- Suppose we need to design a web ontology of desire: where to start from?
Plena mujer, manzana carnal, luna caliente,
espeso aroma de algas, lodo y luz machacados,
quién oscura claridad se abre entre tus columnas?
¿Qué antigua noche el hombre toca con sus sentidos?

Ay, amar es un viaje con agua y con estrellas,
con aire ahogado y bruscas tempestades de harina:
amar es un combate de relámpagos
y dos cuerpos por una sola miel derrotados.

Beso a beso recorro tu pequeño infinito,
tus márgenes, tus ríos, tus pueblos diminutos,
y el fuego genital transformado en delicia
corre por los delgados caminos de la sangre
hasta precipitarse como un clavel nocturno,
hasta ser y no ser sino un rayo en la sombra.

[Pablo Neruda]
Tags: Flickr
Linguistic dictionaries and thesauri

- Oxford American Dictionary
  - desire [dəˈzɪə(r)]
  - noun
  - a strong feeling of wanting to have something or wishing for something to happen: [with infinitive] a desire to work in the dirt with your bare hands.
  - strong sexual feeling or appetite: they were clinging together in fierce mutual desire.
  - verb [trans.]
  - strongly wish for or want (something): he never achieved the status he so desired | [as adj.] (desired) it failed to create the desired effect.
  - want (someone) sexually: there had been a time, years ago, when he had desired her.
  - archaic express a wish to (someone); request or entreat.
  - ORIGIN Middle English: from Old French desir (noun), desirer (verb), from Latin desiderare (see desiderate).

- Thesaurus
  - desire
  - noun
  - a desire to see the world wish, want, aspiration, fancy, inclination, impulse; yearning, longing, craving, hankering, hunger; eagerness, enthusiasm, determination; informal yen, itch, jones.
  - verb
  - 1 they desired peace want, wish for, long for, yearn for, crave, hanker after, be desperate for, be bent on, covet, aspire to; fancy; informal have a yen for, have a Jones for, yen for, hanker after/for.
  - 2 she desired him be attracted to, lust after, burn for, be infatuated by; informal fancy, have the hots for, have a crush on, be mad about, be crazy about.
WordNets
FrameNets

Frame Report (recent data)

Desiring

Definition:
An **Experience** desires that an **Event** occur. (Note that commonly a resultant state of the **Event** will stand in for the **Event**.) In some cases, the **Experience** is an active participant in the **Event**, and in such cases the **Event** itself is often not mentioned, but rather some **local participant** which is subordinately involved in the **Event**.

Generally, the use of a word in this frame implies that the specific **Event** has not yet happened, but that the **Experience** believes that they would be happier if it did. Sometimes the **Time_of_Event**, **Purpose_of_Event**, or the **Location_of_Event** are mentioned without the explicit mention of the **Event**.

I only **WANTED** one piece of candy.
The company was **ZAGS** for him to leave as soon as possible.
Susan **WISHES** that you’d listen to her.

FEs:

Core:

- **Events**
- **Semantic Type**: State_of_affairs

- **Experience**
- **Semantic Type**: Sentient

- **Local Participant**
- **Semantic Type**: Patient

- **Location of Event**
- **Semantic Type**: Place

The change that the **Experience** would like to see.

The **Experience** is the person (or sentient being) who wishes for the **Event** to occur.

This is the entity that the **Experience** wishes to be affected by some **Event**.

The **Location of Event** is the place involved in the desired **Event**.

The prince **WISHES** you **here** before matins.
AGROVOC is a multilingual, structured and controlled vocabulary designed to cover the terminology of all subject fields in agriculture, forestry, fisheries, food and related domains (e.g., environment).

Learn more about AGROVOC by browsing: AGROVOC Flyer
Desire (emotion)

Desire is a sense of longing for a person or object or hoping for an outcome. The same sense is expressed by emotions such as longing, which is evoked by the enjoyment in the thought of the item or person, and they want to take actions to obtain their goal. It is assumed that human desire is the fundamental motivation of all human actions.

The theme of desire is at the core of the roman novel, which often create dramas by showing desire where human desire is in. People ranging from U.S. Nada to T.S. Eliot have dealt with the which use plots that appeal to the heightened emotions of the audience by showing forces of human emotion. (not mentioned)

In philosophy

Main Article: Desire (philosophy)

In philosophy, desire has been studied as a philosophical problem since antiquity. In Plato’s The Republic, he argues that the interchange and the propriety of animals to motion. At the same time, he acknowledges that reasoning also induces with desire. In art, desire is a primary sense for pleasure. In Spinoza (1632 – 1677), a Dutch philosopher, had a view which contrasts with Hellenic in the philosopher, economist, and historian, claimed that desires and passions are non-rational, automatic, instinctive, and reflective, and he proposed that esteem is ethics.

