JÖNKÖPING UNIVERSITY
School of Engineering

LINKÖPING UNIVERSITY

Consiglio Nazionale
delle Ricerche

KARL HAMMAR & VALENTINA PRESUTTI

TEMPLATE-BASED CONTENT ODP INSTANTIATION

OVERVIEW

- Established methods of CODP instantiation.
- Our experiences of using CODPs in projects.
- The alternative: template-based instantiation.
    - Benefits/drawbacks.
    - Instantiation method.
- Evaluation.
- Tool Support.

ESTABLISHED METHODS

- eXtreme Design.
- Falbo et al. / Rui et al.
- OPPL.

CONTENT PATTERN USE WITH EXTREME DESIGN

- XD: *"a family of methods and associated tools, based on the application, exploitation, and definition of Ontology Design Patterns (ODPs) for solving ontology development issues".*
- ODPs are small, autonomous, non-trivial, OWL ontologies.
- Operations: import, specialization, composition, etc.
- XD workflow emphasis: agile, iterative, pairs, testing, ODPs.
- XD workflow core steps: find ODP, instantiate ODP, integrate solution.
- Instantiation typically performed via specialization (though cloning mentioned in passing).

APPLICATION BY EXTENSION OR ANALOGY

- Falbo et al., Ruy et al.: Foundational Ontology Patterns (FOP) vs Domain-related Ontology Patterns (DROP), focusing on conceptual design issue and its solution (i.e., analogous to Fowler’s Analysis Patterns).
- FOPs reused *by analogy* (i.e. ,reproduction of solution), DROPs reused *by extension* (i.e., specialization).
- Our view: FOP-analogy / DROP-extension pairing may be to restrictive.

ENCODING CODPS WITH OPPL

- Ontology Pre-Processing Language – macro language enabling rapid transformation of large ontologies.
- Macro engine adds/removes entities/axioms based on variables set by user and conditions evaluated against ontology.
- CODPs can be written as OPPL macros - unbound variables filled by user indicate new entities to create or existing entities to specialize.
- Tooling also includes annotation properties to track CODP macro usage in target ontologies.
- Promising technique that has seen limited uptake.

OUR EXPERIENCES OF CODP USE – VALCRI

- Project focus: Visual Analytics capability for law enforcement analysts, operating over integrated heterogenous data sources. Triple store backend.
- Goal 1: Easily understandable ontologies, to be used and co-developed by software developers.
- Goal 2: Ontologies easy to modify for deployment in different contexts.

OUR EXPERIENCES OF CODP USE – VALCRI

- Foundational entities from transitive import closure make no sense in target domain. *"What is this Situation class? I don't want it!"*
- CODP labelling to generic for target domain: *"Why is this thing called Agent? We always call it Nominal in policing!"*
- Devs uncomfortable modifying ontologies due to lacking confidence that they understood initial design (largely due to the above mentioned challenges)

OUR EXPERIENCES OF CODP USE – IMSK

- Goals: Reconfigurable area security system, ontologies as pluggable configuration modules.
- Experiences:
    - Some users (quite intensely) disliked transitive import closure as it added concepts they did not ask for nor understand value of.
    - Other users liked transitive import, as it validated the soundness of their design against existing known good practice.
    - When set loose to implement w/o method guidance, users consistently used whiteboard prototyping and recreated CODP structure in tooling from scratch. *owl:imports* was NEVER used.

OUR EXPERIENCES OF CODP USE – E-CARE@HOME

- Goal: improve home healthcare for elderly via IoT / Smart home and data integration for reminders, recommendations, alerts, etc. Ontologies for device configuration and data integration.
- Resulting ontologies contain high-level entities that are unused in target domain.
- Lead dev, to the question of whether import-less CODP instantiation or partial CODP instantiation would be useful:
    - *"Definitely useful. I spent a considerable amount of time to find top- level classes that provide the required links to already designed ones. The lack of such tools is sensed. It can also decrease the rate of errors or inconsistencies in our design."*

EXPERIENCES SUMMARIZED

- Some (not all!) users dislike import of high-level concepts, several steps removed from the domain specifics, into their target model.
  - Such users tend to be practitioners and software developers, rather than researchers or knowledge engineers.
- Some users dislike the generic naming/labeling provided by reused CODPs.
- Some users would prefer the ability to instantiate CODPs *partially* rather than *in whole* (which is not possible using *owl:imports*).

