Ontological Engineering for the Semantic Web
with special focus on
Pattern-based Ontology Transformation

Vojtěch Svátek

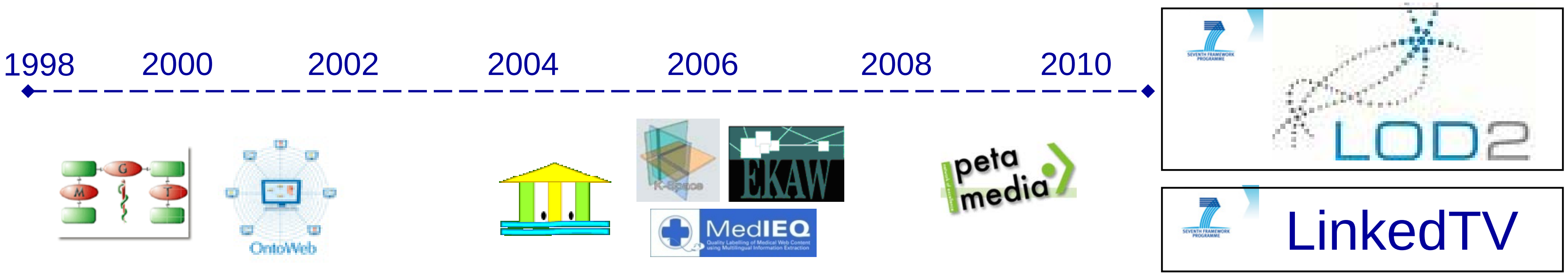
University of Economics, Prague (UEP)
Dept. of Information and Knowledge Engineering

svatek@vse.cz

ISSLOD 2011, September 13, 2011

Speaker's background

- MSc in information science, with focus on AI and expert systems, and eventually machine learning (1991);
  PhD (1998) on prior knowledge in propositional learning
- More than 10 years' research in **ontological engineering** (and related knowledge modelling: PSMs, clinical guidelines)
- In parallel various projects on data/text/multimedia mining
- In the last 2 years (obviously) interested in Linked Data as the 'proximal' side of the semweb: pushing at national level
- Backed by UEP's Knowledge Engineering Group, http://keg.vse.cz



1998 2000 2002 2004 2006 2008 2010

LinkedTV

Credits

- Ondra Šváb-Zamazal, Mirek Vacura (UEP)
- Aldo Gangemi, Valentina Presutti, Enrico Daga (ISTC/CNR, Rome)
- Luigi Iannone (Univ. Manchester)
- Francois Scharffe (INRIA / Univ. Montpellier)

Lecture blocks

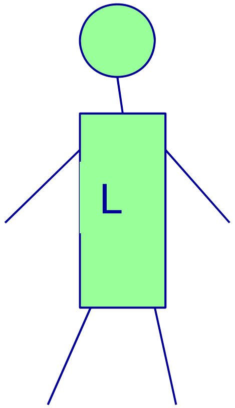
- Prelude: Semantic web as dancing party
- I. Linked Data and ontologies
  - role of ontological engineering on the semantic web
- II. Ontology patterns
  - design patterns & empirical patterns
- III. Pattern-based ontology transformation
  - principles, use cases, implemented tools

Semantic web as dancing party

- Dancers
  - – L = logician
  - – KE = knowledge engineer
  - – WE = web engineer
  - – SE DE = software engineer + data engineer
- Party hats
  - – AI = Artificial Intelligence
  - – Onto(logy)
  - – LD = Linked Data