Kant (1724 – 1804), a German philosopher, called any action based on desires a hypothetical imperative, meaning by this that the desire is a purpose in Critique of Judgment. Hegel claimed that “self-consciousness is desire.”

Because desire can cause humans to become obsessed and ambitious, it has been called one of the causes of man for marketers. Within the teachings of Buddhism, craving is thought to be the cause of all suffering that one experiences in human existence. The eradication of craving leads one to ultimate happiness, or Nirvana. Desire for wholesome things, though, is liberating and encouraging. While the stream of desire for sense-pleasures must be cut eventually, a practitioner on the path to liberation is encouraged by the Buddha to “generate desire” for the fostering of skillful qualities and the abandoning of unskillful ones.
Yago

YAGO-query:

<table>
<thead>
<tr>
<th>?id0:</th>
<th>desire</th>
<th>?x</th>
</tr>
</thead>
<tbody>
<tr>
<td>?id1:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?id2:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Submit Query

?desire = desire
?what0 = subClassOf
?x = tendency

?desire = desire
?what0 = subClassOf
?x = feeling

?desire = desire
?what0 = subClassOf
?x = arousal
Linking Open Data
A lot of data in the web “suq”

- Mash-ups (making apps)
- Linked open data (mapping)
- Wikipedia, DBpedia, Freebase, etc. (modeling, reengineering, mapping)
- Triplify, GRDDL, RDFa, SKOS, SIOC, etc. (reengineering)
- Semantic Interoperability (making apps, reengineering, formalizing)
- Corpora, terminologies, lexica, thesauri, “KOS”, frames, ontologies (modeling)
Standard languages help

- Transform all in RDF, or even OWL
  - Cf. Triplify initiative
- Dataset extracted from heterogeneous sources, and triplified
- Relations are added in direct, naïve ways: Linked Open Data
  - Semantics depends on intended task of data and relations used for linking
- Then search/visualize RDF data, or make integrating applications
Search results for term “desire”, found about 1

Is This Desire? (RDF)
[2008-12-10 - 116 triples in 19.4 kb]
http://dbpedia.org/resource/Is_This_Desire%3F (Search) (Cached) (Onto)

desire (RDF)
[2008-11-11 - 7 triples in 1 kb]
http://wordnet.rubexplore.com/id/synset-desire-verb-3 (Search) (Cached) (Onto)

More...

2- http://www.nuig.org/ontology/ks
[2008-11-11 - 8 triples in 1.3 kb]
http://wordnet.rubexplore.com/id/word-desire (Search) (Cached) (Onto)

desire (RDF)
[2008-11-11 - 7 triples in 1.1 kb]
http://wordnet.rubexplore.com/id/wordsense-desire-noun-1 (Search) (Cached) (Onto)

More...


3- http://mogatu.umbc.edu/ont/2004/01/BDI.owl#MrBDI [2]


Integrated knowledge search: DBpedia
Integrated knowledge search: Freebase
Now we have all those data expressed in a language that allows semantic interoperability ...
What we can do with OWL

- ... (maybe) we can check the consistency, classify, and query all this knowledge
- this is great, but ...
- ... when I locally reuse parts of such a big bunch of knowledge, inferences sometimes produce strange results:
  - a web page same as an email address (e.g. http://.../Aldo owl:sameAs mailto://aldo@...)
  - a person same as a wikipedia article (e.g. Aldo owl:sameAs http://en.wikipedia.org/Aldo)
  - Italy is a continent (e.g. (Italy rdf:type (Country) rdfs:subClassOf Continent))
  - ...
- ... and problems are hardly fixable on a large scale
- 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, Class, ObjectProperty, DatatypeProperty? New problems arising on the Web

- 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
  - 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) a functionality for searching resorts is implemented in our web service
    - owl:Individual(searchResort) rdf:type(Functionality)
  - (2) searching resorts is a type of functionality required for this kind of services
    - owl:Class(searchResort) rdfs:subClassOf(Functionality)
  - (3) who has been searching for what resorts in our web service?
    - owl:ObjectProperty(searchResort) rdfs:range(Resort)
  - (4) how many users have been using our resort searching functionality?
    - owl:DatatypeProperty(searchResort) 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 as **Ontology Design Patterns**, which help reducing arbitrariness without asking for sophisticated skills ...
- ... provided that tools are built for any user :)
Outline