TEMPLATE-BASED INSTANTIATION

- Idea not new: see previous slides on earlier work.
- Our contribution:
  - Discussion on benefits/drawbacks
  - Suggested practical method
  - Initial evaluation of feasibility and utility

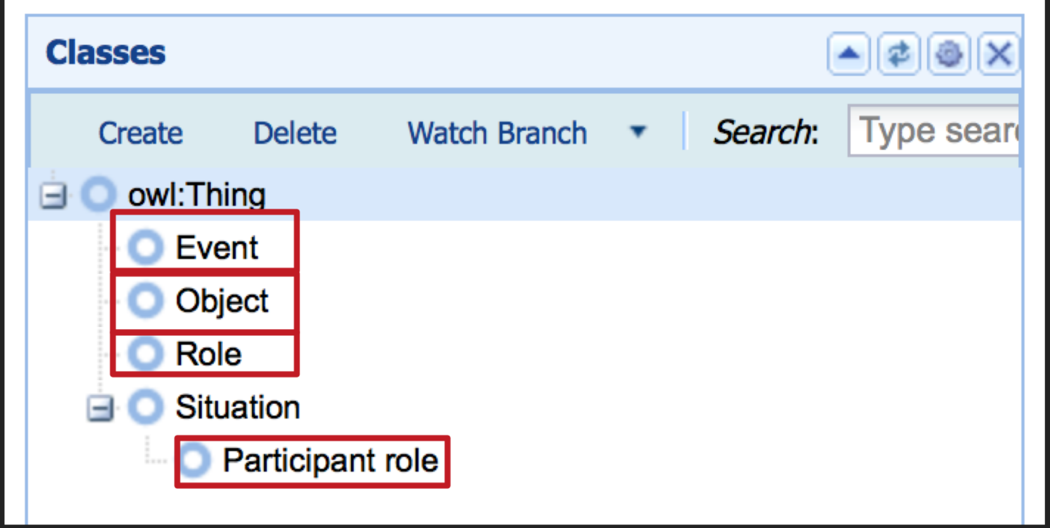
TEMPLATE-BASED CONTENT ODP INSTANTIATION