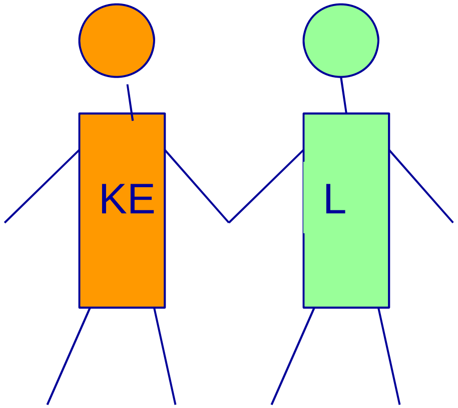
Semantic web as dancing party

- Since old times



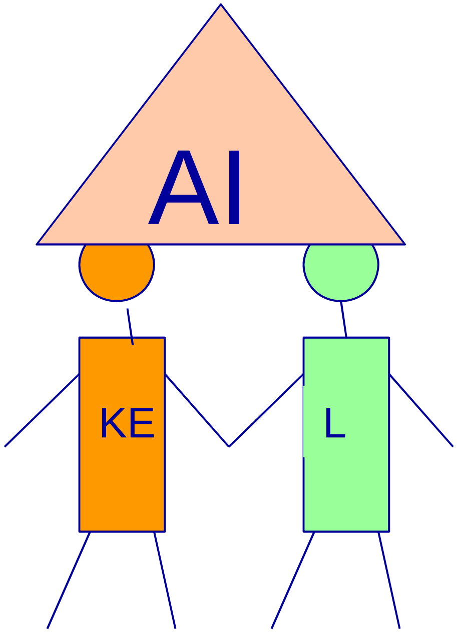
Semantic web as dancing party

- 1970s



Semantic web as dancing party

- 1970s

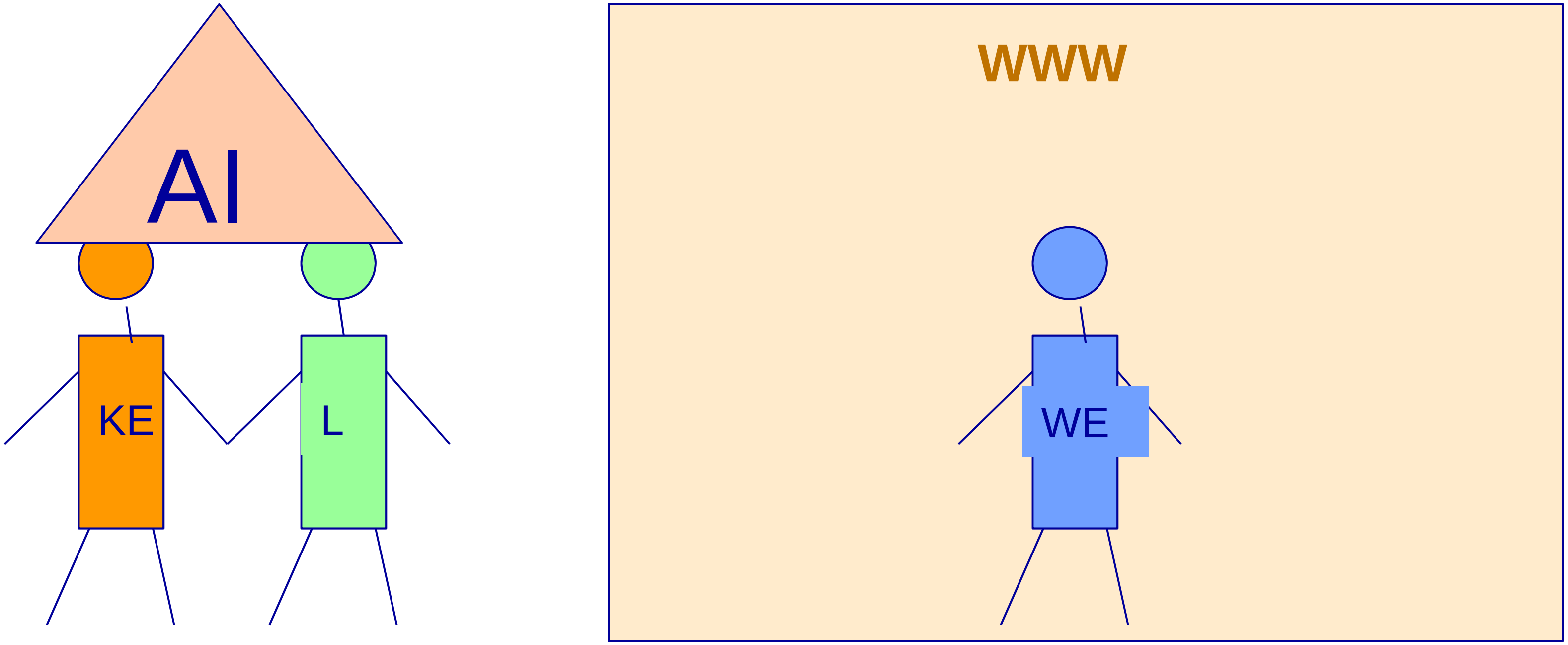


Semantic web as dancing party

- 1991

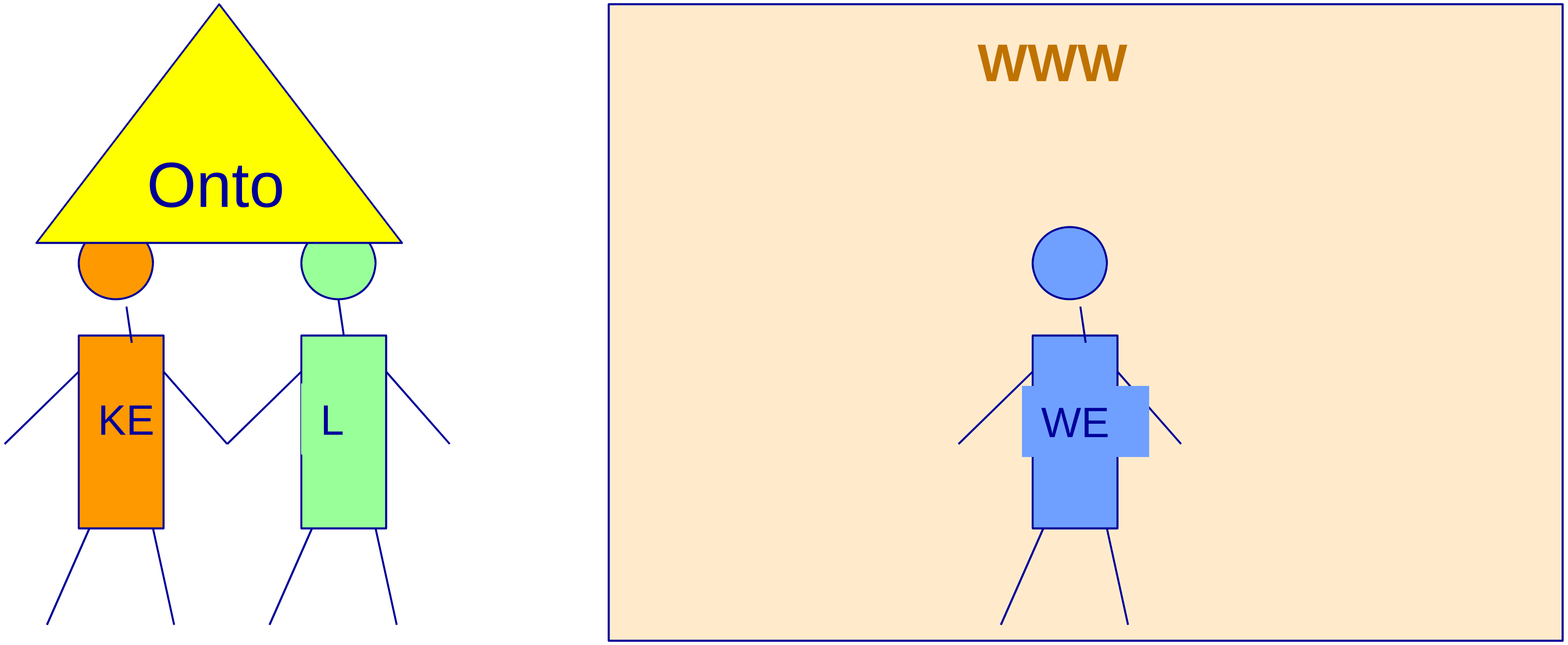
Semantic web as dancing party

- 1991



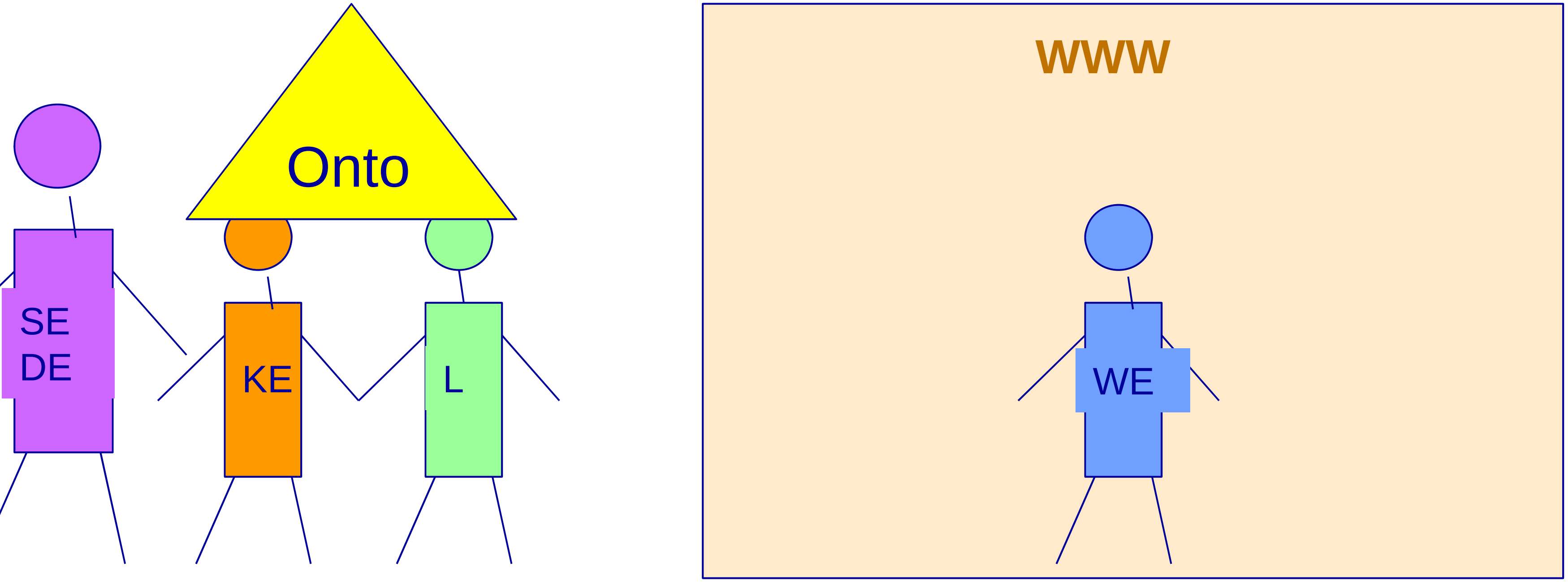
Semantic web as dancing party

- 1993



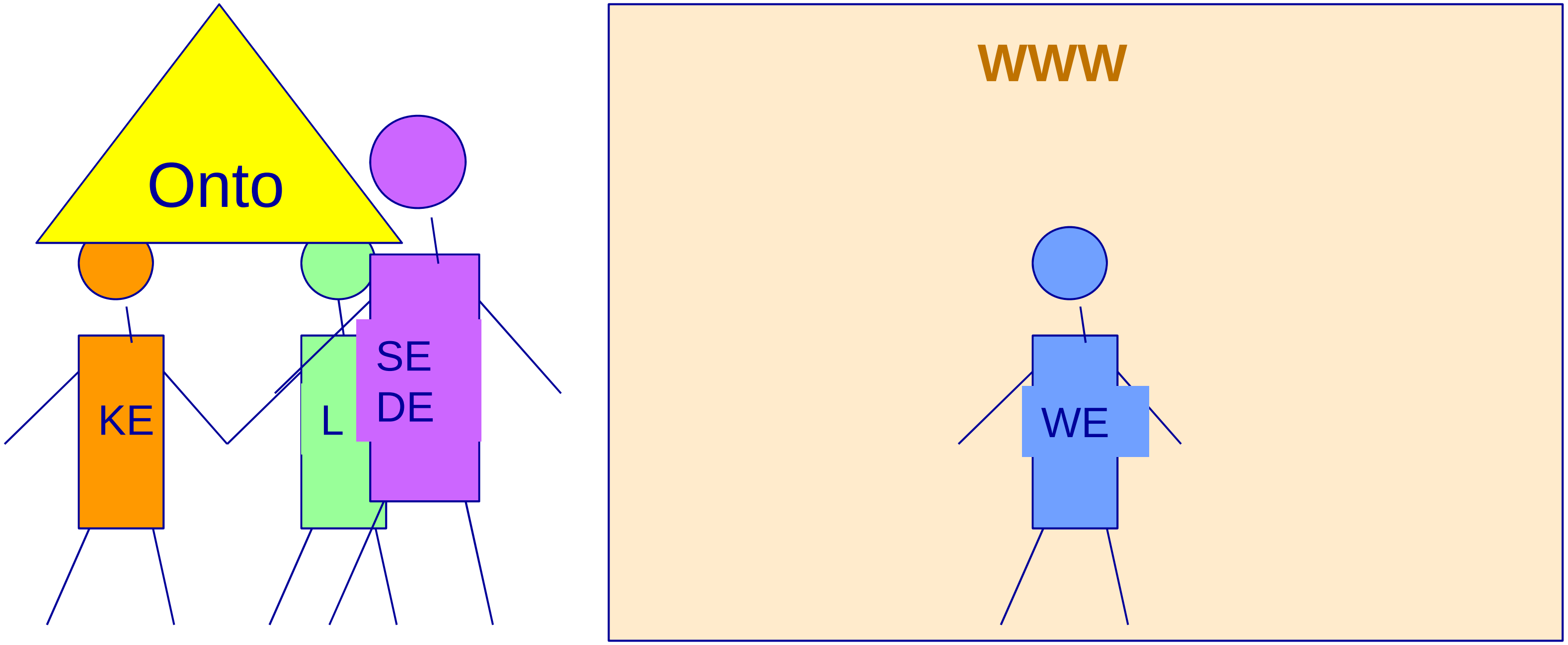
Semantic web as dancing party

- 1993



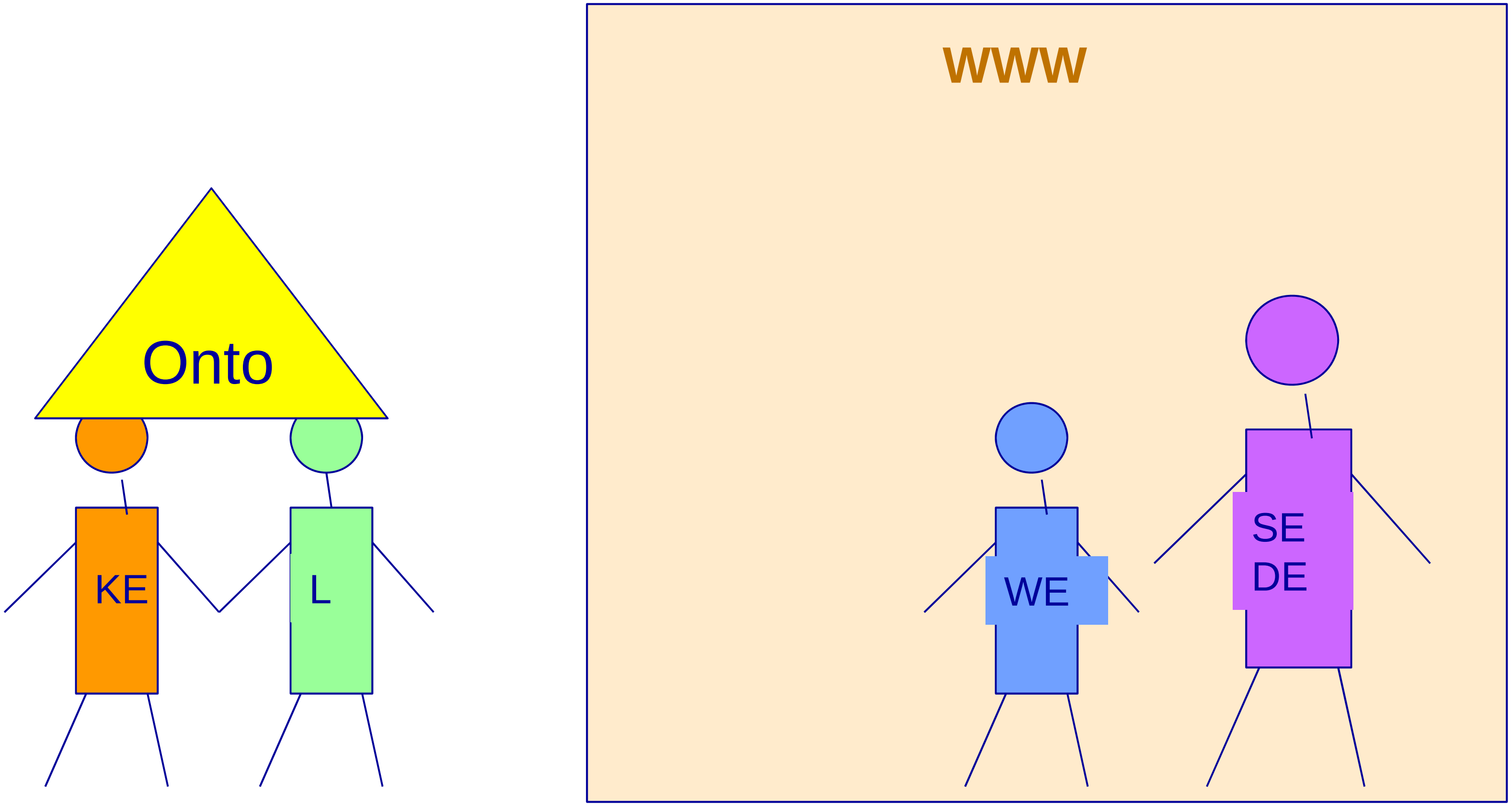
Semantic web as dancing party

- 1995



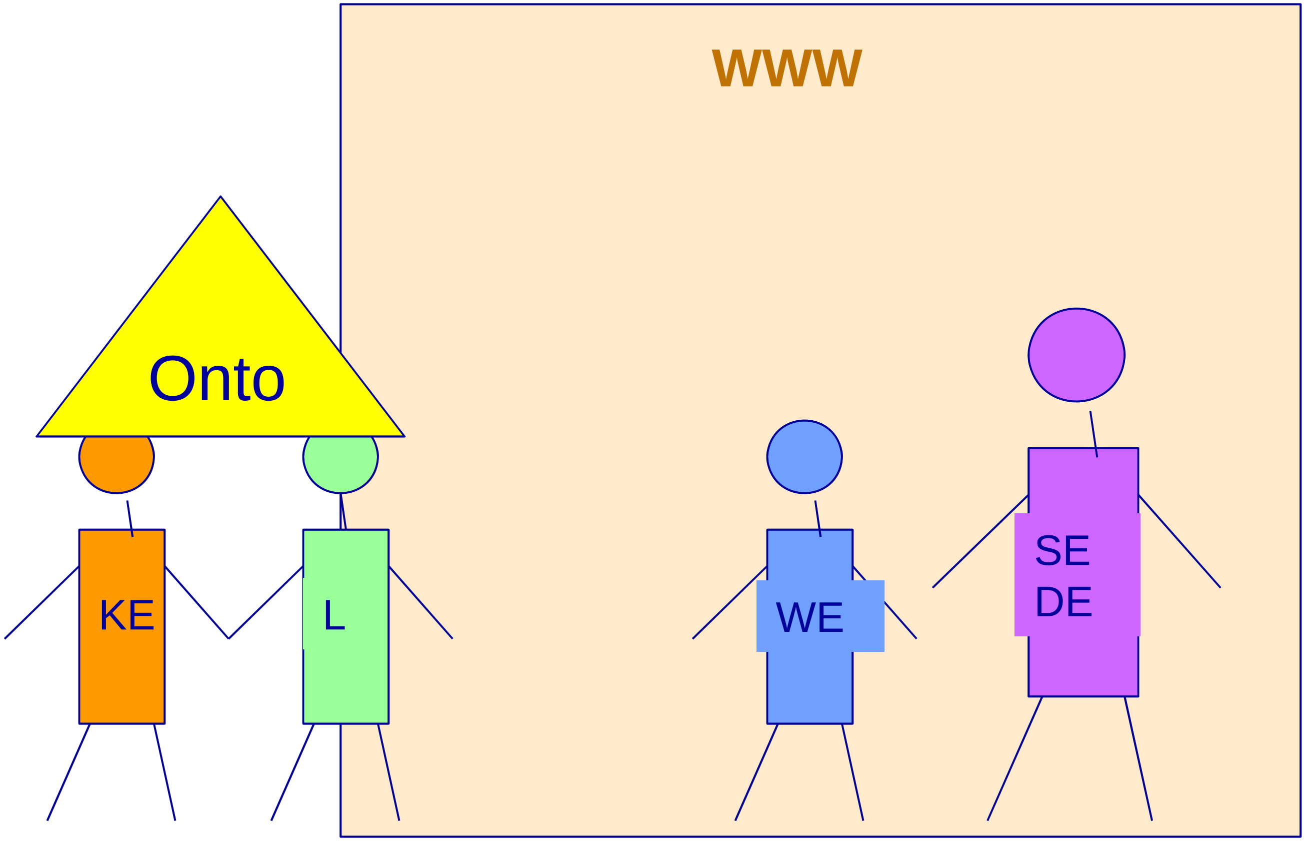
Semantic web as dancing party

- 1995



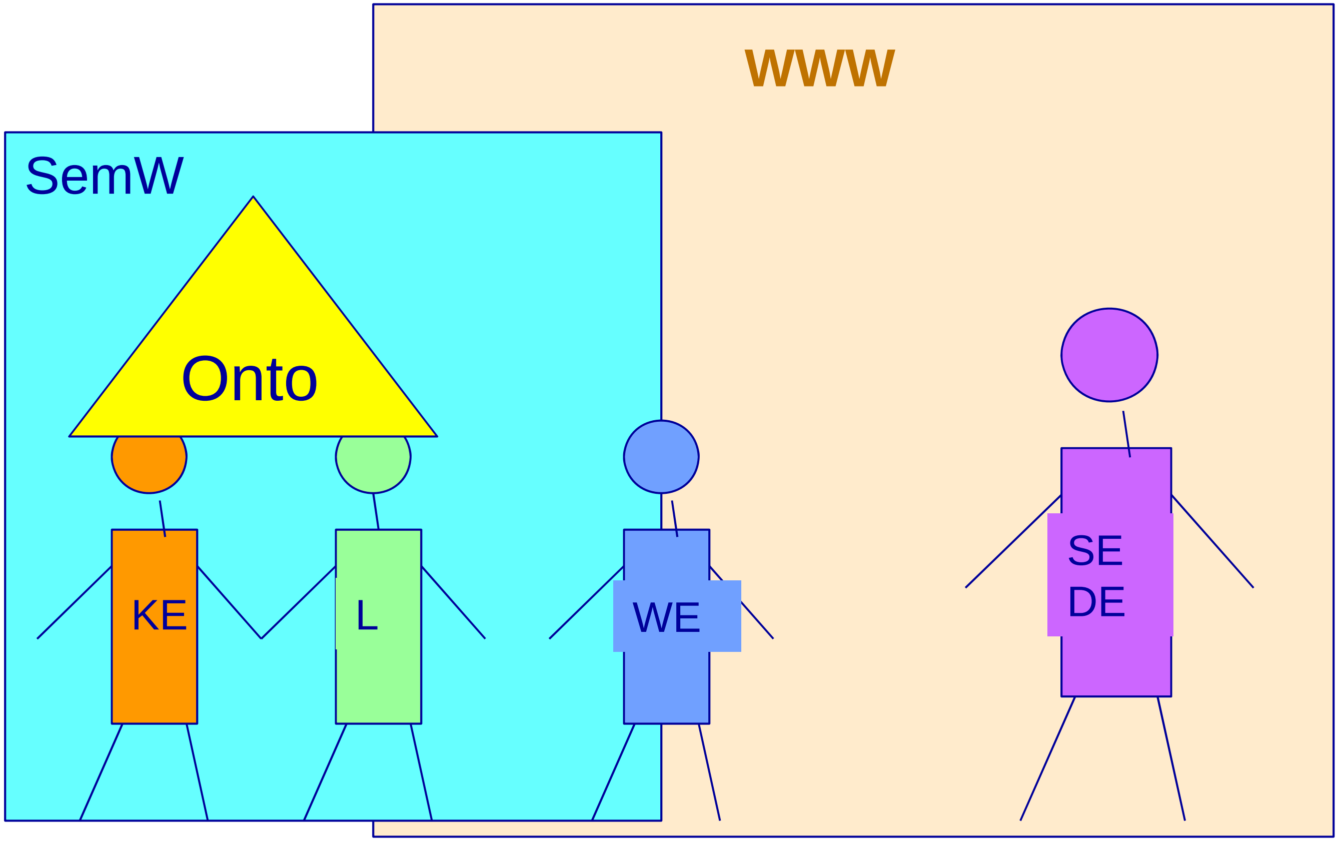
Semantic web as dancing party

- 2000



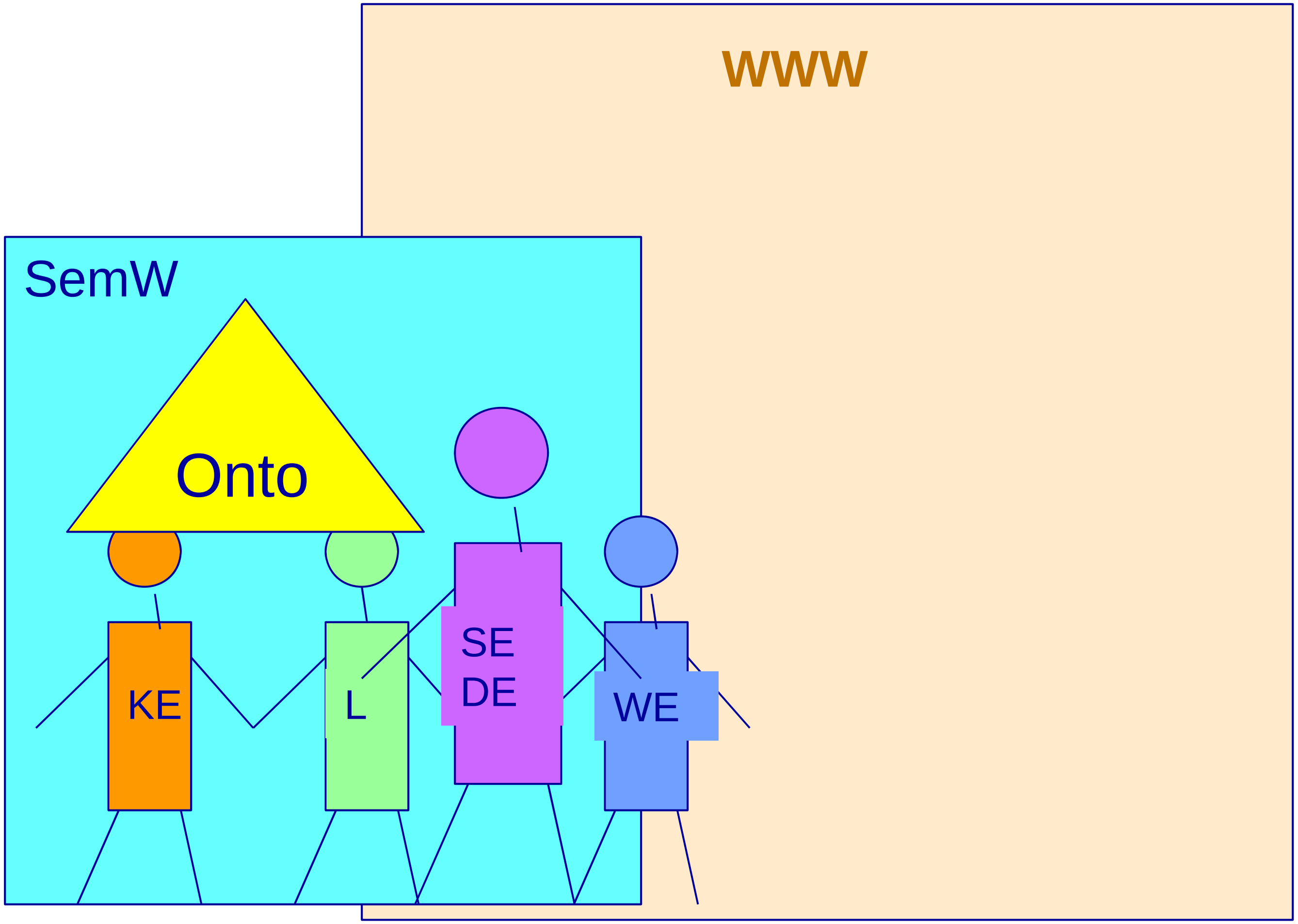
Semantic web as dancing party

- 2000



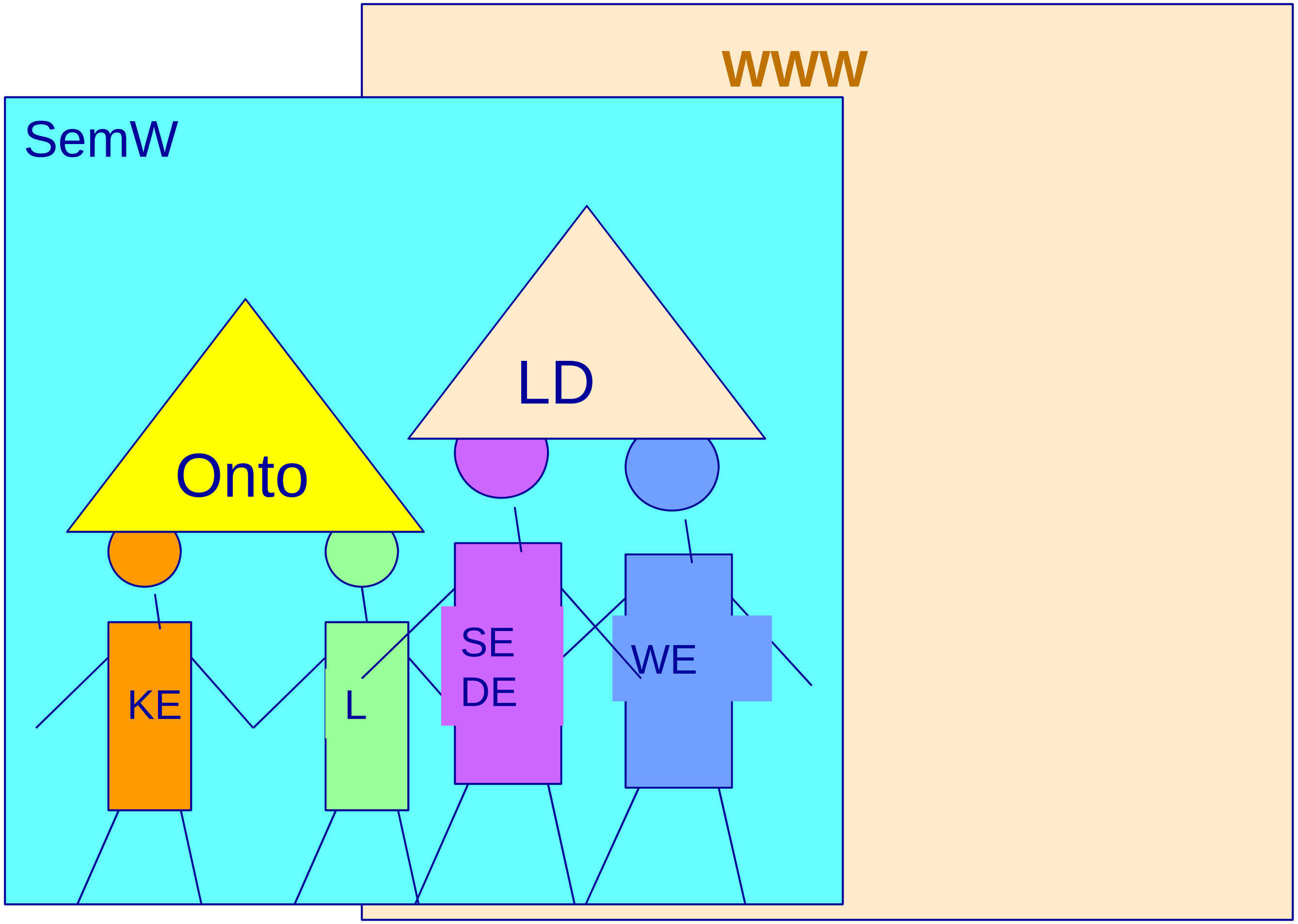
Semantic web as dancing party

- 2005



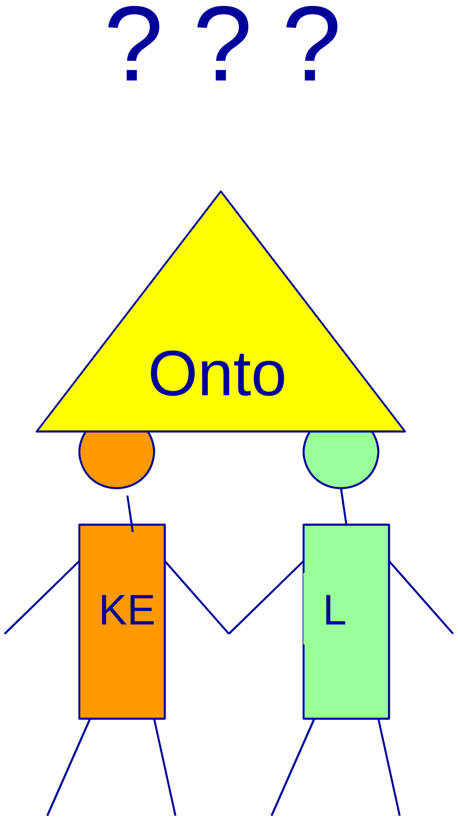
Semantic web as dancing party

- 2007



Semantic web as dancing party

- Is ontological research still a respected dancer?
- Or is it only at the party because there is no porter to kick away those who cannot dance the styles prescribed by the dancing order?



Block I

LINKED DATA AND ONTOLOGIES

Linked Data and ontologies: agenda

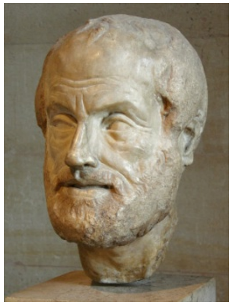
- What is/isn’t an ontology
- Typical settings for ontologies on the semweb (and nearby)
- Brief recap of the OWL language
- Why Linked Data engineers shouldn’t forget about ontological engineers

What is an ontology and what isn’t?

- In philosophy
  - discipline (dealing with ‘being’ as such)
  - system of categories of ‘beings’ in the world

What is an ontology and what isn't?

- In philosophy
  - discipline (dealing with 'being' as such)
  - system of categories of 'beings' in the world

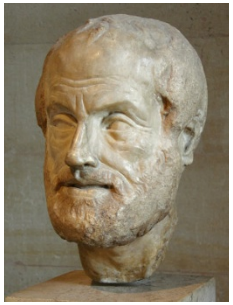


Aristotle: Definitio
per genus proximum
et differentia specifica

A PhD student is a student
that completed a master-level degree
and works on a scholarly topic under
the supervision of a senior researcher

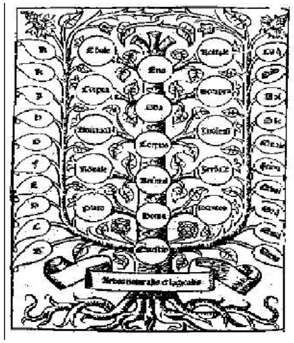
What is an ontology and what isn’t?

- In philosophy
  - discipline (dealing with ‘being’ as such)
  - system of categories of ‘beings’ in the world



Aristotle: Definitio
per genus proximum
et differentia specifica

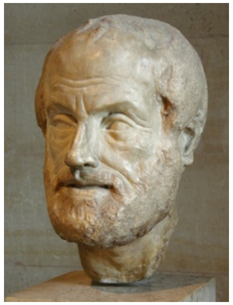
Porphyrian tree:
thinking vs. extended
animate vs. inanim.
rational vs. rrational
etc.



A PhD student is a student
that completed a master-level degree
and works on a scholarly topic under
the supervision of a senior researcher

What is an ontology and what isn’t?

- In philosophy
  - discipline (dealing with ‘being’ as such)
  - system of categories of ‘beings’ in the world



Aristotle: Definitio per genus proximum

cifica

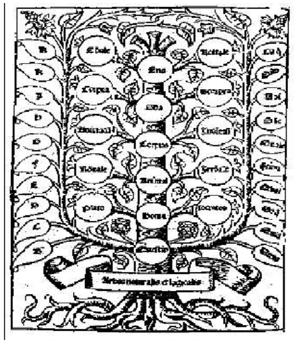
Modular system of interlinked definitions

A PhD stud

that con vel degree and works on a scholarly topic under the supervision of a senior researcher

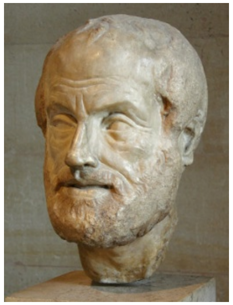
Porphyrian tree:

thinking vs. extended
animate vs. inanim.
rational vs. rrational
etc.



What is an ontology and what isn’t?

- In philosophy
  - – discipline (dealing with ‘being’ as such)
  - – system of categories of ‘beings’ in the world

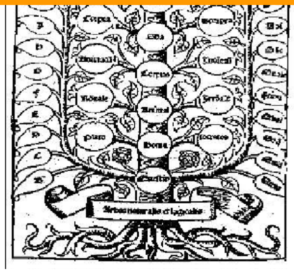


Aristotle: Definitio
per genus proximum
cifica

Modular system of interlinked definitions

Porphyrian tree:
thinking vs. extended
animate vs. inanim
ra
e

Systematic taxonomy



A PhD stud
that con vel degree
and works on a scholarly topic under
the supervision of a senior researcher

What is an ontology and what isn't?

- In computer science (and related fields)
  - information artifact
  - ...(mostly) conceptualizing a certain part of reality
  - ...in a shared manner
  - ...explicitly (not just in the minds),
  - ... in a formal way (concepts rigorously defined)
  - and/or is centered around a hierarchy of terms
- Elements of an ontology can provide semantics to other information elements – vocabulary aspect

Loosely according to Gruber (1993), Borst (1997) and others

What is an ontology and what isn’t?

- Is the following an ontology?

What is an ontology and what isn't?

- Is the following an ontology?
  - MyOntology.owl, which you create in Protégé or similar tool

What is an ontology and what isn’t?

- Is the following an ontology?
  - MyOntology.owl, which you create in Protégé or similar tool
  - Hierarchical chart, made by a panel of medical experts, which categorizes known forms of a virus

What is an ontology and what isn't?

- Is the following an ontology?
  - MyOntology.owl, which you create in Protégé or similar tool
  - Hierarchical chart, made by a panel of medical experts, which categorizes known forms of a virus
  - A set of description logics formulae, set up to illustrate an interesting phenomenon in tableau reasoning

What is an ontology and what isn’t?

- Is the following an ontology?
  - MyOntology.owl, which you create in Protégé or similar tool
  - Hierarchical chart, made by a panel of medical experts, which categorizes known forms of a virus
  - A set of description logics formulae, set up to illustrate an interesting phenomenon in tableau reasoning
  - A Linked Data vocabulary consisting of a set of properties for characterizing a movie

What is an ontology and what isn’t?

- Is the following an ontology?
  - MyOntology.owl, which you create in Protégé or similar tool

    No
  - Hierarchical chart, made by a panel of medica experts, which categorizes known forms of a

    Depends on view
  - A set of description logics formulae, set up to illustrate an interesting phenomenon in tableau reasoning

    No
  - A Linked Data vocabulary consisting of a set properties for characterizing a movie

    Depends on view

What you typically fall upon

What you typically fall upon

- Structured, NL-centered, hierarchical terminology
  - *Terminological ontology*
  - Primarily for improvement of text search

What you typically fall upon

- Structured, NL-centered, hierarchical terminology
  - *Terminological ontology*
  - Primarily for improvement of text search
- Algebraic structure (lattice)
  - Graph operations over terminology / object tables

What you typically fall upon

- Structured, NL-centered, hierarchical terminology
  - *Terminological ontology*
  - Primarily for improvement of text search
- Algebraic structure (lattice)
  - Graph operations over terminology / object tables
- (An advanced form of) schema for data
  - *Information ontology*
  - Primarily for data integration and structured search

What you typically fall upon

- Structured, NL-centered, hierarchical terminology
  - *Terminological ontology*
  - Primarily for improvement of text search
- Algebraic structure (lattice)
  - Graph operations over terminology / object tables
- (An advanced form of) schema for data
  - *Information ontology*
  - Primarily for data integration and structured search
- Knowledge base containing compositional definitions of concepts
  - *Knowledge ontology*
  - Primarily for inferential tasks in logics

Dichotomy from Presutti et al. (ESWC'09 tutorial)

- Coverage-oriented ontologies
  - Cover the terminology in a whole domain
  - Typically used for non-inferential tasks, often in relation to unstructured resources (annotation, retrieval...)
- Task-oriented ontologies
  - Provide semantics to structured facts / KBs
  - Typically used for querying and reasoning
  - Design guided by competence questions

Ontology languages (schema / logical)

- There is a plethora of...
- OWL (and its sublanguages incl. RDFS)
  - Description Logics (DL) semantics
  - standardized by W3C
- Other

  (most seek some interoperability with OWL)
  - Common Logic (ISO Standard), CycL
  - Frame-based (F-Logic etc.)
  - *GOL*
  - *Topic Maps (ISO Standard)*

Ontology languages (schema / logical)

- There is a plethora of...
- OWL (and its sublanguages incl. RDFS)
  - Description Logics (DL) semantics
  - standardized by W3C
- Other

  (most seek some interoperability with OWL)
  - Common Logic (ISO Standard), CycL
  - Frame-based (F-Logic etc.)
  - *GOL*
  - *Topic Maps (ISO Standard)*

OntoLeipzig

(Pre-)history of OWL

- Early KR systems based on DL, such as KL-ONE (1985), distinguished from frame systems
- SHOE (1998) – first ‘web ontology’ language, HTML-based
- DAML-ONT, OIL (2000)
  - more frame aspects (back) to DL; use of RDF
- DAML+OIL (2002) – combination of both
  - E.g. RDF-based instances (x OIL)
  - E.g. local restrictions on properties (x DAML-ONT)
- OWL became W3C Recommendation in 2004
- Current version, OWL 2, became W3C Recommendation in 2009
  - http://www.w3.org/TR/owl2-overview/

Basic representational features of OWL

- As for any DL language, an OWL knowledge base (‘ontology’, theory) consists of logical formulae, called axioms
- Axioms express statements regarding entities
  - Individuals (instances, objects, ...)
  - Classes (concepts, types, ...)
  - Properties (roles, predicates, binary relations, ...)