- Computational Ontologies
- Design applied to coverage-oriented ontologies
- Design applied to task-oriented ontologies
- Ontology Design Patterns
An ontology designer’s world

- Requirements (I want to attend my ideal talk)
- Logical constructs (subClassOf, restriction, ...)
- Existing ontologies (FOAF, BibTex, SWC, DOLCE, ...)
- Informal knowledge resources (CiteSeer, ACM topic catalog)
- Conventions and practices (naming/URI making, disjoint covering, reification patterns, transitive partOf, role-task, ...)
- Tools: editors, reasoners, translators, etc. (Protégé, NeOn Toolkit, TBC, FaCT++, Pellet, SMW, Jena, AllegroGraph, Virtuoso, ...)
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 ...

- We envision small ontologies with explicit documentation of design rationales, and best reengineering practices
  - 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.
Outline

- Computational Ontologies
- Design applied to coverage-oriented ontologies
- Design applied to task-oriented ontologies
- Ontology Design Patterns
Ontology Design Pattern

- An ontology design pattern is a reusable solution to a recurrent modeling problem
Pattern-based design (OP)

- Pattern-based ontology design is the activity of searching, selecting, and composing different patterns
  - Logical, Reasoning, Architectural, Naming, Correspondence, Reengineering, Content
  - Common framework to understand modeling choices (the "solution space") wrt task- and domain-oriented requirements (the "problem space")
  - http://www.ontologydesignpatterns.org
We also distinguish between ontological resources that are not OPs and Ontology Design Anti-Patterns (AntiOPs)
Presentation OPs

Definition

- Presentation OPs deal with usability and readability of ontologies from a user perspective.

- They are meant as good practices that support the reuse of patterns by facilitating their evaluation and selection.

- Two types:
  - Naming OPs
  - Annotation OPs
Definition

Naming OPs are conventions on how to create names for namespaces, files, and ontology elements in general (classes, properties, etc.).

Naming OPs are good practices that boost ontology readability and understanding by humans, by supporting homogeneity in naming procedures.
Namespace declared for ontologies.

It is recommended to use the base URI of the organization that publishes the ontology

followed by a reference directory for the ontologies
  - e.g. http://www.loa-cnr.it/ontologies/

It is also important to choose an approach for encoding versioning, either on the name, or on the reference directory.
Class names

They should not contain plurals, unless explicitly required by the context
- Names like Areas is considered bad practice, if e.g. an instance of the class Areas is a single area, not a collection of areas

It is also recommended to use readable names instead of e.g. alphanumerical codes
- Non-readable name can be used (even if not recommended) if associated to proper annotations (see Annotation OPs)

It is useful to include the name of the parent class as a suffix of the class name
- e.g. MarineArea rdfs:subClassOf Area

Class names conventionally start with a capital letter
- e.g. Area instead of area
Annotation OPs provide annotation properties or annotation property schemas that are meant to improve the understandability of ontologies and their elements.
Examples of Annotation OPs

- RDF Schema labels and comments (crucial for manual selection and evaluation)
  - Each class and property should be annotated with meaningful labels
    - i.e., by means of the annotation property rdfs:label, with also translations in different languages.
  - Each ontology and ontology element should be annotated with the rationale they are based on
    - i.e., by means of the annotation property rdfs:comment
Reasoning OPs

Definition

- Reasoning OPs are applications of Logical OPs oriented to obtain certain reasoning results, based on the behavior implemented in a reasoning engine.
Examples of Reasoning OPs 1/2

- Precise
  - Classification
  - Subsumption
  - Inheritance
  - Materialization
  - De-anonymizing
  - Normalization

- Approximate
  - Approximate classification
  - Similarity induction
  - Taxonomy induction
  - Relevance detection
  - Latent semantic indexing
  - Automatic alignment
Correspondence OPs

Definition

› Correspondence OPs include Reengineering OPs and Mapping OPs.

› Reengineering OPs provide designers with solutions to the problem of **transforming** a conceptual model, which can even be a non-ontological resource, into a new ontology.

› Mapping OPs are patterns for **creating semantic associations** between two existing ontologies.
Definition

- Reengineering OPs are transformation rules applied in order to create a new ontology (target model) starting from elements of a source model

- The target model is an ontology, while the source model can be either an ontology, or a non-ontological resource
  - e.g., a thesaurus concept, a data model pattern, a UML model, a linguistic structure, etc.