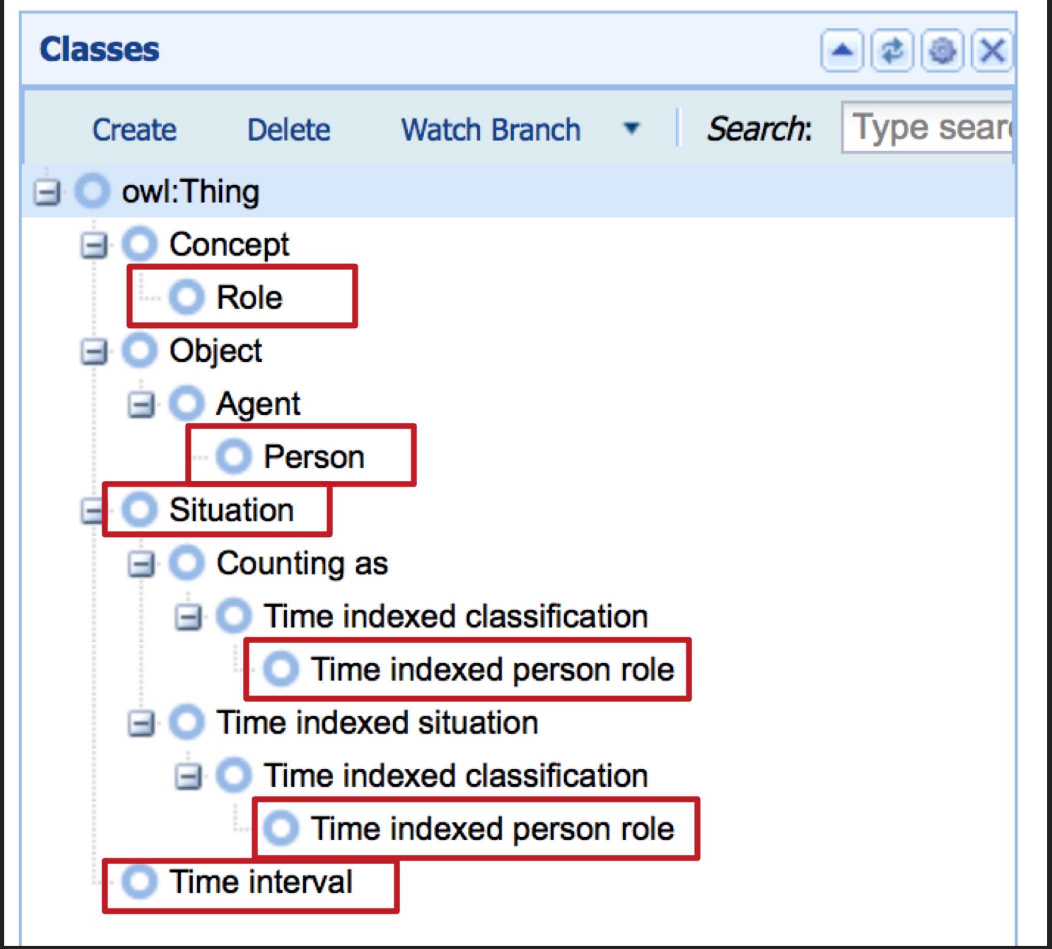
BENEFITS & DRAWBACKS

- Benefit: Alleviates issues previously discussed (no un-needed domain-level concepts, no large import closure including foundational concepts).
- Benefit: Reduces risk of breakage, as CODP instantiations are wholly contained within target ontology namespace (also simplifies tooling implementation).
- Benefit: Reduces barrier-to-entry of future refactoring and debugging, ontology engineer ”owns” their whole implementation module.
- Benefit: Validation with domain experts simplified - no foreign terms that cause confusion.
- Drawback: No instant interoperability between multiple instantiations of same CODP – alignment and OWL reasoning needed.
- Drawback: Higher-level classes in CODP may need to be instantiated multiple times in target ontology, increasing risk of modeling mistakes and inconsistency.

METHOD (STEP 1)