Annotations

- Besides the ‘logical’ aspect, OWL also allows to express `extra-logical’ meta-information via annotations
  - about the ontology as whole
    - ✓e.g. version
  - about entities declared in the ontology
    - ✓e.g. human-readable name of a class
  - about whole formulae (axioms)
    - ✓e.g. creation date

T-box, R-box and A-box

- A knowledge base may have three parts
  - T-box (terminological box)
  - R-box (role box)
  - A-box (assertional box)

T-box, R-box and A-box

- A knowledge base may have three parts
  - T-box (terminological box)
  - R-box (role box)
  - A-box (assertional box)

‘definitions’

‘facts’

T-box, R-box and A-box axioms

- A T-box axiom relates two class expressions
  - via equivalence (owl:equivalentClass) or subsumption (rdfs:subClassOf)
- An R-box axiom relates two property expressions
  - via equivalence (owl:equivalent...Properties) or subsumption (rdfs:subObjectPropertyOf)
- An A-box axiom
  - either assigns a class expression to an individual (rdf:type)
  - or relates two individuals by a property expression

Class and property expressions

- A class expression refers to a set of individuals
- It can be either
  - a named class as (atomic) entity
  - a complex class expression, e.g.
    - ✓ 'C1 and C2' (conjunction)
      'Book and ThingWrittenByBeneluxAuthor'
    - ✓ 'P some C' (existential restriction)
      'writtenBy some BeneluxWriter'
    - ✓ '{i1, i2, i3}' (enumeration)
      {Belgium,Netherlands,Luxembourg}
- Expressions can be further composed
  - *Book and (writtenBy some (Person and livesIn {Belgium,Netherlands,Luxembourg}))*

Class and property expressions

- A property expression refers to a set of ordered pairs of individuals
- It can be either a
  - named property as (atomic) entity
  - complex property expression
    - ✓e.g. ‘inverse of X’
- Property expressions can also be composed

A-box axioms

- Class expression instantiations

  MaigretAfraid a Book

  MaigretAfraid a (Book and (writtenBy some (Person and livesIn {Belgium,Netherlands,Luxembourg}))

- Property instantiations (‘normal facts’)

  MaigretAfraid writtenBy Simenon

(Official) Sublanguages of OWL

- OWL 2 EL
  - existential but not universal quantification
  - conjunction but not disjunction
  - suitable for consistency checking, subsumption and instance checking, even in large T-boxes
- OWL 2 QL
  - no quantification nor disjunction
  - suitable for querying large A-boxes
- OWL 2 RL
  - does not allow inference of anonymous individuals
  - suitable for inference by rule systems

Most important OWL syntaxes

- Functional syntax
  - Directly follows from structural specification of the language
- RDF/XML
  - Mandatory for any tool
  - Assures compliance to RDF processing
- Turtle
- OWL/XML
  - Assures compliance to XML processing
- Manchester syntax
  - Easy to read and write class expressions

Examples (from OWL 2 Primer)

- Class instantiation
  - Mary is a parent
- Object property assertion
  - Mary is John’s wife
- Equivalence axiom with existential restriction over a property
  - Some ‘thing’ is a parent if and only if ‘it’ has at least one child that is a person

Mary is a parent

- **Functional-Style Syntax**

  ```
  ClassAssertion( :Parent :Mary )
  ```

- **RDF/XML Syntax**

  ```
  < Parent rdf:about="Mary"/>
  ```

- **Turtle Syntax**

  ```
  :Mary rdf:type : Parent .
  ```

- **Manchester Syntax**

  ```
  Individual: Mary Types: Parent
  ```

- **OWL/XML Syntax**

  ```
  <ClassAssertion>
     <Class IRI=" Parent "/> <NamedIndividual IRI="Mary"/>
     </ClassAssertion>
  ```

Mary is John’s wife

- **Functional-Style Syntax**

```
ObjectPropertyAssertion( :hasWife :John :Mary )
```

- **RDF/XML Syntax**

```
<rdf:Description rdf:about="John"> <hasWife
   rdf:resource="Mary"/> </rdf:Description>
```

- **Turtle Syntax**

```
 :John :hasWife :Mary .
```

- **Manchester Syntax**

```
Individual: John Facts: hasWife Mary
```

- **OWL/XML Syntax**

```
<ObjectPropertyAssertion> <ObjectProperty IRI="hasWife"/>
   <NamedIndividual IRI="John"/>
   <NamedIndividual IRI="Mary"/>
   </ObjectPropertyAssertion>
```

Some ‘thing’ is a parent if and only if ‘it’ has at least one child that is a person

- **Functional-Style Syntax**

```
EquivalentClasses(  :Parent
   ObjectSomeValuesFrom( :hasChild :Person ) )
```

- **RDF/XML Syntax**

```
<owl:Class rdf:about="Parent">
   <owl:equivalentClass>
   <owl:Restriction>
   <owl:onProperty rdf:resource="hasChild"/>
   <owl:someValuesFrom rdf:resource="Person"/>
   </owl:Restriction>
   </owl:equivalentClass>
   </owl:Class>
```

Some ‘thing’ is a parent if and only if ‘it’ has at least one child that is a person

- **Turtle Syntax**

```
:Parent owl:equivalentClass
    [ rdf:type owl:Restriction ; owl:onProperty  :hasChild ;
    owl:someValuesFrom  :Person ] .
```

- **Manchester Syntax**

```
Class: Parent EquivalentTo: hasChild some Person
```

- **OWL/XML Syntax**

```
<EquivalentClasses>
    <Class IRI="Parent"/>
    <ObjectSomeValuesFrom>
    <ObjectProperty IRI="hasChild"/> <Class IRI="Person"/>
    </ObjectSomeValuesFrom>
    </EquivalentClasses>
```

OWL spread of use

- First-choice for ontologies designed under the influence of academia
- By http://pingthesemanticweb.com to date:
  - 549K documents use the OWL namespace
    - ✓ cf. FOAF: 1.3M
  - Presumably often due to owl:sameAs?

Linked Data view

- Semantics is defined by RDFS vocabularies
  - Mostly consensual to some degree
    - ✓ Research project consortia, VoCamps, ...
  - Structure influenced by ‘what is in data’
  - Usually small, flat, and adopted piecewise

- ‘Ontology-like’ classifications are sometimes modeled at the level of instances
  - E.g. through SKOS vocabulary
  - Usually not referred to as ontologies... but often could be viewed as *terminological* ontologies

Linked Data view

- Semantics is defined by RDFS vocabularies
  - – Mostly consensual to some degree
    - ✓Research project consortia, VoCamps, ..
  - – Structure influenced by ‘what is in data’
  - – Usually small, flat, and adopted piecewise

~ Task-oriented ontologies?

- ‘Ontology-like’ classifications are sometimes modeled at the level of instances
  - – E.g. through SKOS vocabulary
  - – Usually not referred to as ontologies... but often could be viewed as *terminological* ontologies

~ Coverage-oriented ontologies?

Linked Data view

- Ontologies are complex vocabularies
  - Hierarchical, axiomatized, … beyond RDFS
  - Hardly pay off unless inference desired
- (Rare) example: GoodRelations
  - http://www.heppnetz.de/projects/goodrelations/
  - Used by over 10K businesses to describe their company and product data
  - Pragmatically evolves towards a simple vocabulary
  - Yet toughly competes with even simpler approaches such as http://schema.org/
    - ✓Joint initiative by MS, Yahoo!, Google
    - ✓Microdata syntax, ignores RDF etc.

Where are the ‘true’ ontologies?

- Decent ‘knowledge ontologies’ now in medicine
  - Concepts in human anatomy, physiology etc. evolve slowly → there is accumulated experience
  - Very high degree of reuse → investments to careful modeling pay off
  - Very high numbers of mutually related concepts even in a single domain → manual maintenance of taxonomies is hard → room for logical inference
  - DL applications (T-box) have been tested in this domain from the beginning
    - ✓GALEN project (1990s)
  - SNOMED-OWL (400K concepts in 2007)

Examples (SNOMED)

- Concept defined based on other concepts

  -
    ```
    Appendicectomy equivalentTo
    Surgical_Procedure and
    (method some Excision) and
    (procedure-site some Appendix_structure)
    ```

- Unnamed concept

  -
    ```
    Excision and (procedure-site some (kidney
    and (laterality some left)))
    ```

Inferencing in the current Web of Data?

- Classical deductive inference often inadequate
- Some non-standard inference methods under investigation: LARKC project http://www.larkc.eu
  - Tackles some real problems of web data (vagueness, incompleteness…)
  - However, adds further complexity to current reasoners (which are already tough for non-experts)

Inferencing in the current Web of Data?

- If simple A-box inference needed, it can be implemented as ‘inference on demand’
  - SPARQL CONSTRUCT
- Integrity constraints checking
  - ‘repair’ in ORE system (Lehmann et al.)
  - SPIN language proposal by TopQuadrant?
- Inductive inferencing
  - ‘analytical’ rather than ‘transactional’ level of LD
  - ‘enrichment’ in ORE
- In any case, inferencing should be applied selectively, with care, in order not to destroy the scalability and transparency of LD infrastructure

Now comes a quizz, to relax

- A ‘real world problem’ is presented

- Task 1: Suggest a solution for the problem

  (... there might be more solutions – for the next step let’s consider the solution endorsed by me)

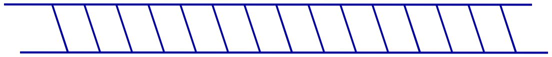
- Task 2: Try to decode the problem and its solution as a metaphor in the semweb/LD context

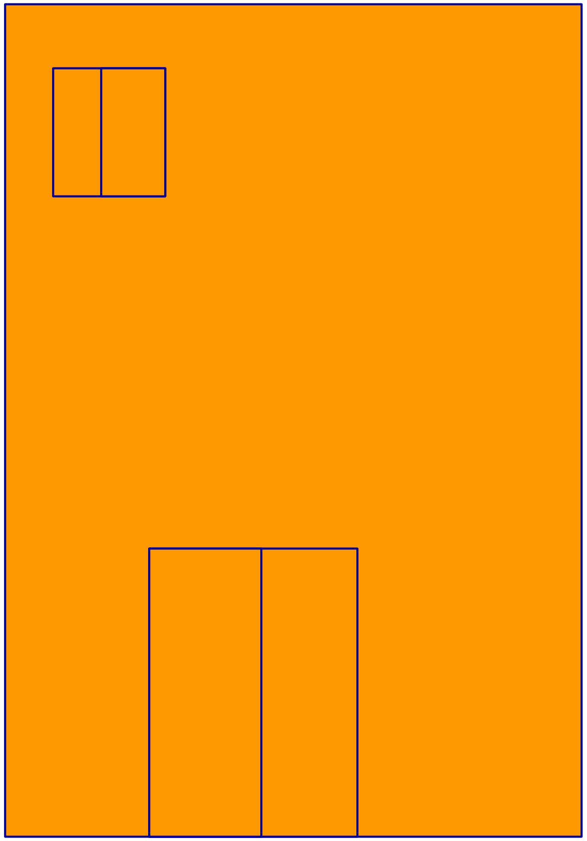
Problem I: Heal the famine using silos

- Problem description:
  - You are a leader of a tribe
  - You got a permission from the king to get grain for your people

Problem I: Heal the famine using silos

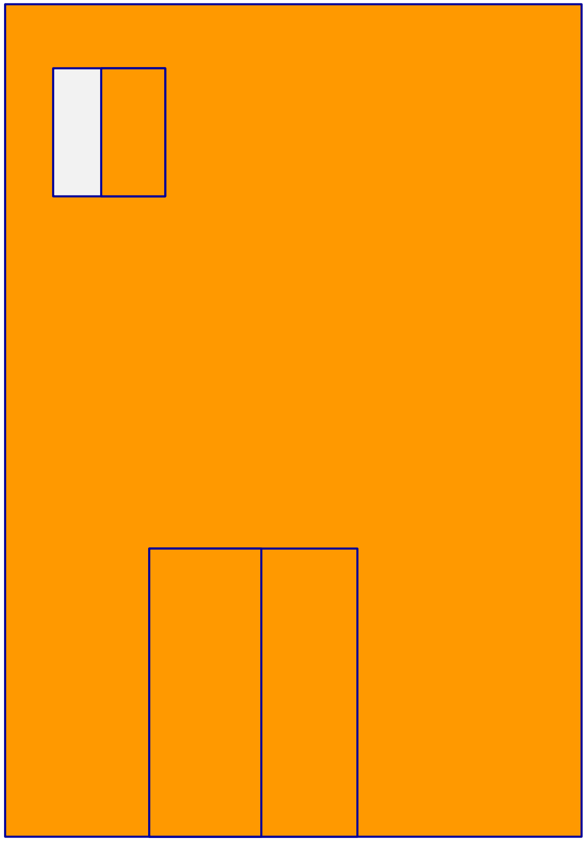
- Problem description:
  - You are a leader of a tribe
  - You got a permission from the king to get grain for your people
  - The granary master is willing to give you grain, but the entrance to the granary is rusted

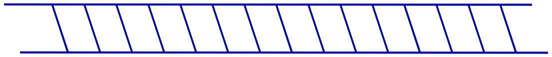




Problem I: Heal the famine using silos

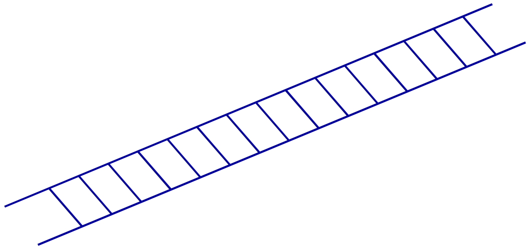
- Problem description:
  - You are a leader of a tribe
  - You got a permission from the king to get grain for your people
  - The granary master is willing to give you grain, but the entrance to the granary is rusted

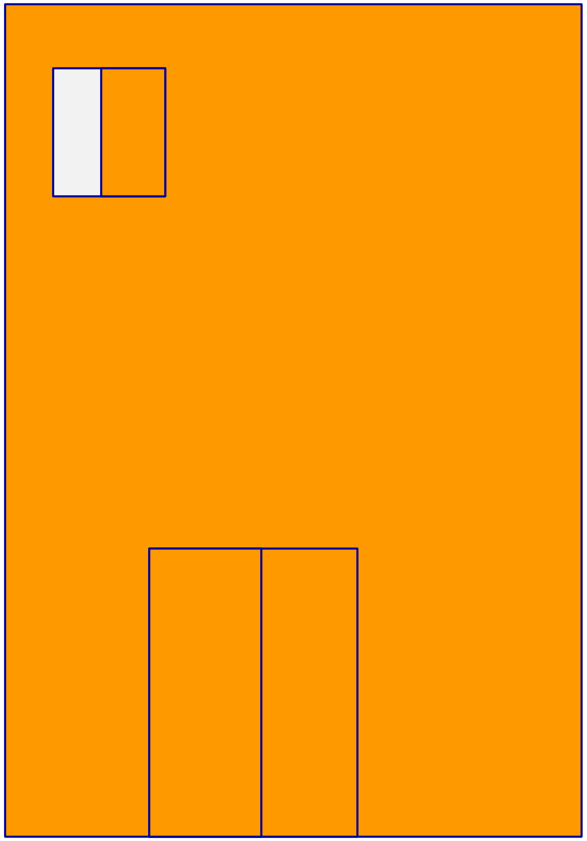




Problem I: Heal the famine using silos

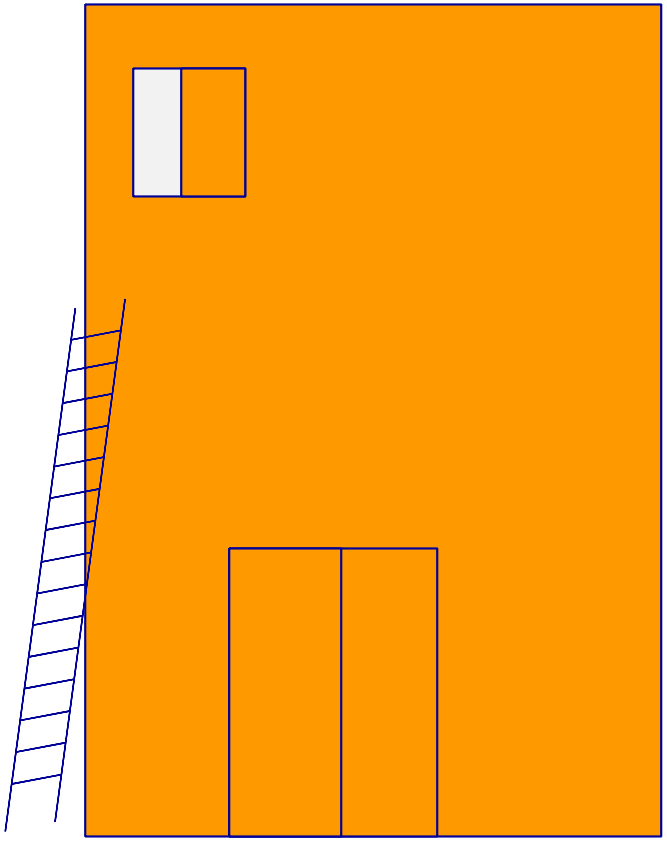
- Problem description:
  - You are a leader of a tribe
  - You got a permission from the king to get grain for your people
  - The granary master is willing to give you grain, but the entrance to the granary is rusted





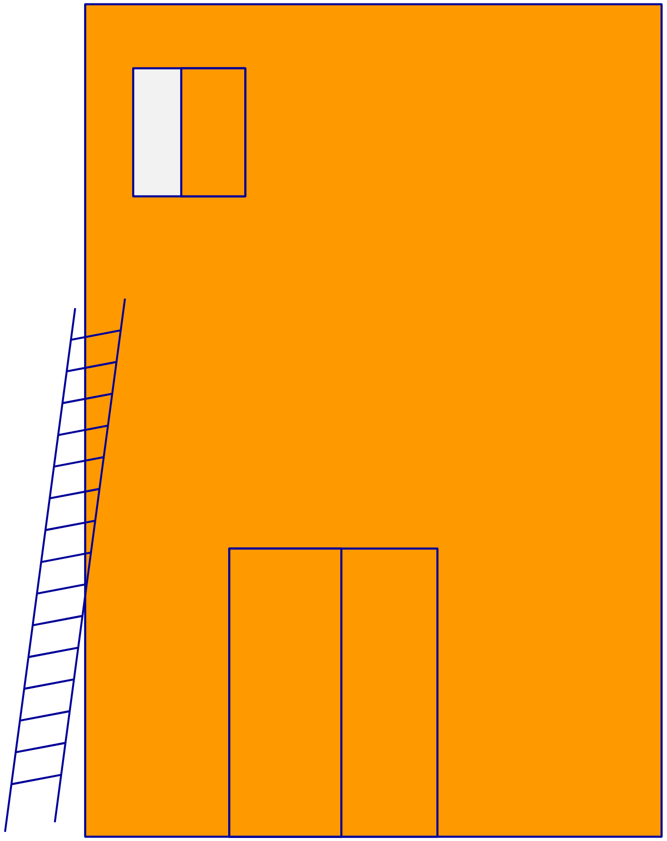
Problem I: Heal the famine using silos

- Problem description:
  - You are a leader of a tribe
  - You got a permission from the king to get grain for your people
  - The granary master is willing to give you grain, but the entrance to the granary is rusted
  - *When enough grain is taken away, the entrance could be open from inside*



Problem I: Heal the famine using silos

- Metaphor for:
  - ‘Raw data first’ principle
  - Initially large effort from consumer/mediator needed
  - Real use of data encourages further data opening / publisher-side enhancement

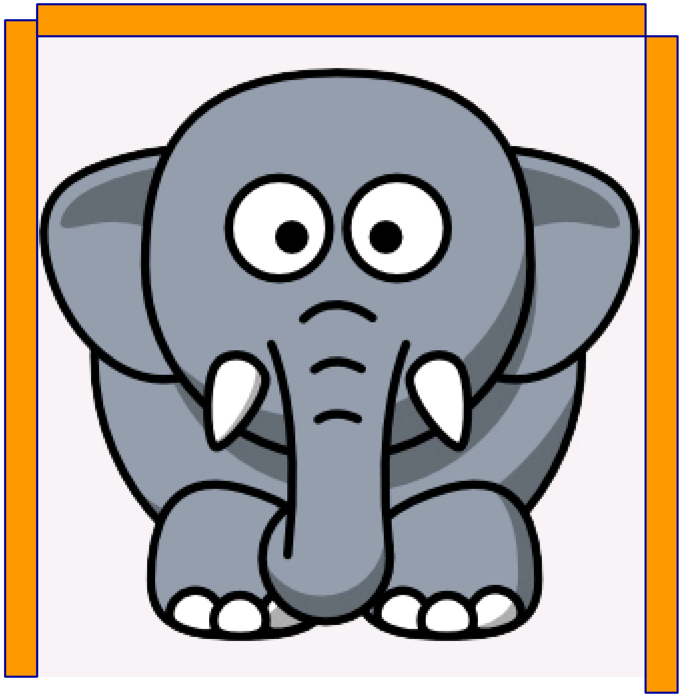


Problem II: Elephant in zoo

- You are a zoo director
- You managed to build the elephant pavilion, and introduced the first elephant

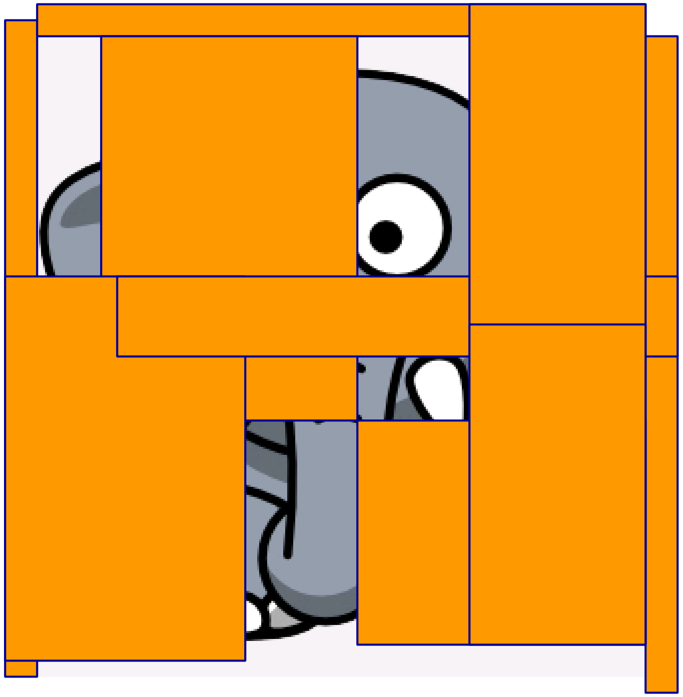
Problem II: Elephant in zoo

- You are a zoo director
- You managed to build the elephant pavilion, and introduced the first elephant
- Children are afraid of approaching, as there was a ‘Furious elephant’ movie on the TV



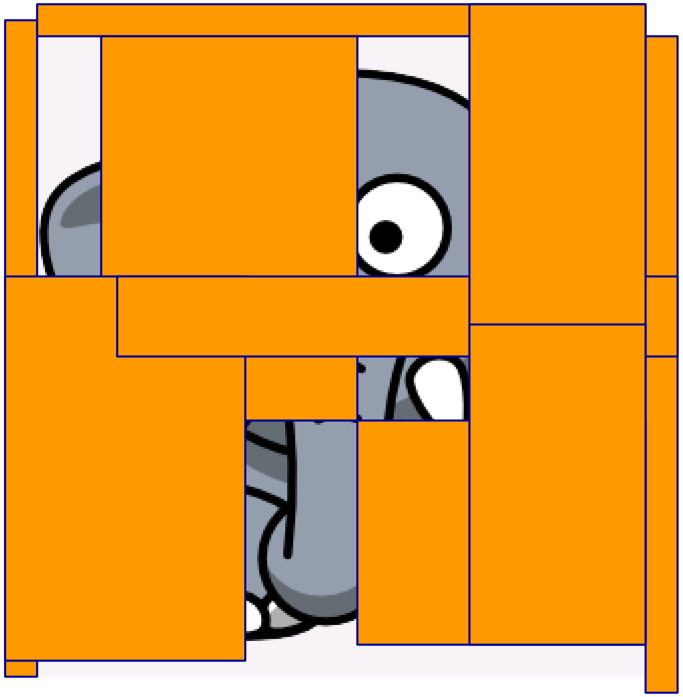
Problem II: Elephant in zoo

- You are a zoo director
- You managed to build the elephant pavilion, and introduced the first elephant
- Children are afraid of approaching, as there was a ‘Furious elephant’ movie on the TV
- *When the initial worry dissolves, they will want to see it whole*



Problem II: Elephant in zoo

- Metaphor for:
  - Web application developers ignore LD resources, as they perceive RDF/SPARQL as too complex and hard to learn
  - REST APIs on top of LD provide ‘RDF-free’ access to fragments of resources’ content
  - This encourages to later explore advanced access options



Problem III: Cupboards and drill bits

- You need to hang bookshelves of various size on the wall
- You picked up a drill bit that would make holes for heavy-duty screws, capable of carrying any shelf you think of

Problem III: Cupboards and drill bits

- You need to hang bookshelves of various size on the wall
- You picked up a thick drill bit that would make holes for heavy-duty screws, capable of carrying any shelf you think of
- However, the drill only made shallow dents into the plaster
- Nothing but empty shelves can be hung

Problem III: Cupboards and drill bits

- You need to hang bookshelves of various size on the wall
- You picked up a thick drill bit that would make holes for heavy-duty screws, capable of carrying any shelf you think of
- However, the drill only made shallow dents into the plaster
- Nothing but empty shelves can be hung
- *Use a narrow bit first, to get deeper*
  - *You hang at least small shelves*
  - *Heavy-duty bit (if ready in toolbox!) can thrust easier into an existing, narrow hole*

Problem III: Cupboards and drill bits

- Metaphor for:
  - Starting the semantic web with complex schemata didn’t work much
  - Simple LD schemata allow to develop useful though lightweight applications
  - More sophisticated ontologies should only be widely applied after the simple schemata sufficiently proved to work
  - Such ontologies should be developed and maintained already now (by an effort from academia); they cannot be instantly built when eventually needed!