- Two types:
  - Schema reengineering OPs are rules for transforming a non-OWL DL metamodel into an OWL DL ontology
  - Refactoring OPs provide designers with rules for transforming, i.e. “refactoring”, an existing OWL DL “source” ontology into a new OWL DL “target” ontology
    - E.g. a guideline to reengineer a piece of an OWL ontology in presence of a requirement change, as when moving from individuals to classes, or from object properties to classes. See also N-ary relation tranformation pattern
Schema Reengineering OP example: kos2skosABox

KOS $\mapsto \text{skos:ConceptSchema}$ \hspace{1cm} (2.1)
Descriptor $\mapsto \text{skos:Concept}$ \hspace{1cm} (2.2)
Broader Term $\mapsto \text{skos:broader}$ \hspace{1cm} (2.3)
Related Term $\mapsto \text{skos:related}$ \hspace{1cm} (2.4)

- The rule (2.1) states that, given a KOS, it maps to an instance of the class skos:ConceptSchema.
- The rule (2.2) maps each “Descriptor” from a KOS to a specific instance of the class skos:Concept.
- The rule (2.3) relates to the case of having two “Descriptors” $d_1$ and $d_2$ in a KOS, where $d_1$ has “Broader Term” $d_2$. Given the corresponding instances of skos:Concept $\text{skos:c1}$ and skos:c2, the broader term relationship between $d_1$ and $d_2$ maps to an object property value having the subject $\text{skos:c1}$, the object property $\text{skos:broader}$, and the object $\text{skos:c2}$.
- The rule (2.4) relates to the case of having two “Descriptors” $d_1$ and $d_2$ in the KOS that are “Related Terms”. Given the corresponding instances of skos:Concept $\text{skos:c1}$ and skos:c2, the related term relationship between $d_1$ and $d_2$ maps to a (symmetric) object property value having the subject $\text{skos:c1}$, the object property $\text{skos:related}$, and the object $\text{skos:c2}$. 

STaLAB
Semantic Technology Laboratory

eXtreme Design (XD) - NeOn Tutorial - Grenoble January 2009
Mapping OPs

Definition

- Mapping OPs refer to the semantic relations between mappable elements:
  - equivalent to, (not equivalent to)
    - foaf:Agent ≡ wn16:Agent-3
  - contained in, (not contained in)
    - foaf:Person ⊑ geo:SpatialThing
  - overlap with
    - foaf:Person ∩ dul:Person
  - disjoint with
    - (dul:PhysicalPerson ∩ dul:SocialPerson) = ∅
- Also called “correspondence patterns” in [16]
- We also consider an additional semantic relation that we call cloned from
  - ontology element oe₁ in one ontology is the clone of an ontology element oe₂ in another ontology
  - this relation is put in place when extracting a Content Ontology Design Pattern (see later)
Structural OPs

- Structural OPs includes Logical OPs and Architectural OPs.

- Architectural OPs affect the overall shape of the ontology either internally or externally.
  - i.e., an internal Architectural OP identifies a composition of Logical OPs that are to be exclusively used in the design of an ontology.

- Logical OPs are compositions of logical constructs that solve a problem of expressivity.
Definition

› Architectural OPs affect the overall shape of the ontology: their aim is to constrain ‘how the ontology should look like’

› Architectural OPs emerged as design choices motivated by specific needs
  ‣ e.g., computational complexity constraints.

› They are useful as reference documentation for those initially approaching the design of an ontology
Architectural OPs 2/2

- Architectural OPs can be of two types: *internal APs* and *external APs*
- Internal APs are defined in terms of collections of Logical OPs that have to be exclusively employed when designing an ontology
  - e.g., an OWL species, or the varieties of description logics: http://www.cs.man.ac.uk/~ezolin/dl/
- External APs are defined in terms of meta-level constructs
  - e.g., the modular architecture consists of an ontology network, where the involved ontologies play the role of modules. The modules are connected by the import operation.
Logical OPs

Definition

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

- Logical OPs are independent from a specific domain of interest
  - i.e. they are content-independent

- Logical OPs depend on the expressivity of the logical formalism that is used for representation
  - They help to solve design problems where the primitives of the representation language do not directly support certain logical constructs

- They can be of two types: logical macros, and transformation patterns
Logical macros

- Logical macros provide a shortcut to model a recurrent intuitive logical expression

Example:

- the macro: ∀R.C [7]
- colloquially means “every R must be a C”
- formally: ∃R.⊤ ∩ ∀R.C

In OWL:

- the combination of an owl:allValuesFrom restriction with an owl:someValuesFrom restriction.
Transformation patterns

Definition

Transformation patterns translate a logical expression from a logical language into another, which approximates the semantics of the first, in order to find a trade-off between requirements and expressivity.