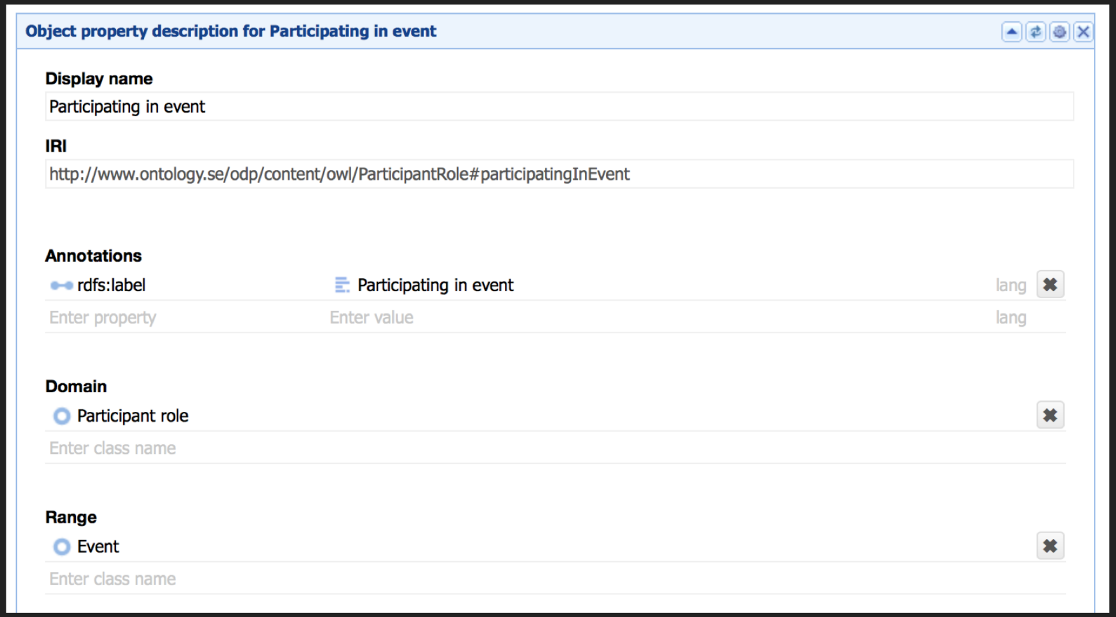
- Copy CODP leaf classes into subclasses of *owl:Thing* in target module. If two leaf classes in source CODP have some shared parent beneath *owl:Thing* level, copy least common consumer also as shared parent to the copied leaves.





METHOD (STEP 2)

- Copy object or datatype properties that have as domain or range the classes copied above. For object properties: try to narrow any unmatched half of the domain/range pair to the least common subsumer or if non-existent, leaf level.