OE: thick drill bit handy in the tool box

- Schemas are adopted based on their popularity and simplicity
- Cost: often conceptually simplified (if not wrong)
- This may lead to problems when already established communities open and interact
  - see the FOAF study in Block III
- Ontological engineering may help
  - Not (necessarily) by rebuilding the schemas proper
  - Rather as an additional, optional layer
- ‘Reactive’ rather than ‘proactive’ attitude of ontological engineering needed now to advance the semantic web

What do the quizz and lecture have in common?

- Object-level relationship: Problem III was mapped on the role of ontological engineering on the semantic web
- Meta-level relationship: The use of metaphors as such is analogous to the LD practice
  - Terms such as ‘bull’ and ‘bear’ for stock-exchange market trends are efficient and mnemotechnic
    - ✓ vs. “market with increasing investor confidence” etc.
  - However, when an outsider steps in, some explanation is necessary
  - Just as solid ontological modeling on top of popular schemata may show useful when moving beyond original communities of ‘tacit consensus’

Block II

ONTOLOGY PATTERNS

Ontology patterns: agenda

- Ontological engineering context
- Overview of pattern types
- Ontology content patterns and the XD approach
- Logical/structural patterns in OWL
- Naming patterns

Inventory of an ontological engineer

- *Set of requirements on the specific ontology*
- Elementary logical constructs (e.g. OWL)
- Existing ontologies / vocabularies (e.g. FOAF)
- Non-formalized schemata
- Conventions and practices
- Software tools (editors, reasoners...)

Adapted from Presutti and al., ESWC’09 tutorial

Ontology design patterns

- Reusable successful solutions to a recurrent modeling problem
- Cf. patterns in software engineering (SE) – typically consist of
  - Problem description
  - Suggested solution
  - Implementation guidelines
  - Discussion on consequences of using the pattern

Design vs. empirical patterns

- Design patterns
  - used intentionally
- Empirical patterns
  - discovered in artifacts
  - may result from design patterns
  - may produce design patterns (even if appeared spontaneously)
- Due to low maturity of ontological engineering, design patterns mostly considered so far
- With growing amount of ontologies available, empirical patterns gain on importance
  - Šváb-Zamazal (2008), Mikroyannidi (2011)

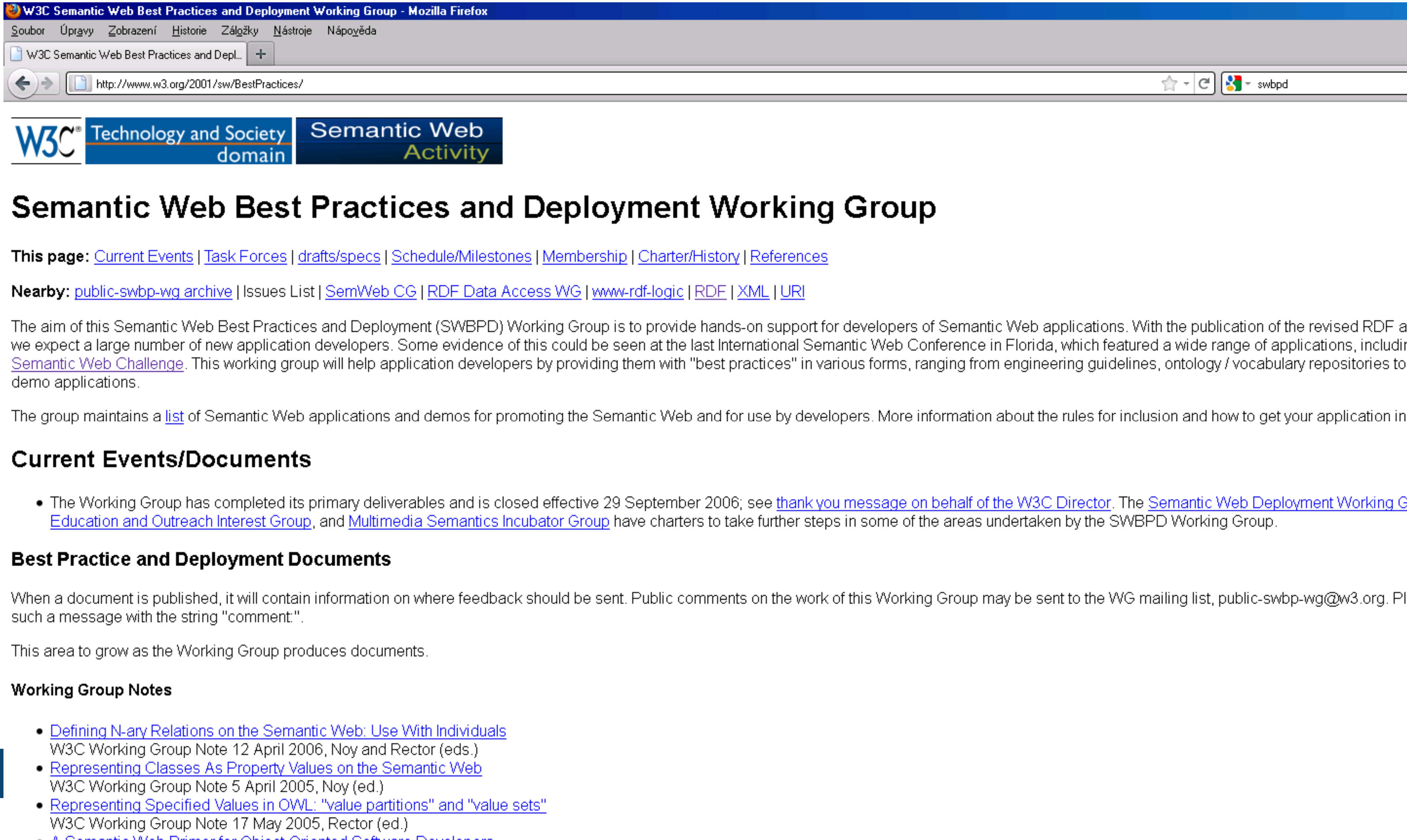
Pre-cursor: Clark’s knowledge patterns (1997)

- An ontology is not just a list of axioms, but a collection of abstract, modular theories and associated modeling decisions
- Examples:
  - a ‘distribution network’ pattern can be used to model electric circuits or
  - a ‘container’ pattern can be used to model bank accounts or computers
- Mapping of the elements (signature) of the pattern to elements of a concrete setting is specified
- Similarly to SE patterns helps avoid repeated writing of same-structured axioms

Traditional streams (after 2000)

- Logical ontology design patterns
  - Address some limitation of a modelling language
  - For OWL: primarily by
    - ✓W3C notes by the SWBPD – OEP group
    - ✓Univ. Manchester (web catalogue)
- Ontology content design patterns
  - Reusable building blocks
  - Often derived from foundational ontologies (esp. DOLCE), originally language-independent
  - To be imported to new / reengineered ontologies (as whole - unlike current vocabularies)
  - Primarily by ISTC/CNR Rome

W3C SWBPD WG OEP TF


W3C Semantic Web Best Practices and Deployment Working Group - Mozilla Firefox
Soubor Úpravy Zobrazení Historie Záložky Nástroje Nápověda
W3C Semantic Web Best Practices and Depl...
+
http://www.w3.org/2001/sw/BestPractices/
swbpd
W3C
Technology and Society domain
Semantic Web Activity

Semantic Web Best Practices and Deployment Working Group

This page: Current Events | Task Forces | drafts/specs | Schedule/Milestones | Membership | Charter/History | References

Nearby: public-swbp-wg archive | Issues List | SemWeb CG | RDF Data Access WG | www-rdf-logic | RDF | XML | URI

The aim of this Semantic Web Best Practices and Deployment (SWBPD) Working Group is to provide hands-on support for developers of Semantic Web applications. With the publication of the revised RDF a we expect a large number of new application developers. Some evidence of this could be seen at the last International Semantic Web Conference in Florida, which featured a wide range of applications, includir Semantic Web Challenge. This working group will help application developers by providing them with "best practices" in various forms, ranging from engineering guidelines, ontology / vocabulary repositories to demo applications.

The group maintains a list of Semantic Web applications and demos for promoting the Semantic Web and for use by developers. More information about the rules for inclusion and how to get your application in

Current Events/Documents

- The Working Group has completed its primary deliverables and is closed effective 29 September 2006; see thank you message on behalf of the W3C Director. The Semantic Web Deployment Working G Education and Outreach Interest Group, and Multimedia Semantics Incubator Group have charters to take further steps in some of the areas undertaken by the SWBPD Working Group.

Best Practice and Deployment Documents

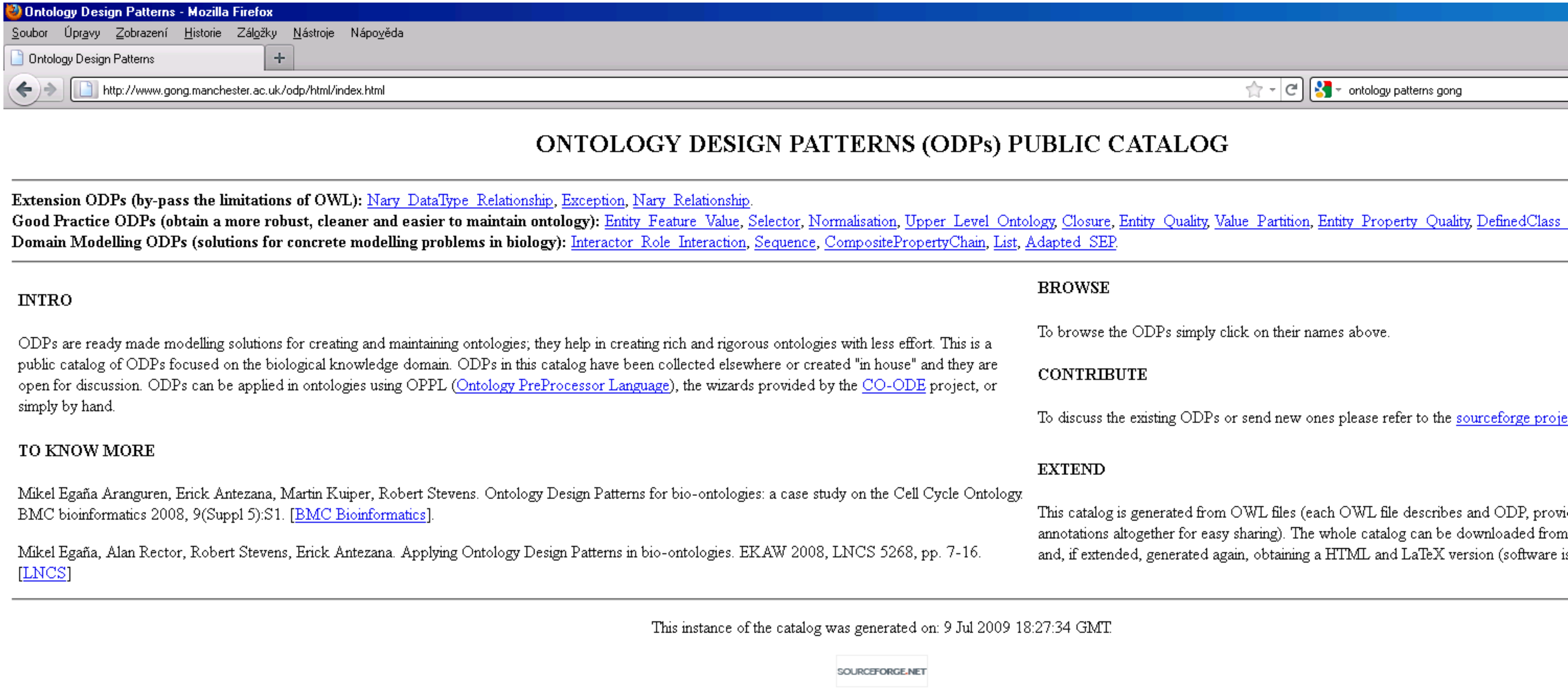
When a document is published, it will contain information on where feedback should be sent. Public comments on the work of this Working Group may be sent to the WG mailing list, public-swbp-wg@w3.org. Pl such a message with the string "comment:".

This area to grow as the Working Group produces documents.

Working Group Notes

- Defining N-ary Relations on the Semantic Web: Use With Individuals
  W3C Working Group Note 12 April 2006, Noy and Rector (eds.)
- Representing Classes As Property Values on the Semantic Web
  W3C Working Group Note 5 April 2005, Noy (ed.)
- Representing Specified Values in OWL: "value partitions" and "value sets"
  W3C Working Group Note 17 May 2005, Rector (ed.)

Manchester (bioinformatics-oriented) catalogue


Ontology Design Patterns - Mozilla Firefox
Soubor Úpravy Zobrazení Historie Záložky Nástroje Nápověda
Ontology Design Patterns
http://www.gong.manchester.ac.uk/odp/html/index.html
ontology patterns gong
SOURCEFORGE.NET

ONTOLOGY DESIGN PATTERNS (ODPs) PUBLIC CATALOG

Extension ODPs (by-pass the limitations of OWL): Nary_DataType_Relationship, Exception, Nary_Relationship.
Good Practice ODPs (obtain a more robust, cleaner and easier to maintain ontology): Entity_Feature_Value, Selector, Normalisation, Upper_Level_Ontology, Closure, Entity_Quality, Value_Partition, Entity_Property_Quality, DefinedClass_
Domain Modelling ODPs (solutions for concrete modelling problems in biology): Interactor_Role_Interaction, Sequence, CompositePropertyChain, List, Adapted_SEP.

INTRO

ODPs are ready made modelling solutions for creating and maintaining ontologies; they help in creating rich and rigorous ontologies with less effort. This is a public catalog of ODPs focused on the biological knowledge domain. ODPs in this catalog have been collected elsewhere or created "in house" and they are open for discussion. ODPs can be applied in ontologies using OPPL (Ontology PreProcessor Language), the wizards provided by the CO-ODE project, or simply by hand.

TO KNOW MORE

Mikel Egaña Aranguren, Erick Antezana, Martin Kuiper, Robert Stevens. Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. BMC bioinformatics 2008, 9(Suppl 5):S1. [BMC Bioinformatics].

Mikel Egaña, Alan Rector, Robert Stevens, Erick Antezana. Applying Ontology Design Patterns in bio-ontologies. EKAW 2008, LNCS 5268, pp. 7-16. [LNCS]

BROWSE

To browse the ODPs simply click on their names above.

CONTRIBUTE

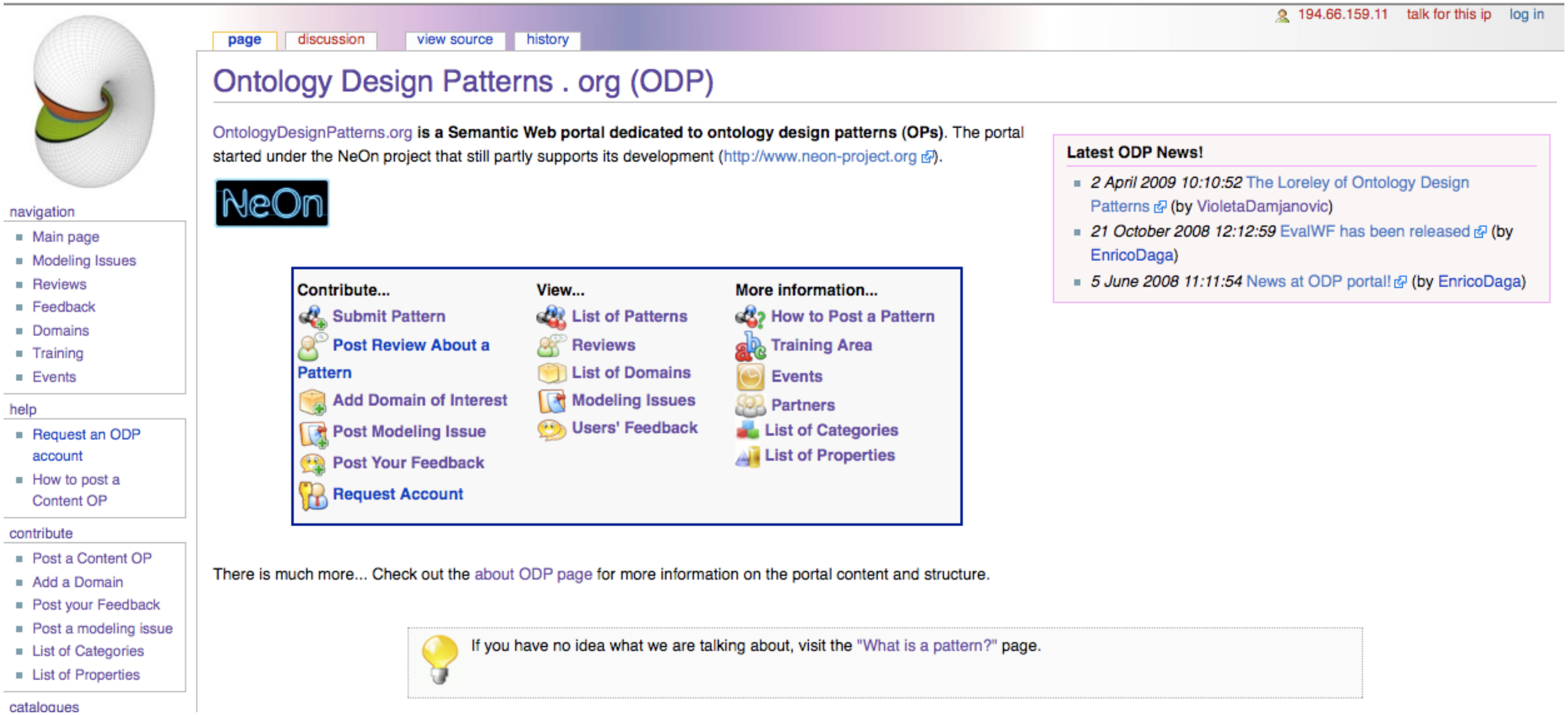
To discuss the existing ODPs or send new ones please refer to the sourceforge proje

EXTEND

This catalog is generated from OWL files (each OWL file describes and ODP, provi
annotations altogether for easy sharing). The whole catalog can be downloaded from
and, if extended, generated again, obtaining a HTML and LaTeX version (software i

This instance of the catalog was generated on: 9 Jul 2009 18:27:34 GMT.

The *ontologydesignpatterns.org* catalogue



194.66.159.11 talk for this ip log in

page discussion view source history

Ontology Design Patterns . org (ODP)

OntologyDesignPatterns.org **is a Semantic Web portal dedicated to ontology design patterns (OPs)**. The portal started under the NeOn project that still partly supports its development (http://www.neon-project.org).

NeOn

Contribute...
- Submit Pattern
- Post Review About a Pattern
- Add Domain of Interest
- Post Modeling Issue
- Post Your Feedback
- Request Account

View...
- List of Patterns
- Reviews
- List of Domains
- Modeling Issues
- Users' Feedback

More information...
- How to Post a Pattern
- Training Area
- Events
- Partners
- List of Categories
- List of Properties

Latest ODP News!
- *2 April 2009 10:10:52* The Loreley of Ontology Design Patterns (by VioletaDamjanovic)
- *21 October 2008 12:12:59* EvalWF has been released (by EnricoDaga)
- *5 June 2008 11:11:54* News at ODP portal! (by EnricoDaga)

There is much more... Check out the about ODP page for more information on the portal content and structure.

If you have no idea what we are talking about, visit the "What is a pattern?" page.