We describe transformation patterns by two diagrams at different levels:

- The first diagram shows the meta model elements needed for representing the pattern in OWL DL. Such elements are defined in http://www.loa-cnr.it/codeps/owl/owl10a.owl, an OWL ontology that encodes OWL DL constructs in a metamodel. The ontology is referred to by the prefix “a:”.

- The second diagram shows an example of usage for the Logical OP.
Examples of Transformation patterns: N-ary relation (1/2)
Examples of Transformation pattern: N-ary relation (2/2)

But beware of identification constraints! [15]
Content Ontology Design Patterns
CPs: some theory
Content OPs (CPs)

- CPs encode conceptual, rather than logical design patterns.
  - Logical OPs 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

- CPs are instantiations of Logical OPs (or of compositions of Logical OPs), 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

- 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 [11]
Peter Clark’s idea

- A pattern is a theory template. It denotes a structure that is invariant under signature transformation (morphism). Pattern validity in an application is then left to a subjective decision.
  - E.g. the axiom:
  - [If a consumer is connected to a producer, then it is supplied]
  - \( \forall c ((\text{consumer}(c) \land \exists p (\text{producer}(p) \land \text{connects}(c,p))) \rightarrow \text{supplied}(c)) \)
  - via signature morphism becomes e.g. in an application:
  - [If a light is connected to a battery, then it is powered]
  - \( \forall c ((\text{light}(c) \land \exists p (\text{battery}(p) \land \text{connects}(c,p))) \rightarrow \text{powered}(c)) \)
  - But if a pattern is just an untyped structure, there are no ways to distinguish a Logical OP vs. a CP
CPs vs. Logical OPs

- \( \forall c \((\text{consumer}(c) \land \exists p(\text{producer}(p) \land \text{connects}(c,p))) \rightarrow \text{supplied}(c)) \)

- SubClassOf
  - ((intersectionOf
    - Consumer
      - (restriction(\text{connects someValuesFrom(Producer)})))
    - Supplied)

- \( \forall c (((\varphi(c) \land \exists p(\psi(p) \land \rho(c,p))) \rightarrow \chi(c)) \)

- SubClassOf
  - ((intersectionOf
    - \text{owl:Class}:\varphi
      - (restriction(\text{owl:ObjectProperty}:\rho \text{ someValuesFrom(owl:Class}:\psi)))
    - \text{owl:Class}:\chi)

- In OWL, this is a GCI (General Concept Inclusion) axiom. Not a typical LP
Formal characteristics of OWL CPs

- (Small) ontology morphing
  - “being a part of something at some time”
- Downward subsumption of at least one element
  - “being a component of a system at some time”
- Only rarely GCI axioms like in Clark’s example
- Mostly graphs of classes and properties that are self-connected through axioms (subClassOf, equivalentClass, domain, range, disjointFrom)
  - ObjectProperty(component domain(System))
- Usually there is an underlying n-ary relation (sometimes polymorphic)
  - component(s,e,t) → System(s) ∧ Entity(e) ∧ Time(t)
  - ? component(s,e,t,...) → System(s) ∧ Entity(e) ∧ Time(t) ∧ ...(...)
Characteristics of CPs

- Requirement-covering components
  - They are defined in terms of the requirements (or cqs) they satisfy
- Computational components.
  - CPs are language-independent, and should be encoded in a high-order representation language.
  - Nevertheless, their (sample) representation in a computational logic that can be processed by parsers and automatic reasoners is needed in order to (re)use them as building blocks in ontology design.
- Small, autonomous components.
  - A CP is a small, autonomous ontology and ensures a certain set of inferences to be enabled on its corresponding knowledge base.
  - Smallness and autonomy of CPs facilitate ontology designers: composing CPs enables them to govern the complexity of the whole ontology.
  - CPs require a critical size, so that their diagrammatical visualizations are aesthetically acceptable and easily memorizable.
Hierarchical components.
- A CP can be an element in a partial order, where the ordering relation requires that at least one of the classes or properties in the pattern is specialized.
- A hierarchy of CPs can be built by specializing or generalizing some of the elements (either classes or relations).
- For example, the agent-role pattern can be specialized to the person-taking-a-musician-role pattern.