Object property description for Participating in event

Display name

Participating in event

IRI

http://www.ontology.se/odp/content/owl/ParticipantRole#participatingInEvent

Annotations

rdfs:label	Participating in event	lang
Enter property	Enter value	lang

Domain

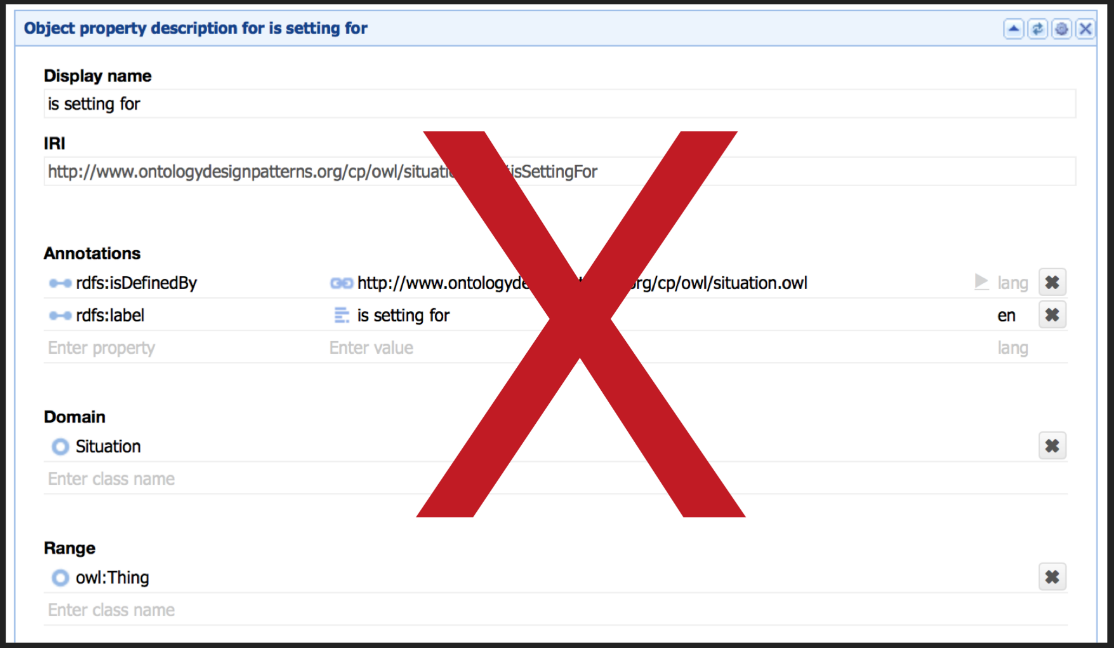
Participant role

Enter class name

Range

Event

Enter class name



Object property description for is setting for

Display name

is setting for

IRI

http://www.ontologydesignpatterns.org/cp/owl/situati [illegible] isSettingFor

Annotations

rdfs:isDefinedBy	http://www.ontologyde [illegible] rg/cp/owl/situation.owl	lang
rdfs:label	is setting for	en
Enter property	Enter value	lang

Domain

Situation

Enter class name

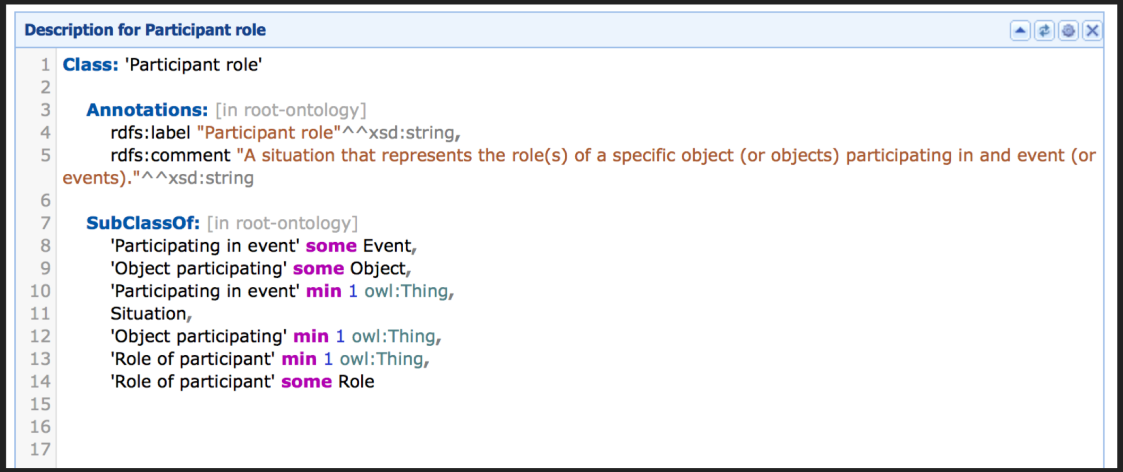
Range

owl:Thing

Enter class name

METHOD (STEP 3)

- Copy (and similarly to step 2, narrow if necessary) any properties involved in class restrictions on classes copied in step 1 – use the copied properties to create equivalent restrictions in the target module.