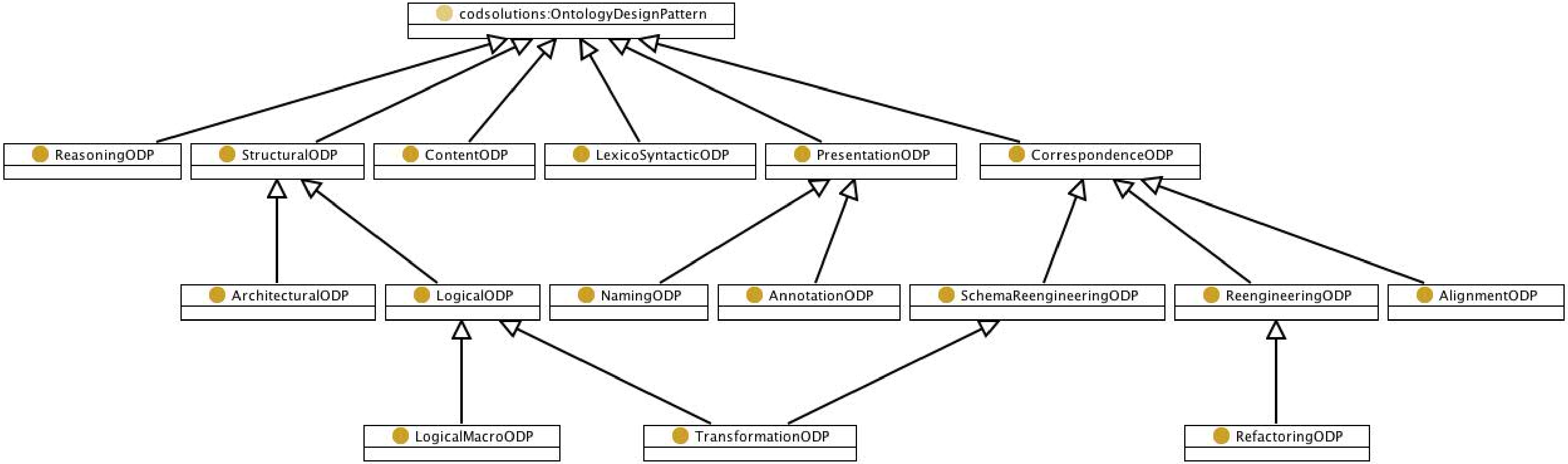
navigation
- Main page
- Modeling Issues
- Reviews
- Feedback
- Domains
- Training
- Events

help
- Request an ODP account
- How to post a Content OP

contribute
- Post a Content OP
- Add a Domain
- Post your Feedback
- Post a modeling issue
- List of Categories
- List of Properties

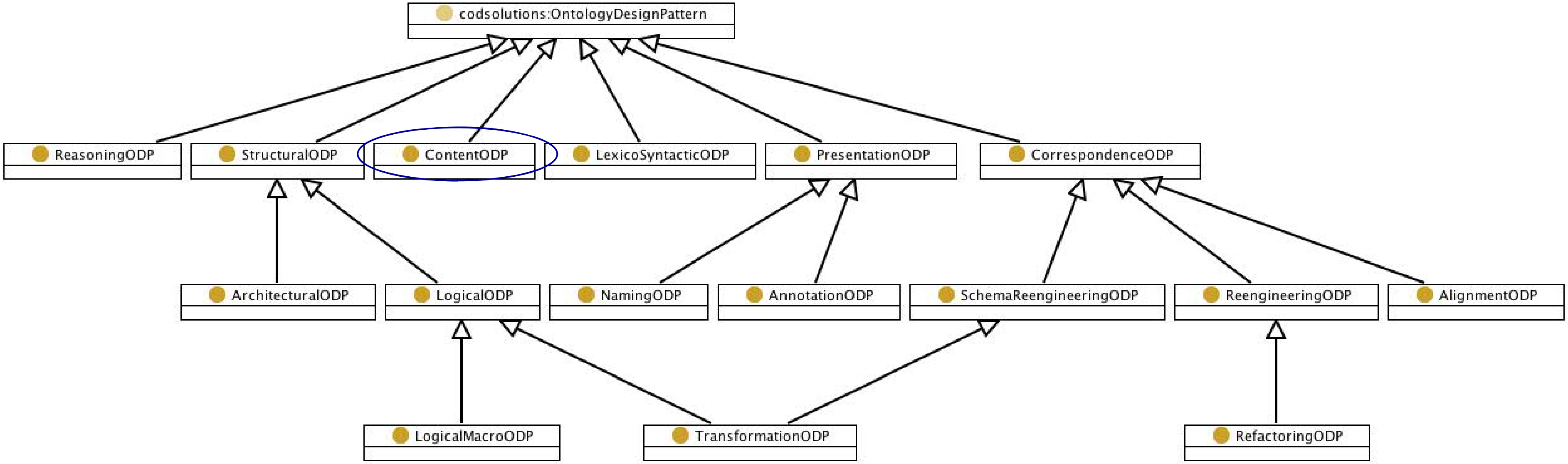
catalogues

Current taxonomy of ontology design patterns



From http://ontologydesignpatterns.org

Current taxonomy of ontology design patterns



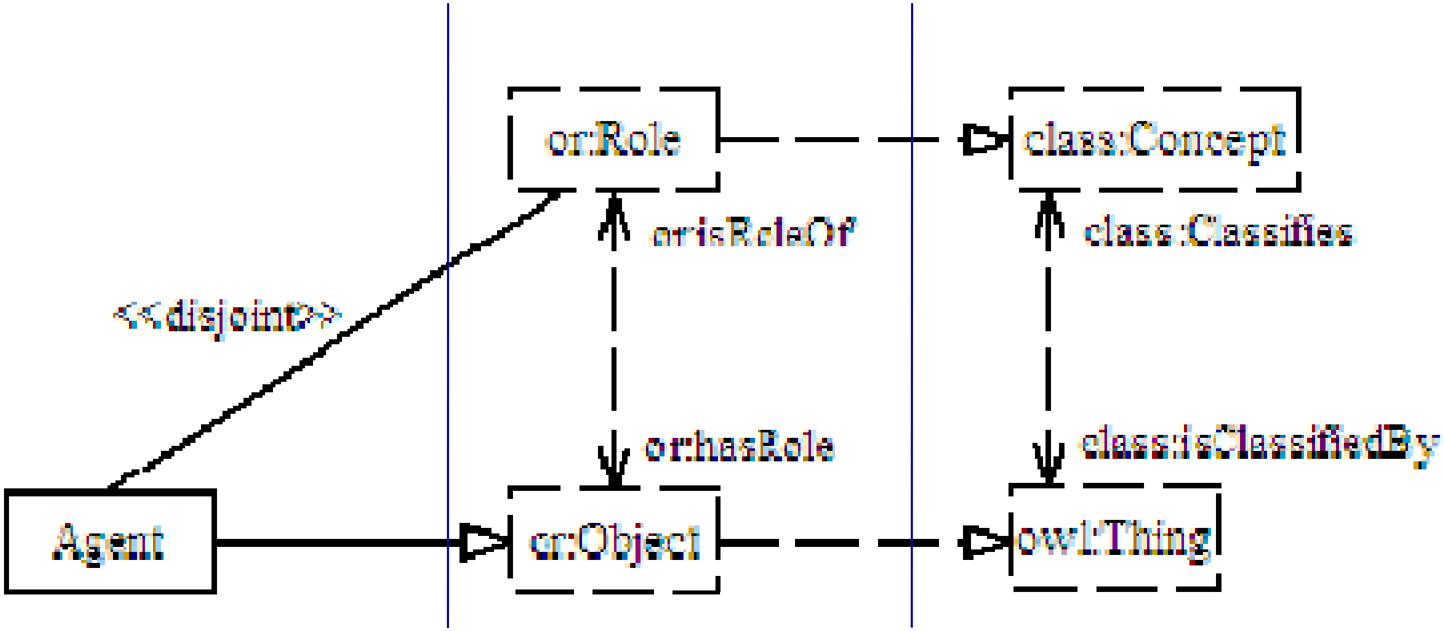
From http://ontologydesignpatterns.org

Ontology content design patterns (CPs)

- Originally conceptual models to be adapted for any particular language
- Currently small ‘micro-ontologies’ in OWL
  - Assumed to be used in the root part of a domain ontology
  - Accompanied with examples, entity lists, links to other (esp. reused)CPs

Example of pattern import+specialization

AgentRole CP ← ObjectRole CP ← Classification CP



Multi-CP modelling example

- Arnold Schwarzenegger is Shylock in the play of "Merchant of Venice”, that is given at the theater “Roma” during September and October 2009

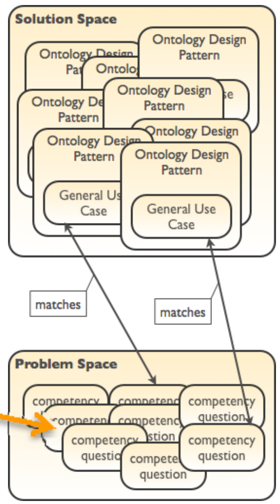
Borrowed from V. Presutti, ESWC’09 tutorial

Multi-CP modelling example

- Arnold Schwarzenegger is Shylock in the play of "Merchant of Venice”, that is given at the theater “Roma” during September and October 2009
- A person plays a character

Multi-CP modelling example

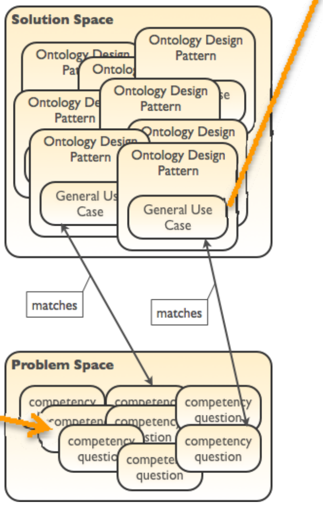
- Arnold Schwarzenegger is Shylock in the play of "Merchant of Venice", that is given at the theater “Roma” during September and October 2009
- A person plays a character

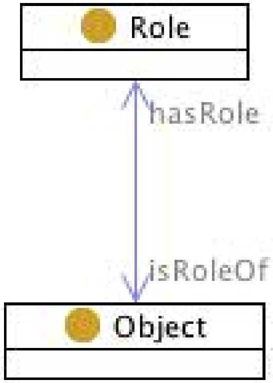


Multi-CP modelling example

- Arnold Schwarzenegger is Shylock in the play of "Merchant of Venice", that is given at the theater "Roma" during September and October 2009
- A person plays a character

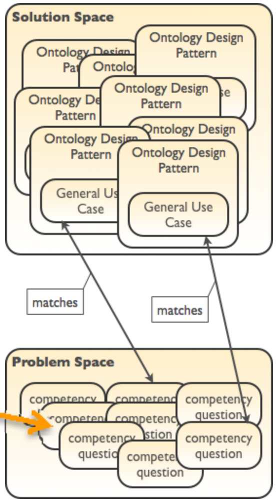
- To represents objects and the roles they play.





Multi-CP modelling example

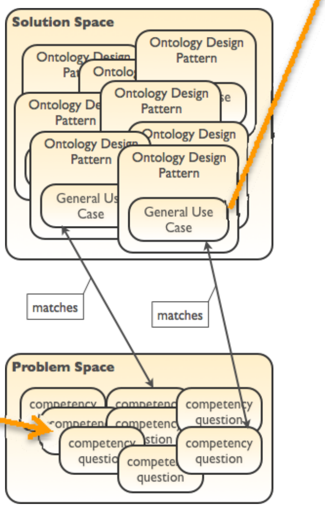
- Arnold Schwarzenegger is Shylock in the play of "Merchant of Venice", that is given at the theater "Roma" during September and October 2009
- The play of some drama

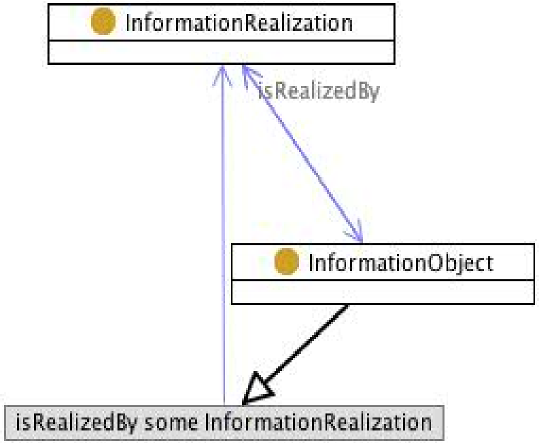


Multi-CP modelling example

- Arnold Schwarzenegger is Shylock in the play of "Merchant of Venice”, that is given at the theater “Roma” during September and October 2009
- The play of some drama

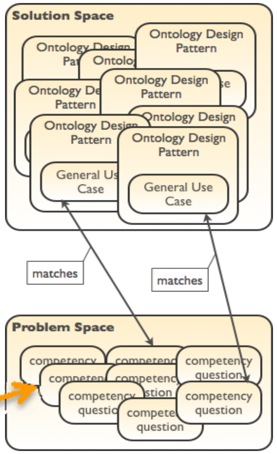
- To distinguish information objects from their concrete realizations.





Multi-CP modelling example

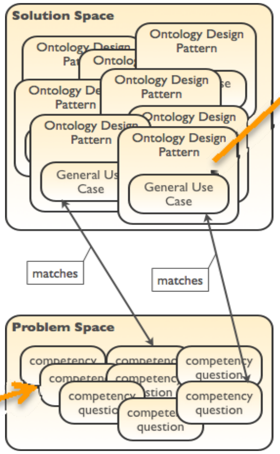
- Arnold Schwarzenegger is Shylock in the play of "Merchant of Venice", that is given at the theater "Roma" during September and October 2009
- A time period

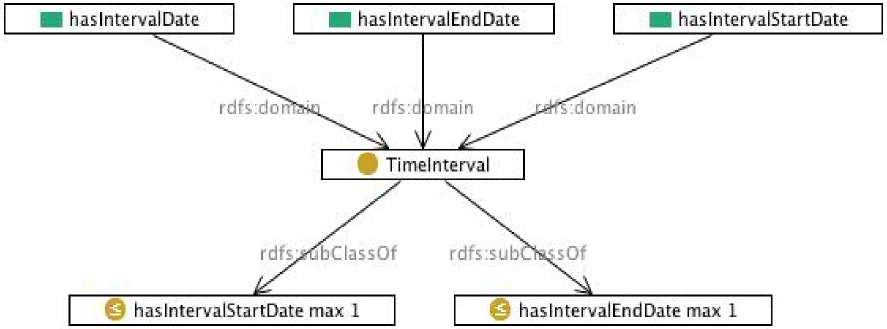


Multi-CP modelling example

- Arnold Schwarzenegger is Shylock in the play of "Merchant of Venice”, that is given at the theater “Roma” **during September and October 2009**
- **A time period**

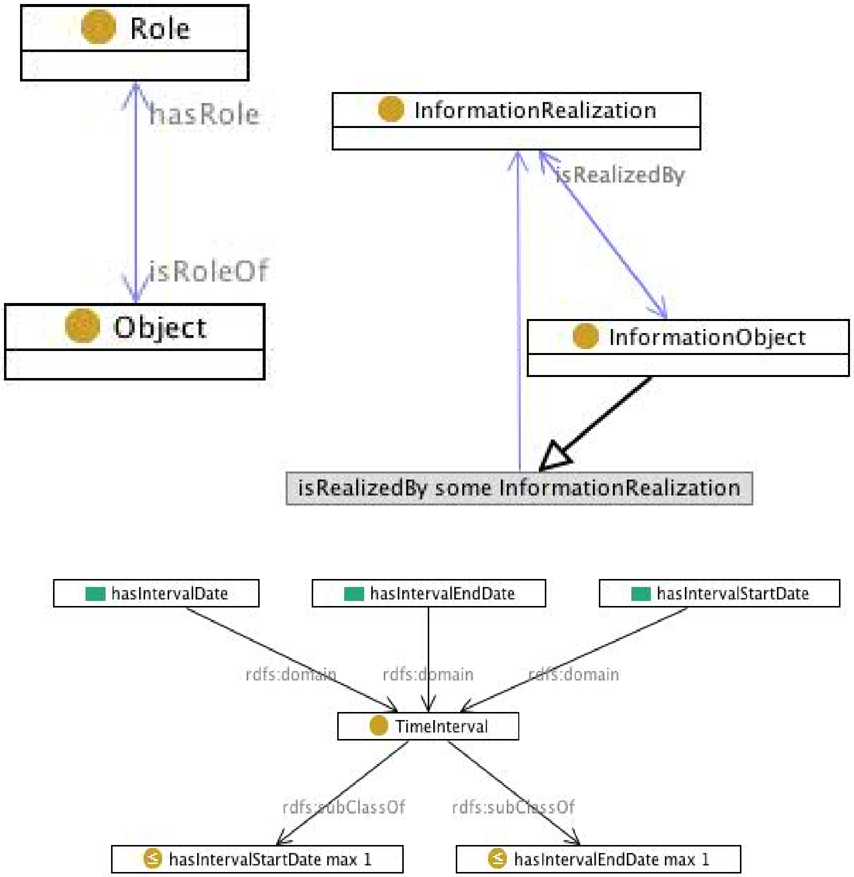
- To represent time intervals, their start/end dates, and any dates falling into the period





Multi-CP modelling example

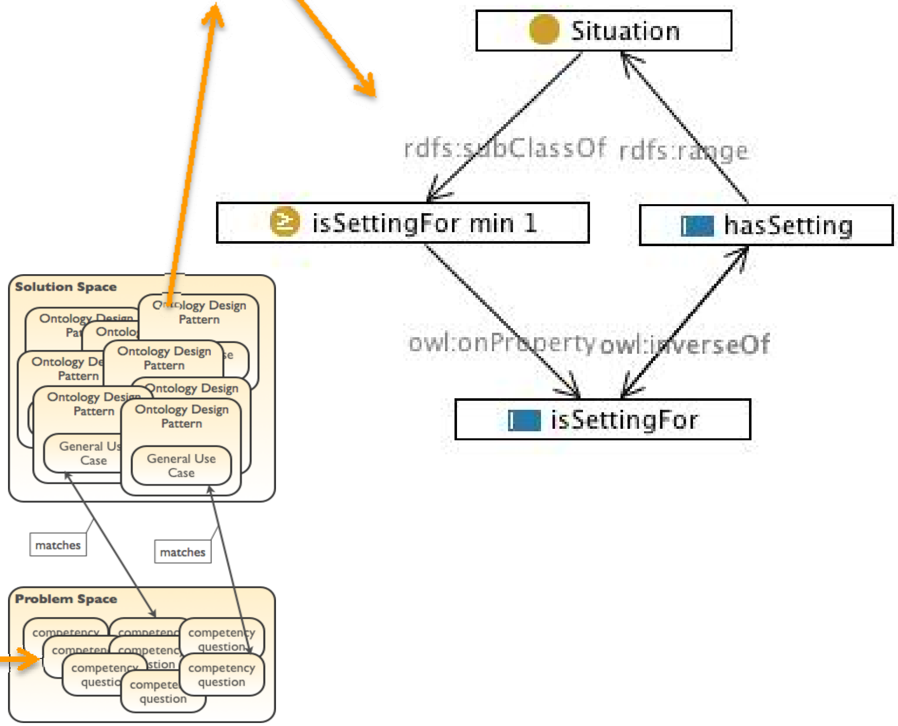
- Arnold Schwarzenegger is Shylock in the play of "Merchant of Venice”, that is given at the theater “Roma” during September and October 2009
- A person plays a character in a play of a drama, given at a theater during a time period
- How can we relate them together?



Multi-CP modelling example

- Arnold Schwarzenegger is Shylock in the play of "Merchant of Venice”, that is given at the theater “Roma” during September and October 2009
- A person plays a character in a play of a drama, given at a theater during a time period

- A situation, a set of circumstances in a defined setting



Creation of CPs

- 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)
- Mix of these

XD methodology / tools

- Developed at ISTC-CNR, Rome
  - See Presutti et al., 2009 (WOP workshop)
- Tailored for the design of small, compact task-oriented ontologies
  - Increase the development speed
  - Allow for better quality control
  - Increase the reuse potential

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
  - reuse ontology design patterns
  - evaluate the result against the task

XD Methodology in nutshell

- Step 1 – Get into the project context.
- Step 2 – Collect requirement stories.
- Step 3 – Select a story that hasn’t been treated yet.
- Step 4 – Transform the story into CQs.
- Step 5 - Select a coherent set of CQs.
- Step 6 - Match the CQs to available CPs.
- Step 7 - Select CPs to use.
- Step 8 - Reuse (import, specialize) and integrate (compose, extend) selected CPs.
- Step 9 - Unit tests, through SPARQL queries, and fix.
- Step 10 – Release the module.
- Step 11 – Integrate, test and fix.
- Step 12 – Release new version of ontology.

XDTools

- Plugin to Eclipse and to NeOn Toolkit
  - http://stlab.istc.cnr.it/stlab/XDTools
- Access to patterns in a repository
  - Browsing
  - Keyword search
- Pattern manipulation
  - Such as specialization
- Pattern annotation
- Pattern-based analysis of ontology
  - Check if best practices were followed
    - ✓ detects e.g. missing labels and comments, isolated entities, unused imported ontologies

Logical / structural ontology patterns

- Do not contain any content vocabulary
- Dependent on language (here, OWL)
- Typically several patterns clustered as different solutions for the same (or similar) modeling problem
- Cannot be directly represented in the target language, only in terms of
  - Verbal descriptions
  - Examples
  - Structures with placeholders (variables)
  - Transformations between different solutions

Examples of popular LPs

- Classes as property values (W3C)
- Normalization (Manchester)

Classes as property values

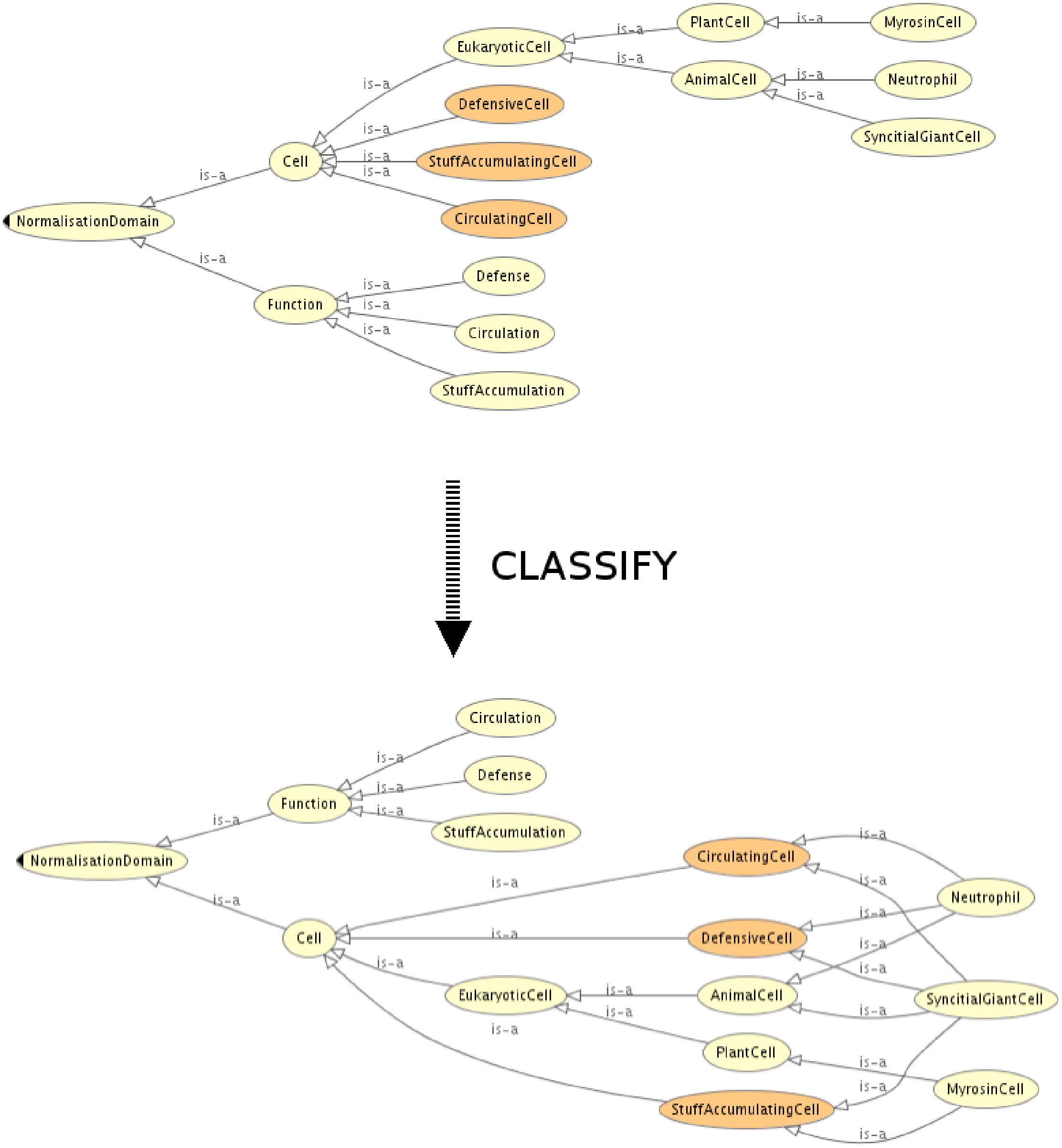
- Problem (arising from modeling heterogeneity)
  - A taxonomy is modeled in terms of classes
  - Individuals have to refer to these classes

```
AfricanLion rdfs:subclassOf Lion
LionsLifeInThePride rdf:type Book
LionsLifeInThePride dc:subject AfricanLion
```

- Solutions within OWL DL
  - Class/individual melting (OWL Full / OWL 2 punning)
  - Represent each class by its (dummy) instance
  - Represent each class by another individual
  - Use ‘subject’ as annotation property
  - Refer to an anonymous individual of a class

Normalization pattern

- T-box oriented
- Untangling polyhierarchies by replacing explici subclass links by existential definitions
- Polyhierarchy is only constructed at reasoning-time



OPPL – Ontology Pre-Processor Language

- University of Manchester
  - http://oppl2.sourceforge.net
- Tool for manipulation with OWL structures
- Pattern-based in version 2
  - Logical patterns
- Typically meant for refactoring of an ontology prior to reasoning
- Example: „Finds subclasses of NamedPizza and make them subclasses of Thing“
  - ?x:CLASS
    SELECT ?x SubClassOf NamedPizza
    BEGIN ADD ?x SubClassOf Thing END;

Naming patterns

- Consider entity names (expressed by URIs and labels) in ontologies as natural language terms
- Both design and analysis aspects are important
- Both users and applications benefit from the use of `best-practice’ naming patterns
- Naming patterns can be considered
  - at the level of indiviual entities (general naming conventions)
  - across multiple interconnected entities (cross-entity patterns leveraging on logical patterns)
- See: Svátek (2009), Schober (2009)