Cognitively relevant components.
- CP visualization must be intuitive and compact, and should catch relevant, “core” notions of a domain.
- An interesting result from cognitive learning is that the development of expert skills typically “selects” patterns of concepts that are richly interconnected, and in normal cases, these patterns are applied without an explicit reference to the underlying detailed knowledge acquired during the training period.
- A CP must contain the central notions that “make rational thinking move” for an expert in a given domain for a given task.
Characteristics of CPs

- **Reasoning-relevant components**
  - They allow some form of inference (minimal axiomatization, e.g. not an isolated class)

- **Linguistically relevant components.**
  - Many CPs nicely match linguistic patterns called frames.
  - A frame can be described as a lexically founded ontology design pattern.
    - Frames typically encode argument structures for verbs, e.g. the frame Desiring associates elements (or “semantic roles”) such as Experiencer, Event, FocalParticipant, LocationOfEvent, etc. The richest repository of frames is FrameNet.
  - Frames can be used for validating CPs with respect to lexical coverage, for lexicalizing them, and can be reengineered in order to populate the CP catalogue

- **Best practice components.**
  - A CP should be used to describe a “best practice” of modelling.
  - Best practices are intended as local, thus derived from experts.
  - The quality of CPs is currently based on the personal experience and taste of the proposers, or on the provenance of the knowledge resource where the pattern comes from.
A catalogue of CPs
- http://www.ontologydesignpatterns.org (odp-web)
- catalogue entry
Annotation properties:
- http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl
- annotation of OWL implementation of CPs
Example 1: AgentRole

The AgentRole Content OP locally defines the following ontology elements:

- **Agent** (owl:Class)

Any agentive Object, either physical or social.

- **Agent page**

Reviews about AgentRole

The are no reviews.

Go back to the List of Content OP proposals

The time indexed person role CP allows to represent temporariness of roles played by persons. It can be generalized for including objects or, alternatively the n-ary classification CP can be specialized in order to obtain the same expressivity.

The elements of this Content OP are added with the elements of its components and/or the elements of the Content OPs it is a specialization of.
Agent Role Instantiation
Example 2: Time Interval

**TimeInterval**

- **Submitted by**: Valentina Presutti
- **Name**: time interval
- **Also Known As**:
- **Intent**: To represent time intervals.
- **Domains**: Time
- **Consequences**: The dates of the time interval are not part of the domain of discourse, they are datatype values. If there is the need of reasoning about dates the Content OP should be used in composition with the `region` Content OP.
- **Scenarios**: The time interval “January 2008” starts at 2008-01-01 and ends at and ends at 2008-01-31.

**Elements**

- **Time interval** (owl:Class)
  Any region in a dimensional space that represents time.
  - **TimeInterval page**

- **hasIntervalDate** (owl:DatatypeProperty)
  A datatype property that encodes values from xsd:date for a time interval; a same time interval can have more than one xsd:date value: begin date, end date, date at which the interval holds, as well as dates expressed in different formats: xsd:Year, xsd:dateTime, etc.
  - **hasIntervalDate page**

- **hasIntervalStartDate** (owl:DatatypeProperty)
  The start date of a time interval.
  - **hasIntervalStartDate page**

- **hasIntervalEndDate** (owl:DatatypeProperty)
  The end date of a time interval.
  - **hasIntervalEndDate page**

**Reusable OWL**

- [http://www.ontologydesignpatterns.org/cp/owl/timeinterval.owl](http://www.ontologydesignpatterns.org/cp/owl/timeinterval.owl)

**Building Block**

- [files](http://www.ontologydesignpatterns.org/cp/examples/timeinterval/january2008.owl)

**Known Uses**

- Web
- References
- Other
- References

**Examples (OWL)**


**Extracted From**

- Reengineered From
- Has Components
- Specialization Of
- Related CPs
Example 3: Part

This also uses transitivity reasoning pattern

Example 4: Time-indexed Participation

This also uses N-ary logical pattern
Covering

- The covering property expresses the fact that a CP satisfies a set $CQ$ of competency questions $(cq_1, ..., cq_n)$.
  
  $cov(CP, CQ)$

- A $cq_i$ can be transformed into a query $q_i$ that is submitted to a knowledge base.