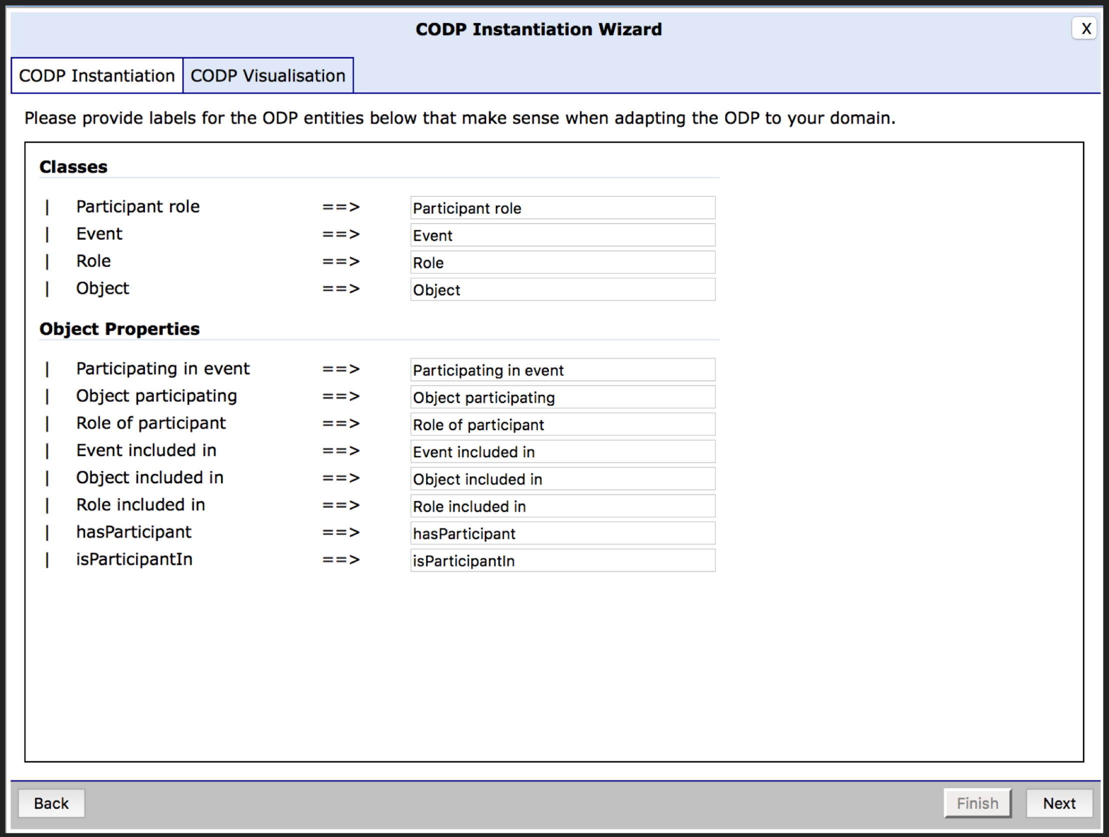


Description for Participant role

```
Class: 'Participant role'

    Annotations: [in root-ontology]
        rdfs:label "Participant role"^^xsd:string,
        rdfs:comment "A situation that represents the role(s) of a specific object (or objects) participating in and event (or events)."^^xsd:string

    SubClassOf: [in root-ontology]
        'Participating in event' some Event,
        'Object participating' some Object,
        'Participating in event' min 1 owl:Thing,
        Situation,
        'Object participating' min 1 owl:Thing,
        'Role of participant' min 1 owl:Thing,
        'Role of participant' some Role
```



CODP Instantiation Wizard

Please provide labels for the ODP entities below that make sense when adapting the ODP to your domain.