Naming patterns: for human user

- User-focused initiatives in ontological engineering, such as the introduction of Manchester syntax for OWL, aim to improve the readability at the level of meta-model constructions
- Naming patterns could play an analogous role of at the level of model entities

Example: T-box axiom in Manchester syntax

- Careless of naming patterns
  - `StateOwned Director only (nomination some ministry)`
- Same axiom, same syntax, but careful naming
  - `StateOwnedCompany hasDirector only (nominatedBy some Ministry)`
- What made the difference?
  - Explicitly present head noun (‘company’)
  - Avoiding plain nouns as object property names (‘director’, ‘nomination’)
  - Consistent capitalisation for same entity type

Benefits for applications

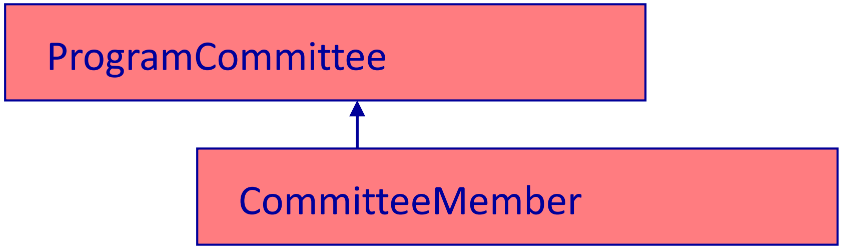
- Aside pure (deductive) logical reasoning, automated semantic processing of ontology content is needed e.g. for
  - Detection (and even suggestion of repair) of possible conceptual mismatches
  - Automated alignment and (modular) importing
  - Model transformation, e.g.
    - ✓ For better alignment
    - ✓ For better tractability by a reasoner
- Such heuristic processing typically require human assistence in selecting among alternative operations
- To reduce the number of alternatives offered to a human (or rank such alternatives), even not-too-reliable evidence, incl. entity naming, should be exploited

Benefits for applications

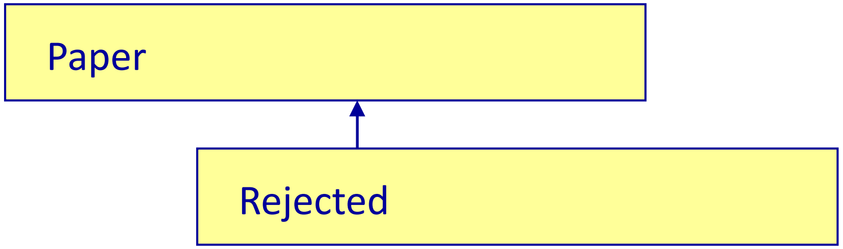
- Many conceptualisation errors are not manifested at the level of logical consistency
- Naming analysis can reveal problems that are either conceptualisation errors or awkward naming
- Example (Šváb-Zamazal, 2008): detection of lexical head incompatibility in a taxonomy
  - 40-70% precision (detection indeed pointing to a probable conceptualisation issue)
  - depends on reliable detection of thesaurus correspondence
  - seems to work best on narrow-focused ontologies with lots of (compound) technical terms

Lexical head incompatibility

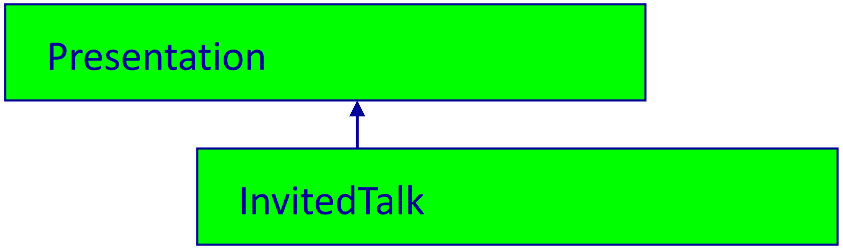
- Set-theoretic problem



- Bad naming policy
  - Would have been detected by other means



- Synonymy/Hyperonymy



Block III

PATTERN-BASED ONTOLOGY TRANSFORMATION

Agenda

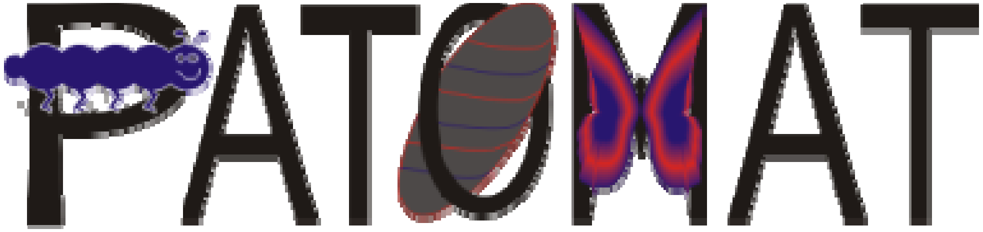
- Context and motivations
- (Ontology) Transformation patterns
  - Structure and (abstract) use
- Use cases
  - Ontology matching
  - Content pattern import
  - Special use case: FOAF ‘knows’
- Transformation workflow and implementation
- Ongoing and future work

Agenda

- Context and motivations
- (Ontology) Transformation patterns
  - Structure and (abstract) use
- Use cases
  - Ontology matching
  - Content pattern import
  - Special use case: FOAF ‘knows’
- Transformation workflow and implementation
- Ongoing and future work

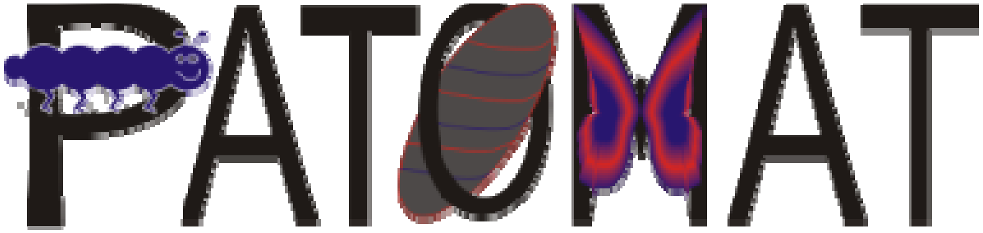
Context: PatOMat project

- Funded by the Czech Science Foundation, 2010-2012
- Central thread: „metamorphing ontologies“
  - The same conceptualisation can be expressed differently in the same language (OWL), depending on the modelling style used



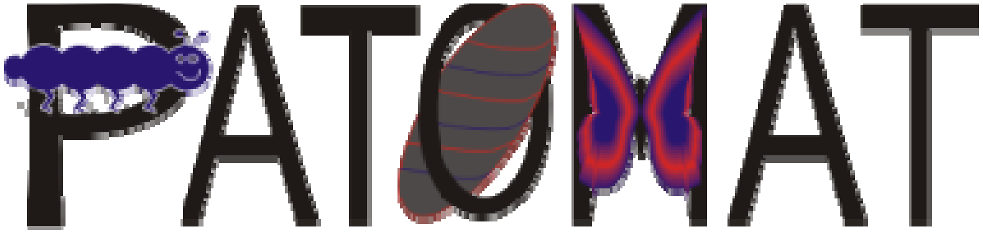
Context: PatOMat project

- Funded by the Czech Science Foundation, 2010-2012
- Central thread: „metamorphing ontologies“
  - The same conceptualisation can be expressed differently in the same language (OWL), depending on the modelling style used
  - The modelling style should (semi-)automatically adjust to current needs



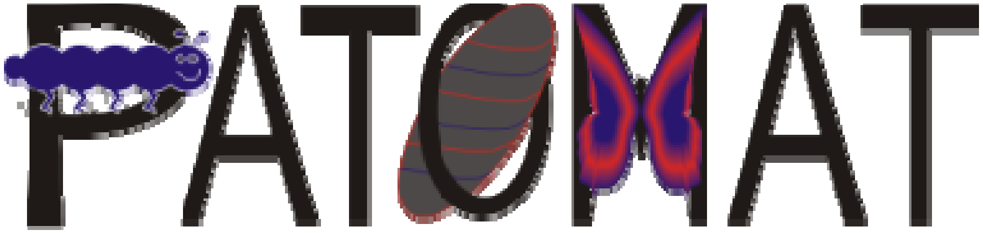
Context: PatOMat project

- Funded by the Czech Science Foundation, 2010-2012
- Central thread: „metamorphing ontologies“
  - The same conceptualisation can be expressed differently in the same language (OWL), depending on the modelling style used
  - The modelling style should (semi-)automatically adjust to current needs
  - For example, for the given ontology to smoothly map to or import another one



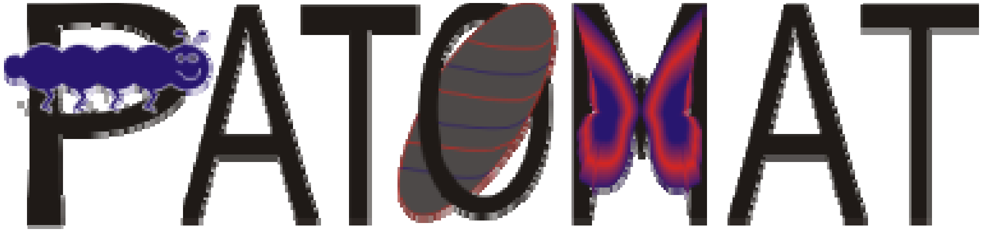
Context: PatOMat project

- Funded by the Czech Science Foundation, 2010-2012
- Central thread: „metamorphing ontologies“
  - The same conceptualisation can be expressed differently in the same language (OWL), depending on the modelling style used
  - The modelling style should (semi-)automatically adjust to current needs
  - For example, for the given ontology to smoothly map to or import another one
  - Or for removing features that make problems to a reasoner



Context: PatOMat project

- Funded by the Czech Science Foundation, 2010-2012
- Central thread: „metamorphing ontologies“
  - The same conceptualisation can be expressed differently in the same language (OWL), depending on the modelling style used
  - The modelling style should (semi-)automatically adjust to current needs
  - For example, for the given ontology to smoothly map to or import another one
  - Or for removing features that make problems to a reasoner
- http://patomat.vse.cz



Motivation: example of style heterogeneity

Notion of „acceptance/rejection of a paper at a conference“

Motivation: example of style heterogeneity

Notion of „acceptance/rejection of a paper at a conference“

- Modelling via sibling classes
  - PaperAcceptanceAct SubClassOf: ReviewerAct
  - PaperRejectionAct SubClassOf: ReviewerAct

Motivation: example of style heterogeneity

Notion of „acceptance/rejection of a paper at a conference“

- Modelling via sibling classes
  - PaperAcceptanceAct SubClassOf: ReviewerAct
  - PaperRejectionAct SubClassOf: ReviewerAct
- Modelling via object properties
  - accepts Domain: Reviewer accepts Range: Paper
  - rejects Domain: Reviewer rejects Range: Paper

Motivation: example of style heterogeneity

Notion of „acceptance/rejection of a paper at a conference“

- Modelling via sibling classes
  - PaperAcceptanceAct SubClassOf: ReviewerAct
  - PaperRejectionAct SubClassOf: ReviewerAct
- Modelling via object properties
  - accepts Domain: Reviewer   accepts Range: Paper
  - rejects Domain: Reviewer   rejects Range: Paper
- Modelling via enumeration class, i.e. individuals
  - reviewerDecision Domain: Paper
  - reviewerDecision Range: (EquivalentTo {acceptance, rejection})

Motivation: example of style heterogeneity

Notion of „acceptance/rejection of a paper at a conference“

- Modelling via sibling classes
  - PaperAcceptanceAct SubClassOf: ReviewerAct
  - PaperRejectionAct SubClassOf: ReviewerAct
- Modelling via object properties
  - accepts Domain: Reviewer accepts Range: Paper
  - rejects Domain: Reviewer rejects Range: Paper
- Modelling via enumeration class, i.e. individuals
  - reviewerDecision Domain: Paper
  - reviewerDecision Range: (EquivalentTo {acceptance, rejection})

- Similar setting but slightly more higher-level than (SPARQL-based) EvoPat or R2R are meant for?

PatOMat and patterns

PatOMat and patterns

- Alternative modelling styles are captured via (logical/structural) ontology patterns: OWL structures (mostly) containing placeholders instead of real entities
  - source OP
  - target OP

PatOMat and patterns

- Alternative modelling styles are captured via (logical/structural) ontology patterns: OWL structures (mostly) containing placeholders instead of real entities
  - source OP
  - target OP
- Transformation of (occurrences of) one OP into another is defined by a transformation pattern
  - namely, in its pattern transformation (PT) part

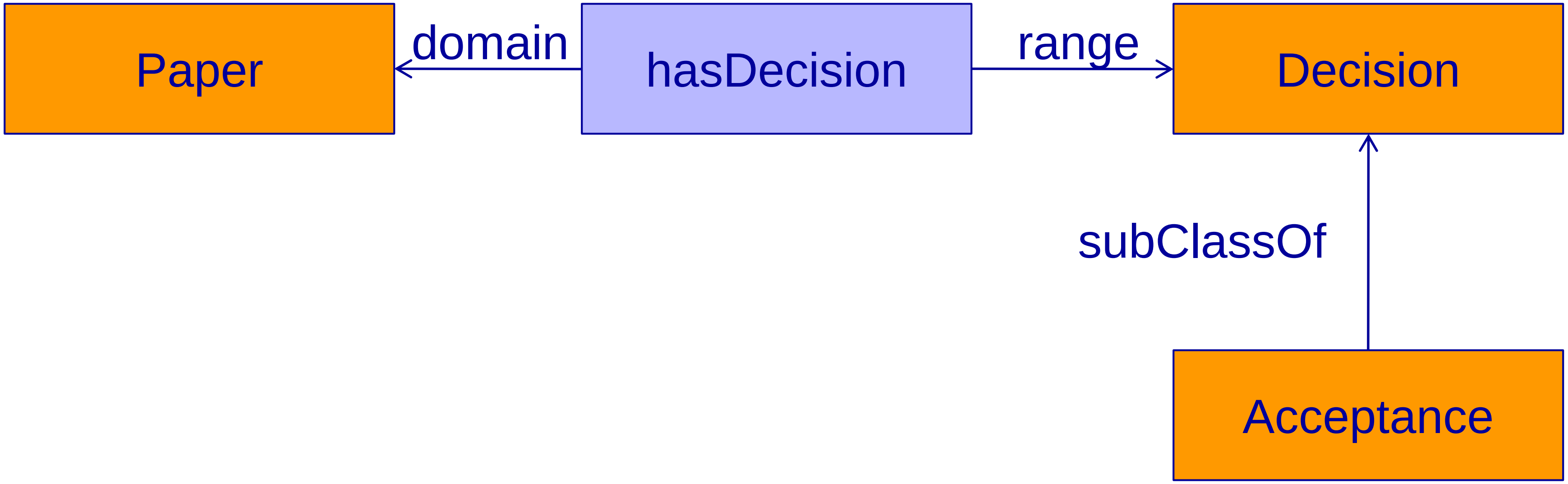
PatOMat and patterns

- Alternative modelling styles are captured via (logical/structural) ontology patterns: OWL structures (mostly) containing placeholders instead of real entities
  - source OP
  - target OP
- Transformation of (occurrences of) one OP into another is defined by a transformation pattern
  - namely, in its pattern transformation (PT) part
- Both ontology patterns and transformation patterns may contain naming patterns with linguistic grounding
  - naming detection patterns
  - naming transformation patterns

Agenda

- Context and motivations
- (Ontology) Transformation patterns
  - Structure and (abstract) use
- Use cases
  - Ontology matching
  - Content pattern import
  - Special use case: FOAF ‘knows’
- Transformation workflow and implementation
- Ongoing and future work

Example of source fragment



Example of transformation pattern

- OP1 : E={Class: ?A, Class: ?B, Class: ?C, ObjectProperty: ?p}, Ax={?p Domain: ?A, ?p Range: ?B, ?C SubClassOf: ?B}, NDP={comparison(?B, head term(?p)), exists(verb form(?C))}

- OP2 : E={Class: ?D, Class: ?E, Class: ?F, Class: ?G, ObjectProperty: ?q}, Ax={?q Domain: ?D, ?q Range: ?E, ?F SubClassOf: ?E, ?G EquivalentTo: (?q some ?F)}

- PT : LI={?A EquivalentTo: ?D, ?B EquivalentTo: ?E, ?C EquivalentTo: ?F, EquivalentProperties: ?p, ?q}, NTP= {(?G, make passive verb(?C) + head noun(?A))}.

transformation pattern

Example of transformation pattern

source ont. patt.

- OP1 : E={Class: ?A, Class: ?B, Class: ?C, ObjectProperty: ?p}, Ax={?p Domain: ?A, ?p Range: ?B, ?C SubClassOf: ?B}, NDP={comparison(?B, head term(?p)), exists(verb form(?C))}

target ont. patt.

- OP2 : E={Class: ?D, Class: ?E, Class: ?F, Class: ?G, ObjectProperty: ?q}, Ax={?q Domain: ?D, ?q Range: ?E, ?F SubClassOf: ?E, ?G EquivalentTo: (?q some ?F)}

- PT : LI={?A EquivalentTo: ?D, ?B EquivalentTo: ?E, ?C EquivalentTo: ?F, EquivalentProperties: ?p, ?q}, NTP= {(?G, make passive verb(?C) + head noun(?A))}.

transformation pattern

Example of transformation pattern

source ont. patt.

- OP1 : E={Class: ?A, Class: ?B, Class: ?C, ObjectProperty: ?p},
  Ax={?p Domain: ?A, ?p Range: ?B, ?C SubClassOf: ?B},
  NDP={comparison(?B, head term(?p)), exists(verb form(?C))}

naming detection pattern

target ont. patt.

- OP2 : E={Class: ?D, Class: ?E, Class: ?F, Class: ?G,
  ObjectProperty: ?q},
  Ax={?q Domain: ?D, ?q Range: ?E, ?F SubClassOf: ?E,
  ?G EquivalentTo: (?q some ?F)}

- PT : LI={?A EquivalentTo: ?D, ?B EquivalentTo: ?E,
  ?C EquivalentTo: ?F, EquivalentProperties: ?p, ?q},
  NTP= {(?G, make passive verb(?C) + head noun(?A))}.

naming transformation pattern

transformation pattern

Example of transformation pattern

hasDecision Domain: Paper
hasDecision Range: Decision
Acceptance SubClassOf: Decision

source ont. patt.

- OP1 : E={Class: ?A, Class: ?B, Class: ?C, ObjectProperty: ?p},
  Ax={?p Domain: ?A, ?p Range: ?B, ?C SubClassOf: ?B},
  NDP={comparison(?B, head term(?p)), exists(verb form(?C))}

naming detection pattern

target ont. patt.

- OP2 : E={Class: ?D, Class: ?E, Class: ?F, Class: ?G, ObjectProperty: ?q},
  Ax={?q Domain: ?D, ?q Range: ?E, ?F SubClassOf: ?E,
  ?G EquivalentTo: (?q some ?F)}

- PT : LI={?A EquivalentTo: ?D, ?B EquivalentTo: ?E,
  ?C EquivalentTo: ?F, EquivalentProperties: ?p, ?q},
  NTP= {(?G, make passive verb(?C) + head noun(?A))}.

naming transformation pattern

transformation pattern

Example of transformation pattern

hasDecision Domain: Paper
hasDecision Range: Decision
Acceptance SubClassOf: Decision

Paper Decision Acceptance hasDecision

- OP1 : E={Class: ?A, Class: ?B, Class: ?C, ObjectProperty: ?p}, Ax={?p Domain: ?A, ?p Range: ?B, ?C SubClassOf: ?B}, NDP={comparison(?B, head term(?p)), exists(verb form(?C))}
- OP2 : E={Class: ?D, Class: ?E, Class: ?F, Class: ?G, ObjectProperty: ?q}, Ax={?q Domain: ?D, ?q Range: ?E, ?F SubClassOf: ?E, ?G EquivalentTo: (?q some ?F)}
- PT : LI={?A EquivalentTo: ?D, ?B EquivalentTo: ?E, ?C EquivalentTo: ?F, EquivalentProperties: ?p, ?q}, NTP= {(?G, make passive verb(?C) + head noun(?A))}.

Example of transformation pattern

hasDecision Domain: Paper
hasDecision Range: Decision
Acceptance SubClassOf: Decision

Paper Decision Acceptance hasDecision

- OP1 : E={Class: ?A, Class: ?B, Class: ?C, ObjectProperty: ?p},
  Ax={?p Domain: ?A, ?p Range: ?B, ?C SubClassOf: ?B},
  NDP={comparison(?B, head term(?p)), exists(verb form(?C))}
  ‘Decision’=‘Decision’ accept (according to WordNet)
- OP2 : E={Class: ?D, Class: ?E, Class: ?F, Class: ?G, ObjectProperty: ?q},
  Ax={?q Domain: ?D, ?q Range: ?E, ?F SubClassOf: ?E, ?G EquivalentTo: (?q some ?F)}
- PT : LI={?A EquivalentTo: ?D, ?B EquivalentTo: ?E, ?C EquivalentTo: ?F, EquivalentProperties: ?p, ?q},
  NTP= {(?G, make passive verb(?C) + head noun(?A))}.