- A CP covers $CQ$ if it is as expressive as it is needed to store the necessary knowledge for answering $q_1, \ldots, q_n$. 
(Re)use situations:
matching CPs covering against local problems
Representing local problems

- Local problems can be expressed in different ways:
  - use cases, scenarios, user requirements, local competency questions (cqs), etc.
- Following [11] all can be transformed to local “cqs”:
  - Red Hot Chili Peppers recorded the Stadium Arcadium album during 2005
  - When did Red Hot Chili Peppers record the Stadium Arcadium album?
  - Which albums did Red Hot Chili Peppers record during 2005?
  - ...
- Local “cqs” are not usually at the same level of generality as the cqs of CPs
  - e.g., they may contain reference to instance element e.g. Stadium Arcadium
  - we need to abstract them
  - When did a certain band record a certain album?
  - Which albums did a certain band record during a certain time period?
  - ...
What we mean by matching cqs to CPs

- What do we mean by matching a cq to CPs?
  - To compare the local cqs to the cqs covered by a CP in order to evaluate the CP suitability for solving the local problems.
  - There is not yet automatic support for this task, hence it is performed as a human task (in this tutorial).
  - Ongoing work on automatic support for CP selection starting from local cqs
    - parsing of requirements and extraction of cqs
    - formalization of cqs
    - NLP support to match cqs terminology to CP lexicalizations
    - case-based reasoning [13]
    - ontology matching (tomorrow’s tutorial)
    - ...
Summary of reuse situations and examples

- Precise or redundant matching
- Broader or narrower matching
- Partial matching
Operations

- CP creation and reuse relies on a set of operations:
  - import
  - specialization
  - generalization (converse of specialization)
  - composition
  - expansion
  - clone
Import

- Import is the basic mechanism for ontology reuse.

- It is also the only one directly supported in the OWL vocabulary
  - i.e., owl:import

- Import is applicable to ontologies, hence also to CPs

- If an ontology $O_2$ imports an ontology $O_1$, all the ontology elements and OWL axioms from $O_1$ are included in $O_2$

- The imported ontology elements and axioms cannot be modified
  - i.e., the ontology elements and axioms are read-only entities for $O_2$

- By importing a CP, an ontology ensures the set of inferences allowed by the CP in its corresponding knowledge base
Sample import
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
Specialization and Generalization of CPs

- Specialization introduces a partial order between CPs, which is defined in terms of their taxonomical order.
- The subsumption relation between ontology elements of two CPs determines which of the two CPs is more or less general than the other one.
- Specialization and generalization only rely on `rdf:subClassOf`, and `owl:subPropertyOf` OWL axioms.
- If two CPs have specialized elements in both directions, neither of the two cases apply. Maintaining a partial order when adapting CPs is anyway a good practice.
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.
Where do CPs come from?

- Content ontology design patterns (CPs) come from the experience of ontology engineers in modeling foundational, core, or domain ontologies.

- There are four ways of creating CPs, which can be summarized as follows:
  - Reengineering from patterns expressed in other data models
    - Data model patterns, Lexical Frames, Workflow patterns, Knowledge discovery patterns, etc.
  - Specialization/Generalization/Composition of other CPs
  - Extraction from reference ontologies (by cloning)
  - Creation by combining extraction, specialization, generalization, composition, and expansion.
Clone

- The extraction process relies on the clone operation
- The clone operation consists of duplicating an ontology element, which is used as a prototype
Types of clone operation

- Shallow clone:
  - consists of creating a new ontology element $oe_2$ by duplicating an existing ontology element $oe_1$. OWL restrictions of and axioms defined for $oe_1$ and $oe_2$ will be exactly the same.

- Deep clone:
  - consists of creating a new ontology element $oe_2$ by duplicating an existing ontology element $oe_1$, and by deep-cloning a new ontology element for each one that is referred in $oe_1$’s axiomatization, recursively.

- Partial clone:
  - consists of deep-cloning an ontology element, but by keeping only a subset of its axioms, and of partial-cloning the kept elements, recursively.

- Some ontology design tools support the shallow clone operation
  - e.g., TopBraid Composer

- Deep clone and partial clone are not yet supported by any existing tool.
The extraction process

1. Create a new OWL ontology and assign a suitable namespace
2. Import source ontologies (locked)
3. Partial clone of classes and properties
4. Update restrictions with local elements
5. Create sub-classes and sub-properties (Specialization)
6. Run the reasoner for inferences
7. Run the reasoner for consistency checking
8. Check disjointness
9. Add classes, properties, and axioms (Expansion)
10. Remove external references
11. Save desired inferences
12. Remove imports
13. Annotate the CP and the CP elements
CP definition (finally!)