Classes

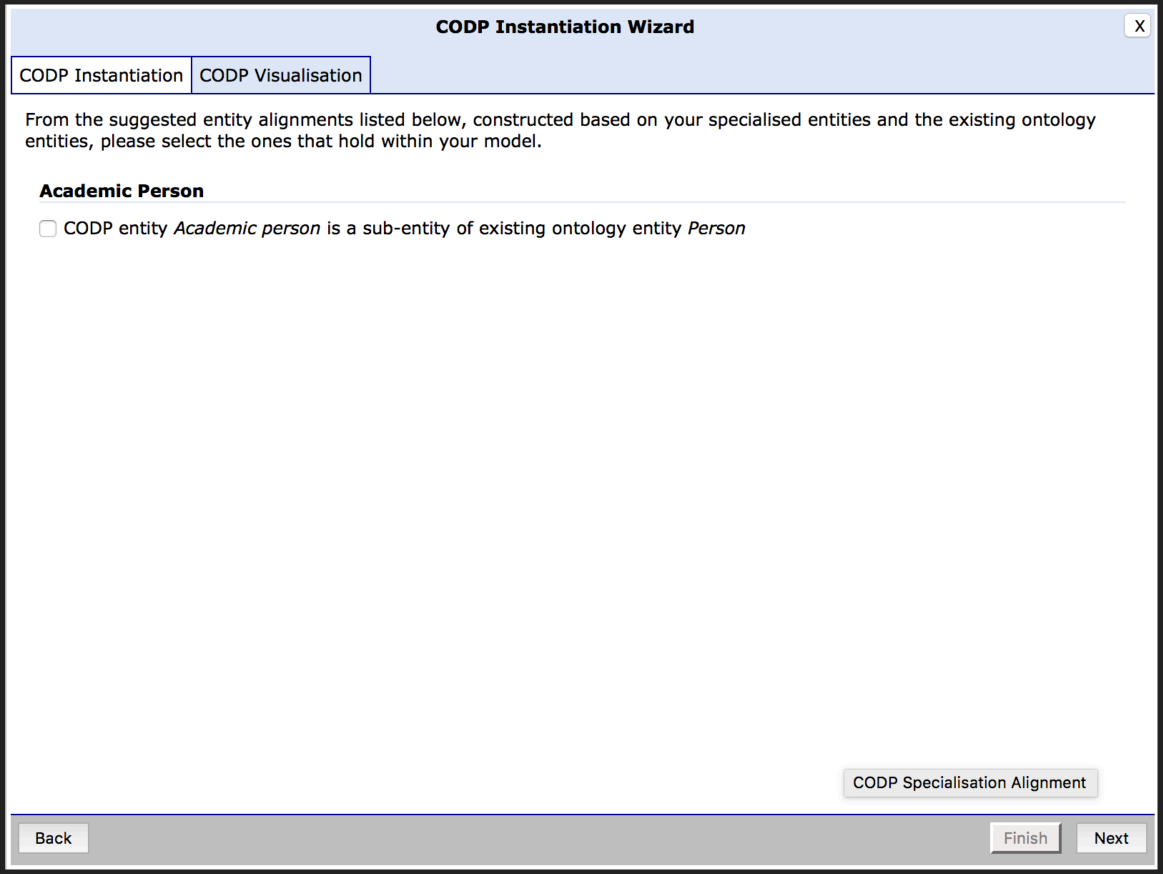
Participant role	==>	Participant role
Event	==>	Event
Role	==>	Role
Object	==>	Object

Object Properties

Participating in event	==>	Participating in event
Object participating	==>	Object participating
Role of participant	==>	Role of participant
Event included in	==>	Event included in
Object included in	==>	Object included in
Role included in	==>	Role included in
hasParticipant	==>	hasParticipant
isParticipantIn	==>	isParticipantIn

METHOD (STEP 4)

- Merge the resulting structure with existing entities in the target module using suitable ontology matching techniques to find candidate matches.



CODP Instantiation Wizard

From the suggested entity alignments listed below, constructed based on your specialised entities and the existing ontology entities, please select the ones that hold within your model.

Academic Person

□ CODP entity *Academic person* is a sub-entity of existing ontology entity *Person*

EDGE CASES

- The proposed method has worked well in initial testing with CODPs from the portal. However, there are many cases where it would not work without further refinement:
  - CODPs where individual leaf classes need to be instantiated twice or more (e.g., the *Place* class in the *Place* CODP, which could be instantiated both as narrower and broader Place in target module)
  - When a CODP reuses and specializes higher-level concepts from another CODP, it might be the case that child CODP classes are leaves on the same level as classes from the parent CODP (which are not intended to be instantiated in the child CODP).

EVALUATION

- Constructed two sets of ontology requirements, A and B, in the form of Competency Questions, Contextual Statements, and Reasoning Requirements.
- Based on each requirement set, generated two sibling ontologies using template-based instantiation and traditional specialization-based instantiation, for a total of four ontologies.
- Gave participants three tasks:
    1. For requirements set A, answer which out of seven provided CQs that the developed ontologies fulfill.
    2. For requirements set B, answer which out of nine provided CQs that the developed ontologies fulfill.
    3. For requirement set A, modify the two sibling ontologies by adding four object properties, specializing some of the more generic properties already in place.
- Surveyed users on which of the two ontology variants they found easiest to understand (for tasks 1 and 2) and easiest to modify (task 3).