Example of transformation pattern

hasDecision Domain: Paper
hasDecision Range: Decision
Acceptance SubClassOf: Decision

Paper Decision Acceptance hasDecision

- OP1 : E={Class: ?A, Class: ?B, Class: ?C, ObjectProperty: ?p},
  Ax={?p Domain: ?A, ?p Range: ?B, ?C SubClassOf: ?B},
  NDP={comparison(?B, head term(?p)), exists(verb form(?C))}
  ‘Decision’=‘Decision’ accept (according to WordNet)

- OP2 : E={Class: ?D, Class: ?E, Class: ?F, Class: ?G, ObjectProperty: ?q},
  Ax={?q Domain: ?D, ?q Range: ?E, ?F SubClassOf: ?E,
  ?G EquivalentTo: (?q some ?F)}

- PT : LI={?A EquivalentTo: ?D, ?B EquivalentTo: ?E,
  ?C EquivalentTo: ?F, EquivalentProperties: ?p, ?q},
  NTP= {(?G, make passive verb(?C) + head noun(?A))}.
  accepted Paper

Example of transformation pattern

hasDecision Domain: Paper
hasDecision Range: Decision
Acceptance SubClassOf: Decision

Paper Decision Acceptance hasDecision

- OP1 : E={Class: ?A, Class: ?B, Class: ?C, ObjectProperty: ?p},
  Ax={?p Domain: ?A, ?p Range: ?B, ?C SubClassOf: ?B},
  NDP={comparison(?B, head term(?p)), exists(verb form(?C))}
  ‘Decision’=‘Decision’ accept (according to WordNet)

  Paper Decision Acceptance AcceptedPaper
- OP2 : E={Class: ?D, Class: ?E, Class: ?F, Class: ?G,
  ObjectProperty: ?q},hasDecision
  Ax={?q Domain: ?D, ?q Range: ?E, ?F SubClassOf: ?E,
  ?G EquivalentTo: (?q some ?F)}

- PT : LI={?A EquivalentTo: ?D, ?B EquivalentTo: ?E,
  ?C EquivalentTo: ?F, EquivalentProperties: ?p, ?q},
  NTP= {(?G, make passive verb(?C) + head noun(?A))}.
  accepted Paper

Example of transformation pattern

hasDecision Domain: Paper
hasDecision Range: Decision
Acceptance SubClassOf: Decision

Paper Decision Acceptance hasDecision

- OP1 : E={Class: ?A, Class: ?B, Class: ?C, ObjectProperty: ?p},
  Ax={?p Domain: ?A, ?p Range: ?B, ?C SubClassOf: ?B},
  NDP={comparison(?B, head term(?p)), exists(verb form(?C))}
  ‘Decision’=‘Decision’ accept (according to WordNet)

  Paper Decision Acceptance AcceptedPaper

- OP2 : E={Class: ?D, Class: ?E, Class: ?F, Class: ?G,
  ObjectProperty: ?q},hasDecision
  Ax={?q Domain: ?D, ?q Range: ?E, ?F SubClassOf: ?E,
  ?G EquivalentTo: (?q some ?F)} AcceptedPaper = hasDecision some Acceptance

- PT : LI={?A EquivalentTo: ?D, ?B EquivalentTo: ?E,
  ?C EquivalentTo: ?F, EquivalentProperties: ?p, ?q},
  NTP= {(?G, make passive verb(?C) + head noun(?A))}.
  accepted Paper

Agenda

- Context and motivations
- (Ontology) Transformation patterns
  - Structure and (abstract) use
- Use cases
  - Ontology matching
  - Content pattern import
  - Special use case: FOAF ‘knows’
- Transformation workflow and implementation
- Ongoing and future work

Use cases

- ‘Smoother’ matching of style-wise heterogeneous ontologies

Use cases

- ‘Smoother’ matching of style-wise heterogeneous ontologies
  - Alternative to building complex ‘Mannheim-style’ correspondences, such as
    - ✓ $\text{AcceptedPaper}_2 = \text{hasDecision}_1 \text{ some } \text{Acceptance}_1$

Use cases

- ‘Smoother’ matching of style-wise heterogeneous ontologies
  - Alternative to building complex ‘Mannheim-style’ correspondences, such as
    - ✓ $\text{AcceptedPaper}_2 = \text{hasDecision}_1 \text{ some } \text{Acceptance}_1$
  - Or, complex correspondences can be built ex post by composing two pieces of correspondence into one
    - ✓ $\text{AcceptedPaper}_1\text{'} = \text{hasDecision}_1 \text{ some } \text{Acceptance}_1$
    - ✓ $\text{AcceptedPaper}_2 = \text{AcceptedPaper}_1\text{'}$

Use cases

- ‘Smoother’ matching of style-wise heterogeneous ontologies
  - Alternative to building complex ‘Mannheim-style’ correspondences, such as
    - ✓ AcceptedPaper_2 = hasDecision_1 some Acceptance_1
  - Or, complex correspondences can be built ex post by composing two pieces of correspondence into one
    - ✓ AcceptedPaper_1’ = hasDecision_1 some Acceptance_1
    - ✓ AcceptedPaper_2 = AcceptedPaper_1’
- Solving structural problems when importing an ontology into another

Use cases

- ‘Smoother’ matching of style-wise heterogeneous ontologies
  - Alternative to building complex ‘Mannheim-style’ correspondences, such as
    - ✓ $AcceptedPaper_2$ = $hasDecision_1$ some $Acceptance_1$
  - Or, complex correspondences can be built ex post by composing two pieces of correspondence into one
    - ✓ $AcceptedPaper_1$’ = $hasDecision_1$ some $Acceptance_1$
    - ✓ $AcceptedPaper_2$ = $AcceptedPaper_1$’
- Solving structural problems when importing an ontology into another
  - Currently investigated for content patterns (CPs) from the OntologyDesignPatterns.org portal

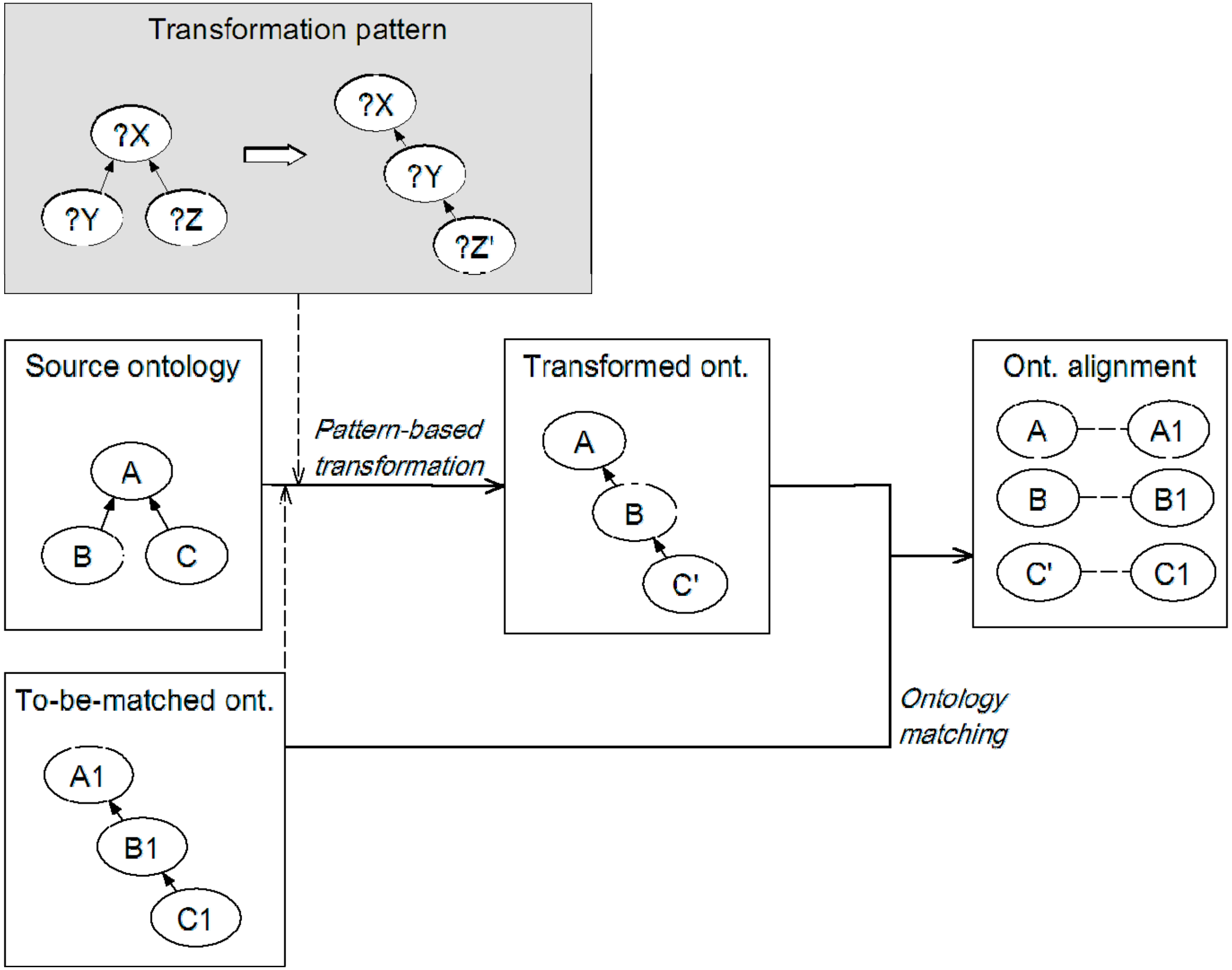
Use cases

- ‘Smoother’ matching of style-wise heterogeneous ontologies
  - Alternative to building complex ‘Mannheim-style’ correspondences, such as
    - ✓ $AcceptedPaper_2$ = $hasDecision_1$ some $Acceptance_1$
  - Or, complex correspondences can be built ex post by composing two pieces of correspondence into one
    - ✓ $AcceptedPaper_1$’ = $hasDecision_1$ some $Acceptance_1$
    - ✓ $AcceptedPaper_2$ = $AcceptedPaper_1$’
- Solving structural problems when importing an ontology into another
  - Currently investigated for content patterns (CPs) from the OntologyDesignPatterns.org portal

Use cases

- ‘Smoother’ matching of style-wise heterogeneous ontologies
  - Alternative to building complex ‘Mannheim-style’ correspondences, such as
    - ✓ AcceptedPaper$_2$ = hasDecision$_1$ some Acceptance$_1$
  - Or, complex correspondences can be built ex post by composing two pieces of correspondence into one
    - ✓ AcceptedPaper$_1$’ = hasDecision$_1$ some Acceptance$_1$
    - ✓ AcceptedPaper$_2$ = AcceptedPaper$_1$’
- Solving structural problems when importing an ontology into another
  - Currently investigated for content patterns (CPs) from the OntologyDesignPatterns.org portal
- Canonically reducing complexity for reasoners by `transforming away’ less palatable constructs

“Transformation for Matching” Scenario



Possible Workflow for Data Mediation Tasks

Given a source ontology O_1 and a to-be-matched ontology O_2, with associated instance pools

Possible Workflow for Data Mediation Tasks

Given a source ontology O_1 and a to-be-matched ontology O_2, with associated instance pools

1. Detect content compatibility of O_1 and O_2

Possible Workflow for Data Mediation Tasks

Given a source ontology O_1 and a to-be-matched ontology O_2, with associated instance pools

1. Detect content compatibility of O_1 and O_2
2. Detect style discrepancy of O_1 and O_2

Possible Workflow for Data Mediation Tasks

Given a source ontology O_1 and a to-be-matched ontology O_2, with associated instance pools

1. Detect content compatibility of O_1 and O_2
2. Detect style discrepancy of O_1 and O_2
3. Identify relevant transformation pattern/s TP
   - ✓ in terms of contentwise matcheable ontology patterns in O_1 and O_2

Possible Workflow for Data Mediation Tasks

Given a source ontology O_1 and a to-be-matched ontology O_2, with associated instance pools

1. Detect content compatibility of O_1 and O_2
2. Detect style discrepancy of O_1 and O_2
3. Identify relevant transformation pattern/s TP
   - ✓ in terms of contentwise matcheable ontology patterns in O_1 and O_2
4. Identify the boundaries of relevant fragment of source ontology

Possible Workflow for Data Mediation Tasks

Given a source ontology O_1 and a to-be-matched ontology O_2, with associated instance pools

1. Detect content compatibility of O_1 and O_2
2. Detect style discrepancy of O_1 and O_2
3. Identify relevant transformation pattern/s TP
   - ✓ in terms of contentwise matcheable ontology patterns in O_1 and O_2
4. Identify the boundaries of relevant fragment of source ontology
5. Transform O_1 to O_1’ using TP

Possible Workflow for Data Mediation Tasks

Given a source ontology O_1 and a to-be-matched ontology O_2, with associated instance pools

1. Detect content compatibility of O_1 and O_2
2. Detect style discrepancy of O_1 and O_2
3. Identify relevant transformation pattern/s TP
   - ✓ in terms of contentwise matcheable ontology patterns in O_1 and O_2
4. Identify the boundaries of relevant fragment of source ontology
5. Transform O_1 to O_1' using TP
6. Align O_1' to O_2, yielding an ontology alignment OA

Possible Workflow for Data Mediation Tasks

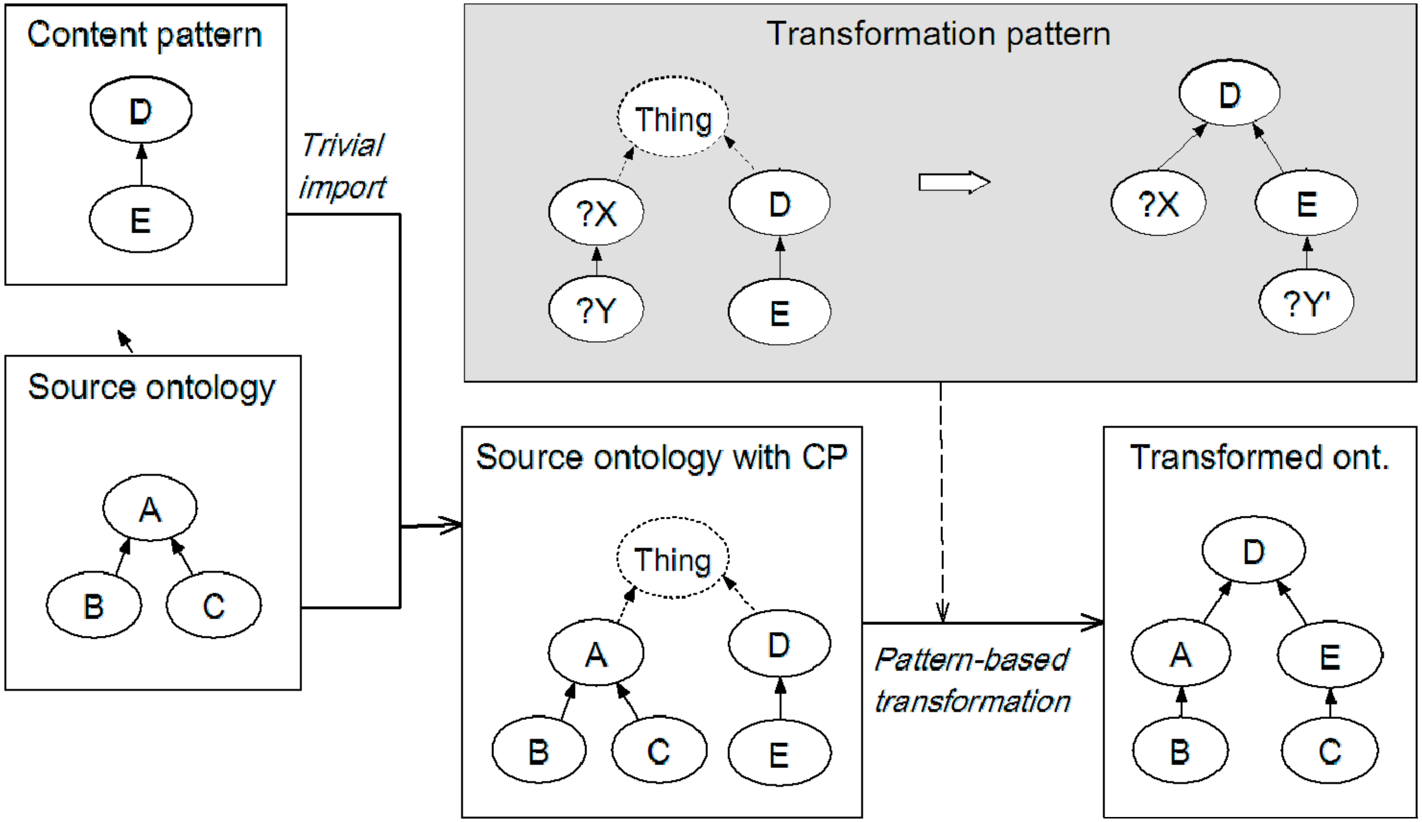
Given a source ontology O_1 and a to-be-matched ontology O_2, with associated instance pools

1. Detect content compatibility of O_1 and O_2
2. Detect style discrepancy of O_1 and O_2
3. Identify relevant transformation pattern/s TP
   - ✓ in terms of contentwise matcheable ontology patterns in O_1 and O_2
4. Identify the boundaries of relevant fragment of source ontology
5. Transform O_1 to O_1' using TP
6. Align O_1' to O_2, yielding an ontology alignment OA
7. Mediate (query/merge) instances over two-tiered links
   - ✓ Links between O_1 and O_1' built according to TP
   - ✓ Correspondences from OA

Agenda

- Context and motivations
- (Ontology) Transformation patterns
  - Structure and (abstract) use
- Use cases
  - Ontology matching
  - Content pattern import
  - Special use case: FOAF ‘knows’
- Transformation workflow and implementation
- Ongoing and future work

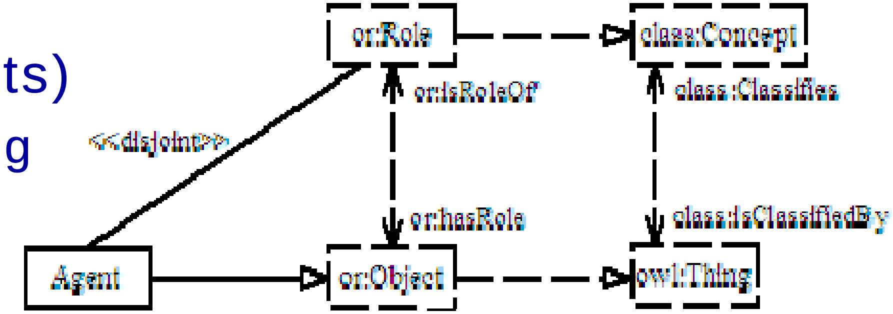
„Content pattern importing“ scenario



Example: Importing AgentRole content pattern

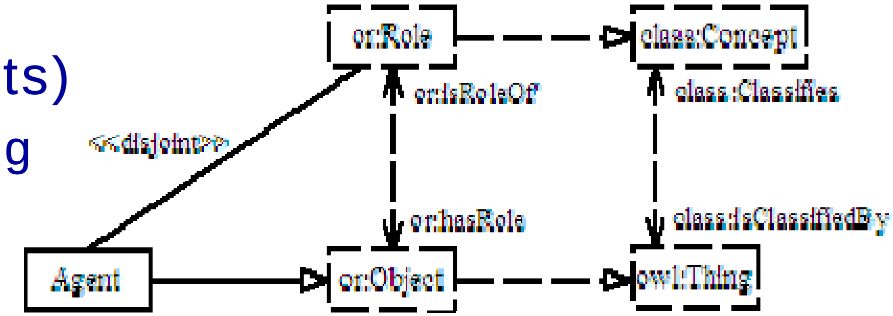
Example: Importing AgentRole content pattern

- AgentRole (with own imports)
  - OntologyDesignPatterns.org portal

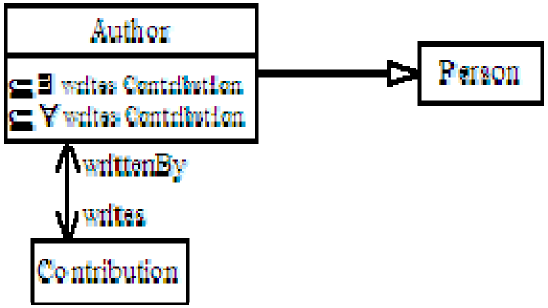


Example: Importing AgentRole content pattern

- AgentRole (with own imports)
  - OntologyDesignPatterns.org portal

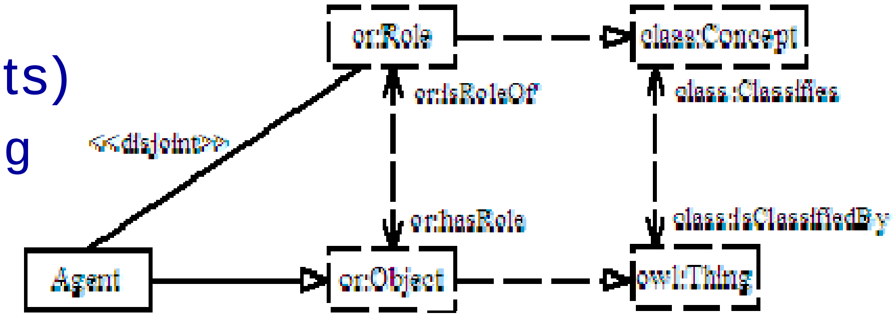


- (Fragment of) ConfOf ontology from OntoFarm collection
  - modelling the ‘conference organisation’ domain

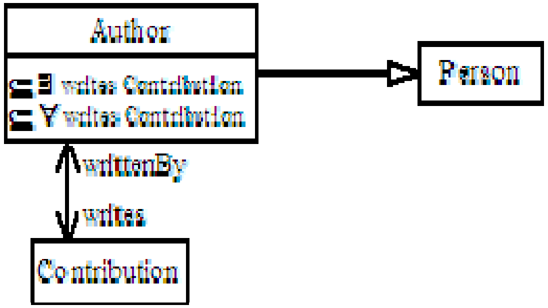


Example: Importing AgentRole content pattern

- AgentRole (with own imports)
  - OntologyDesignPatterns.org portal



- (Fragment of) ConfOf ontology from OntoFarm collection
  - modelling the 'conference organisation' domain



- Need for adaptation: we should be able to say that a person has the role of author (rather than just 'is author')

Dimensions of the ‘import’ transformation

Dimensions of the ‘import’ transformation

- Source logical pattern in the ontology to be transformed
  - ‘class-centric’ modelling approach in ConfOf

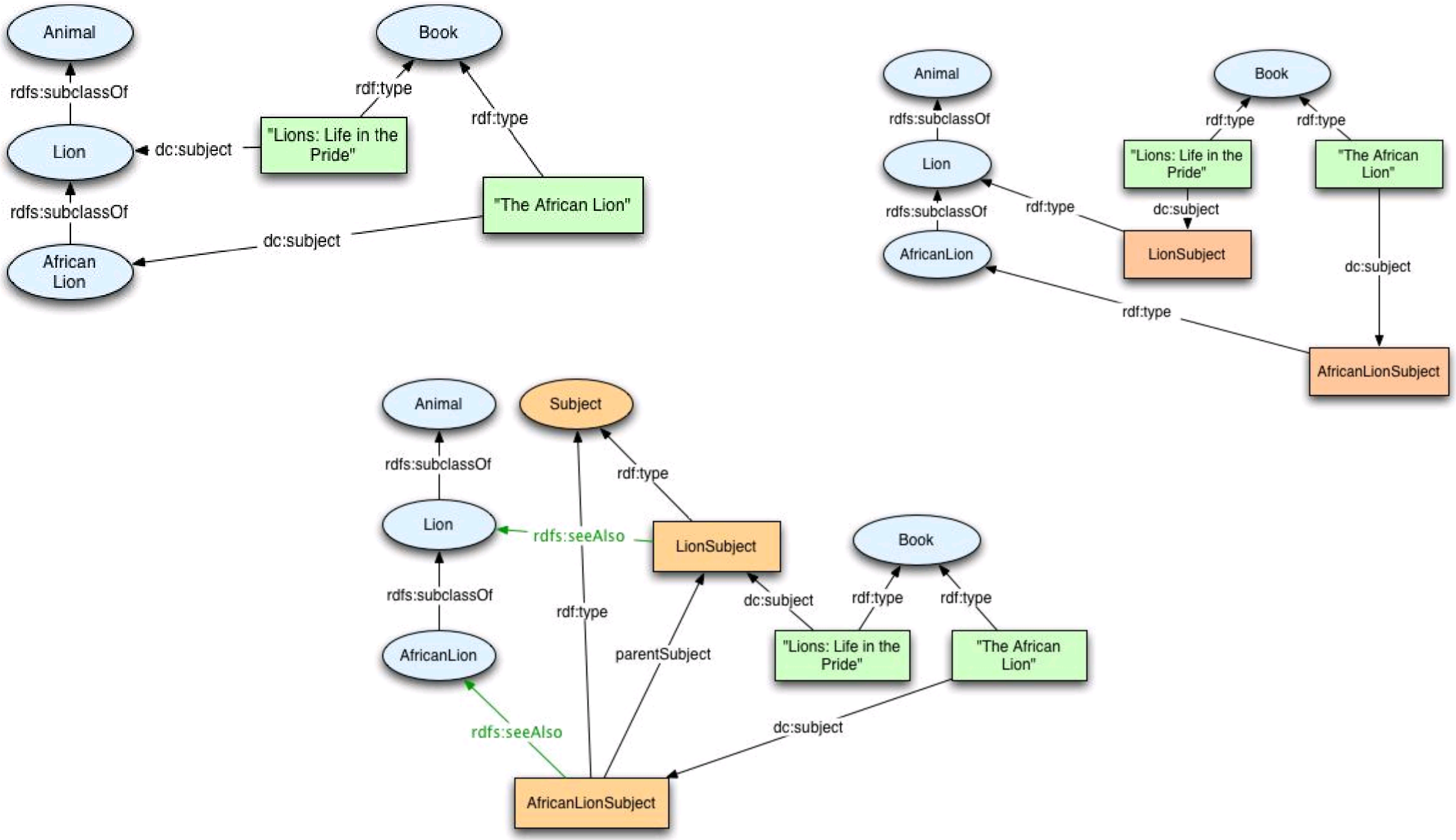
Dimensions of the ‘import’ transformation

- Source logical pattern in the ontology to be transformed
  - ‘class-centric’ modelling approach in ConfOf
- Additional axioms referring to to-be-affected entities from the source pattern
  - e.g. local and global restrictions over the ‘writes’ property