Definition

- CPs are distinguished networked ontologies and have their own namespace.
- They cover a specific set \( CQ \) of competency questions (requirements), which represent the problem they provide a solution for.
- A CP emerges from existing conceptual models and can be extracted from a reference ontology (based on the clone operation), can be reengineered from other conceptual models (e.g. data models), can be created by composition of other CPs, by expansion of a CP, and either by specialization or generalization of another CP.
- A CP is associated with two sets, which are both unique:
  - the set of its ontology elements, and the set of its OWL axioms.
- CPs instantiate Logical OPs, or some composition of them.
- Furthermore, CPs show a set of pragmatic characteristics.
Pattern-based ontology design method: eXtreme ontology Design (XD)

- Inspired by eXtreme Programming basic rules
  - e.g., pair programming, test-oriented, continued integration, etc.
- Main principles
  - divide & conquer
    - understand the task and express it by means of competency questions
  - re-use “good” solutions i.e., ontology design patterns
  - evaluate the result against the task
- As an example, I will apply an XD iteration with CPs
Sample XD iteration

- Sentence: Charlie Parker is the alto sax player on Lover Man, Dial, 1946
  - Charlie Parker (person)
  - the alto sax player (player role)
  - on Lover Man (tune)
  - Dial (publisher)
  - 1946 (recording year)

- CQs
  - what persons play a musical instrument?
  - on what tune?
  - for what publisher?
  - in what recording year?

- Queries
  - SELECT ?z ?w WHERE { ?z ?t ?w . ?z a :Tune . ?w a :Publisher }
  - SELECT ?z ?k WHERE { ?z :recordingYear ?k . ?z a :Tune . ?k a xsd:gYear }

Alternative abstractions do exist!
Retrieve/Match cqs to CPs, or possibly propose new ones
- agentrole.owl, timeindexedpersonrole.owl, timeinterval.owl, ...

Specialize/Compose/Expand CPs to local cq terminology
- person-playerrole, playing-instrument-on-a-tune, playing-on-a-tune-in-recordingyear

Populate ABox
- Person(CharlieParker), PlayerRole(AltoSaxPlayer), Tune(LoverMan), Session(LoverManWithParkerOnDial), ...

Run unit test/Iterate until fixed
- SELECT ?x ?y ?z ?w ?k
- WHERE {
  ?x a :Person .
  ?y a :PlayerRole .
  ?x ?s ?z .
  ?z a :Tune .
  ?z ?t ?w .
  ?w a :Publisher .}
XD iteration with Content OPs

- Requirements are divided into small stories
- get your story (local problem)
- divide & conquer
  - read carefully the story and divide them into simple sentences $s_1, \ldots, s_n$

FOR EACH SENTENCE $s_i$
- transform $s_i$ to an instance-free sentence (“abstraction”)
  - an instance can be either an individual or a property value (fact)
- transform the instance-free sentence to local competency questions (cqs)
- translate local cqs to queries to be submitted to the knowledge base, and collect them in a unit test [12]
- match the CP coverage to the local cqs
- identify the CPs you need, and associate each CP with the local cqs it covers
  - if any local competency question remains uncovered, define separate small ontologies that cover them, and import them into the ontology. Treat these as CPs (you might want to propose them on the odp-web)
- identify ontology elements to be specialized, and specialize them
- identify axioms and ontology elements to involve in the composition of chosen CPs, and compose them
  - i.e., define the composition axioms
- expand the ontology in order to cover uncovered competency question
- populate the ontology ABox with the instances from the story
  - complete the ABox with additional instances if needed
- test using the collected queries and fix until all tests succeed

END FOR
Experiments: first results

- During a four-day course for PhD students
  - Most have never constructed an ontology, or only a small example
  - Mostly taxonomies or lightweight ontologies
  - Most subjects familiar with some modeling language (like ER or UML), but only a few have tried OWL
- Background questionnaires, ontology design exercises (end of every day), subjective feedback questionnaire after exercise
  - first two days no patterns, second two days with patterns
- Some preliminary results based on subjective feedback questionnaires only
  - Main difficulties: mapping from the problem to the patterns, pattern composition
  - Most found the patterns useful and many perceived that they introduced some solution they did not think of themselves
  - Most perceived the second exercise as the easiest to solve, and the fourth as the most successfully modeled
  - The last day we had also got pattern proposals
Ongoing and future work

✓ Bootstrapping and improving functionalities in the ODP portal
✓ ODP APIs
  ‣ Building the NeOn Toolkit ODP plugin
  ‣ Continue with experimentation
  ‣ Use of CBR for pattern-based automatic ontology construction … authority moves behind the curtains: do we need to redefine the theory of numbers in order to pay the grocery account?

› Join the ODP community! [http://www.ontologydesignpatterns.org](http://www.ontologydesignpatterns.org)
References (1)