Dimensions of the ‘import’ transformation

- Source logical pattern in the ontology to be transformed
  - ‘class-centric’ modelling approach in ConfOf
- Additional axioms referring to to-be-affected entities from the source pattern
  - e.g. local and global restrictions over the ‘writes’ property
- Target logical pattern
  - only partly constrained by the content pattern
  - alternatives may well map on the ‘approaches’ in the notes published by the W3C SWBPD group
  - In the example, as we have to transform the subclassOf relationship (Author-Person) to a property relationship between instances of a natural class (Person) and a ‘role’ class, Classes as Property Values pattern is relevant

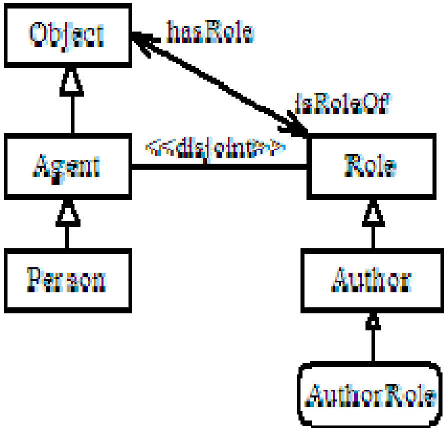
‘Approaches’ for ‘Classes as Property Values’



CPW pattern applied for AgentRole import

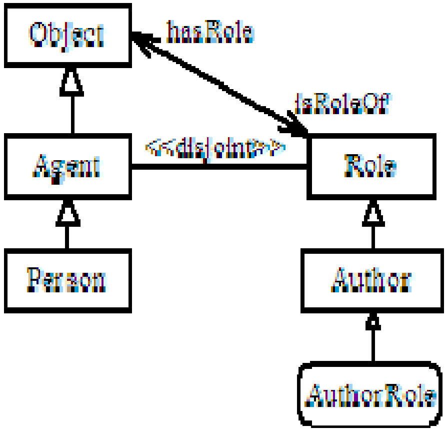
CPW pattern applied for AgentRole import

- Approach 2: Create special instances of the class to be used as property values

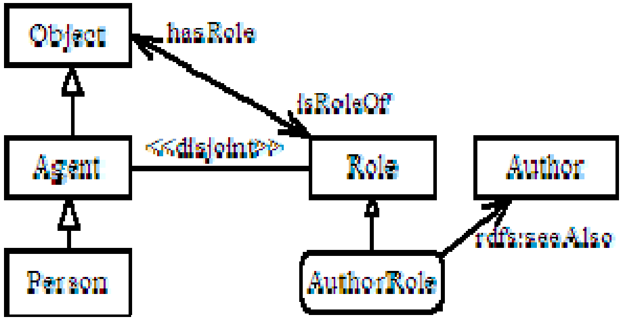


CPW pattern applied for AgentRole import

- Approach 2: Create special instances of the class to be used as property values

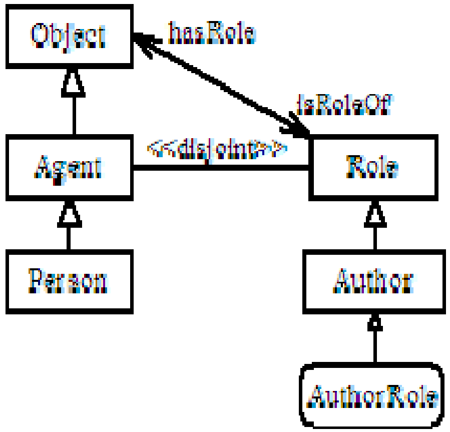


- Approach 3: Create a parallel hierarchy of instances as property values

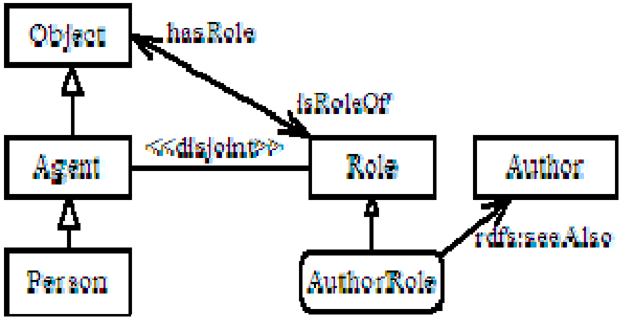


CPW pattern applied for AgentRole import

- Approach 2: Create special instances of the class to be used as property values



- Approach 3: Create a parallel hierarchy of instances as property values



- and other…

Agenda

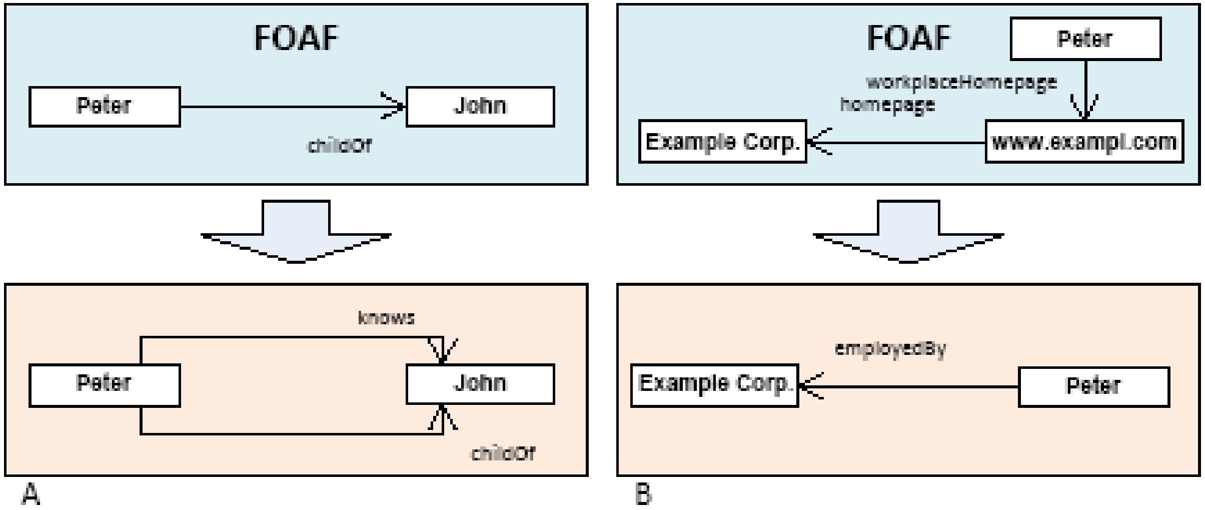
- Context and motivations
- (Ontology) Transformation patterns
  - Structure and (abstract) use
- Use cases
  - Ontology matching
  - Content pattern import
  - Special use case: FOAF ‘knows’
- Transformation workflow and implementation
- Ongoing and future work

FOAF ‘knows’ use case for pattern transformation

- Initial study from ‘foundational’ perspective
- foaf:knows is probably the most prominent representative of object property on the Web of Data
- Object properties are most interesting as bridges to substantial conceptual (ontological) modelling
- Analysis (Vacura, 2010) of
  - Adding implicit relationships
  - Relation expansion

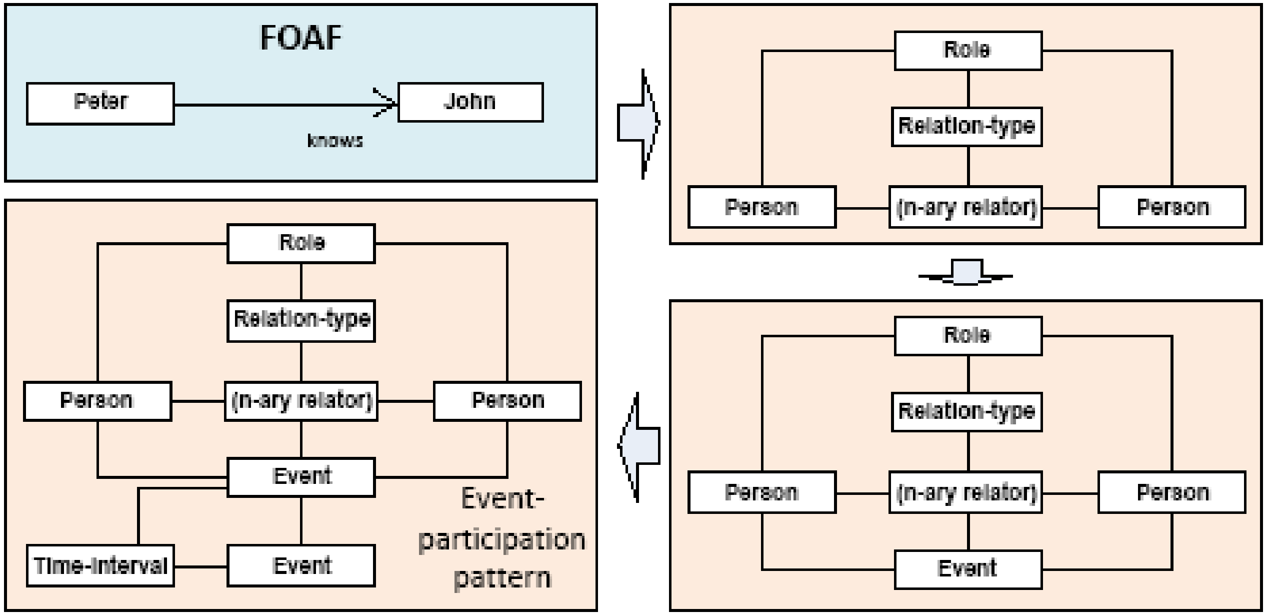
Adding implicit relationships

- May be needed to guarantee integration into other ontologies that do not share the same assumptions



Relation expansion ('unfolding') paths

- May be used to disambiguate the specific semantic of an entity as conventionally used by a LD source



Agenda

- Context and motivations
- (Ontology) Transformation patterns
  - – Structure and (abstract) use
- Use cases
  - – Ontology matching
  - – Content pattern import
  - – Special use case: FOAF ‘knows’
- Transformation workflow and implementation
- Ongoing and future work

Prototype implementation

- Three-phase transformation

Prototype implementation

- Three-phase transformation
  - detection of source pattern in ontology

Prototype implementation

- Three-phase transformation
  - detection of source pattern in ontology
  - generation of transformation instructions
    - ✓ instantiation of the transformation part of the pattern

Prototype implementation

- Three-phase transformation
  - detection of source pattern in ontology
  - generation of transformation instructions
    - ✓ instantiation of the transformation part of the pattern
  - actual transformation
    - ✓ using OPPL and directly OWL-API

Prototype implementation

- Three-phase transformation
  - detection of source pattern in ontology
  - generation of transformation instructions
    - ✓ instantiation of the transformation part of the pattern
  - actual transformation
    - ✓ using OPPL and directly OWL-API
- The user can interact in each step

Prototype implementation

- Three-phase transformation
  - detection of source pattern in ontology
  - generation of transformation instructions
    - ✓ instantiation of the transformation part of the pattern
  - actual transformation
    - ✓ using OPPL and directly OWL-API
- The user can interact in each step

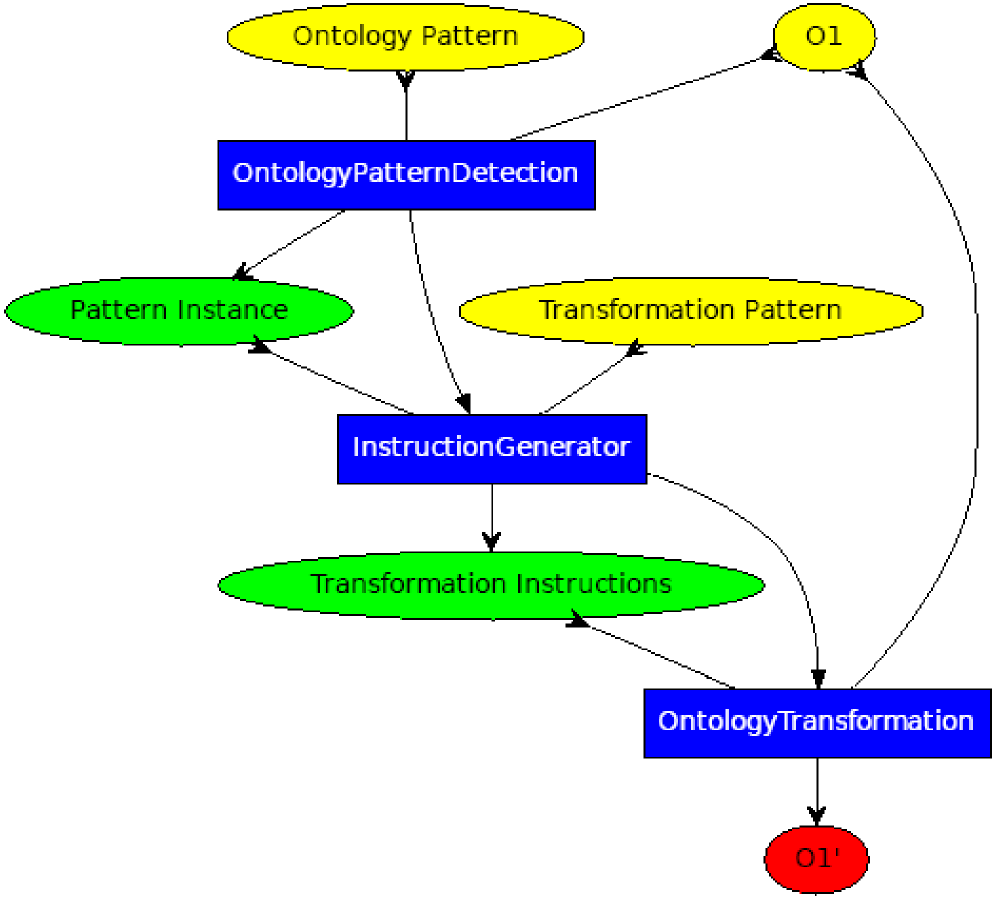
- *Services available via POST method at* http://owl.vse.cz:8080

Prototype implementation

- Three-phase transformation
  - – detection of source pattern in ontology
  - – generation of transformation instructions
    - ✓ instantiation of the transformation part of the pattern
  - – actual transformation
    - ✓ using OPPL and directly OWL-API
- The user can interact in each step

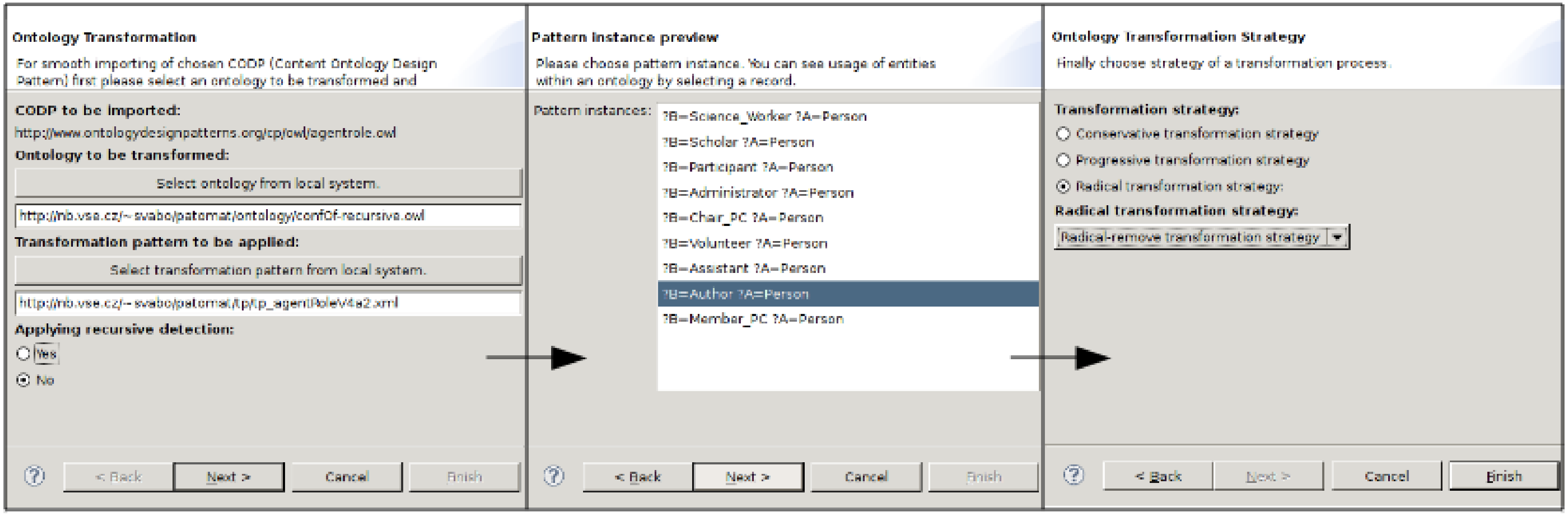
- *Services available via POST method at* http://owl.vse.cz:8080
- *Tutorial, including technical details and sample codes, available* http://owl.vse.cz:8080/tutorial/

Pipeline of RESTful Services



Alternative implementation – Java library

- Used by the XDTools ontology engineering environment (ISTC/CNR, Rome)
- Wizard-based user interface



Ontology Transformation

For smooth importing of chosen CODP (Content Ontology Design Pattern) first please select an ontology to be transformed and

CODP to be imported:

http://www.ontologydesignpatterns.org/cp/owl/agentrole.owl

Ontology to be transformed:

http://nb.vse.cz/~svabo/patomat/ontology/confOf-recursive.owl

Transformation pattern to be applied:

http://nb.vse.cz/~svabo/patomat/tp/tp_agentRoleV4a2.xml

Applying recursive detection:

○ Yes

● No

Pattern instance preview

Please choose pattern instance. You can see usage of entities within an ontology by selecting a record.

Pattern instances:

- ?B=Science_Worker ?A=Person
- ?B=Scholar ?A=Person
- ?B=Participant ?A=Person
- ?B=Administrator ?A=Person
- ?B=Chair_PC ?A=Person
- ?B=Volunteer ?A=Person
- ?B=Assistant ?A=Person
- ?B=Author ?A=Person
- ?B=Member_PC ?A=Person

Ontology Transformation Strategy

Finally choose strategy of a transformation process.

Transformation strategy:

○ Conservative transformation strategy

○ Progressive transformation strategy

● Radical transformation strategy:

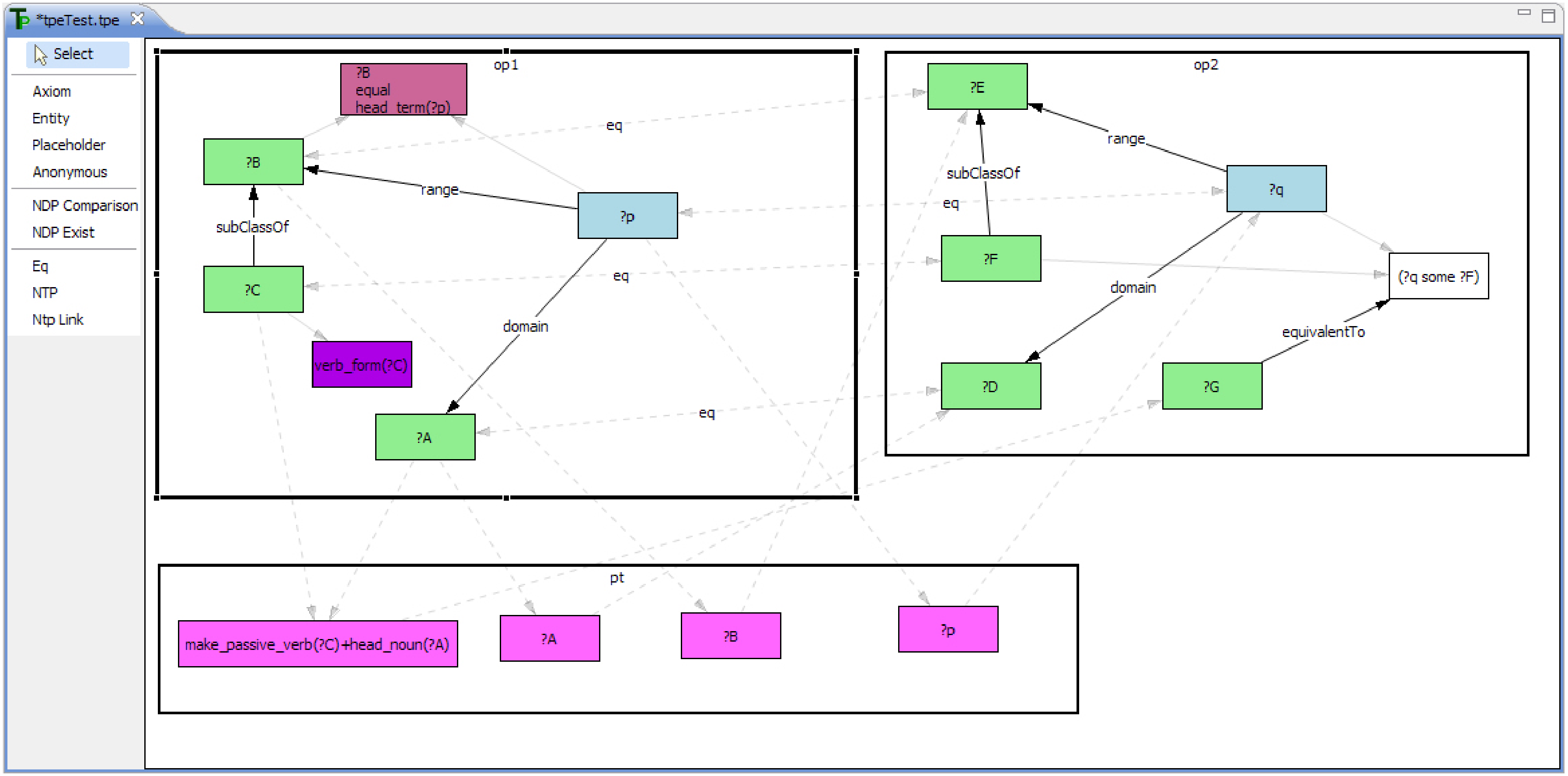
Radical transformation strategy:

Radical-remove transformation strategy

Select ontology from local system.
Select transformation pattern from local system.
< Back
Next >
Cancel
Finish
< Back
Next >
Cancel
Finish
< Back
Next >
Cancel
Finish

Support for transformation pattern authoring

- TPE – Transformation Pattern Editor



Agenda

- Context and motivations
- (Ontology) Transformation patterns
  - – Structure and (abstract) use
- Use cases
  - – Ontology matching
  - – Content pattern import
  - – Special use case: FOAF ‘knows’
- Transformation workflow and implementation
- Ongoing and future work

Ongoing and future work

- Comprehensive library of naming patterns relevant for ontology style transformation
  - Implementation on top of existing lexical sources
- Canonical methods for swapping info between logical and annotation spaces while transforming
- Ontologies of logical/structural patterns
  - Patterns structure; categorisation facets
  - Patterns usage, esp. matching to modelling issues
- Elaborate more use cases
  - other CPs; matching settings; reasoning settings
- More advanced detection techniques

THANKS FOR YOUR ATTENTION

References

- W.N. Borst: Construction of Engineering Ontologies for Knowledge Sharing and Reuse. PhD dissertation, University of Twente, Enschede, 1997.
- P. Clark et al.: Knowledge Patterns. In: Handbook of Ontologies, Springer 2003.
- T. Gruber: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 1993.
- E. Mikroyannidi et al.: Inspecting regularities in ontology design using clustering. In: ISWC'11.
- V. Presutti et al.: Extreme Design (XD): Pattern-based Ontology Design. Tutorial at ESWC'09.
- V. Presutti et al.: eXtreme Design with Content Ontology Design Patterns. In: WOP@ISWC'09
- D. Schober et al.: Survey-based naming conventions for use in OBO Foundry ontology development
- O. Šváb-Zamazal, V. Svátek: Analysing Ontological Structures through Name Pattern Tracking. In: EKAW'08.
- V. Svátek et al., Ontology Naming Pattern Sauce for (Human and Computer) Gourmets . In: WOP@ISWC'09