EVALUATION RESULTS

	Task 1	Task 2	Task 3
Template-based easiest	4	2	3
Equally easy/difficult	1	2	0
Specialisation-based easiest	0	0	0
Correct answer rate	83 %	81 %	

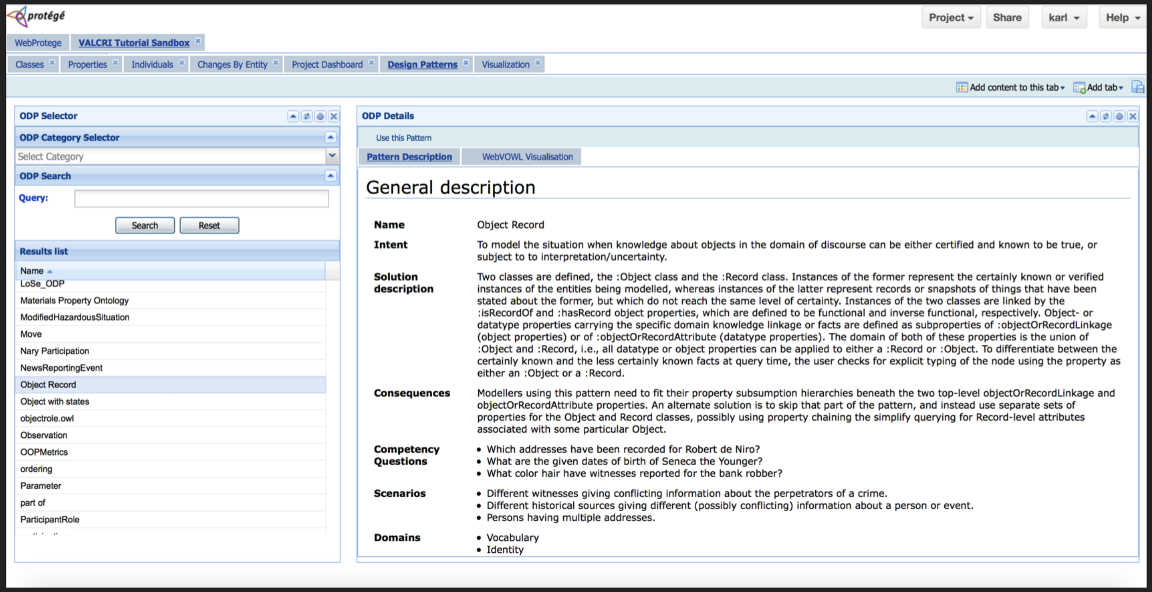
Responses to tasks 1-2 indicate ease of understanding, task 3 indicates ease of modifying.
Response rate decreases as not all participants completed all tasks within the workshop time-frame.

EVALUATION FINDINGS

- Among our (admittedly very small) set of respondents, *no one* preferred working with results of specialization-based CODP instantiation.
- The previously discussed method for CODP-based instantiation actually works in practice!

TOOL SUPPORT

- XD for WebProtégé:
  - http://wp.xd-protege.com
  - https://github.com/hammar
- Features:
  - ODP Browser & Search
  - Instantiation Wizard
  - Visualization with WebVOWL (many thanks to the VisualDataWeb project, including particularly Steffen Lohmann!)



LoSe_ODP
Materials Property Ontology
ModifiedHazardousSituation
Move
Nary Participation
NewsReportingEvent
Object Record
Object with states
objectrole.owl
Observation
OOPMetrics
ordering
Parameter
part of
ParticipantRole

General description

Name Object Record

Intent To model the situation when knowledge about objects in the domain of discourse can be either certified and known to be true, or subject to to interpretation/uncertainty.

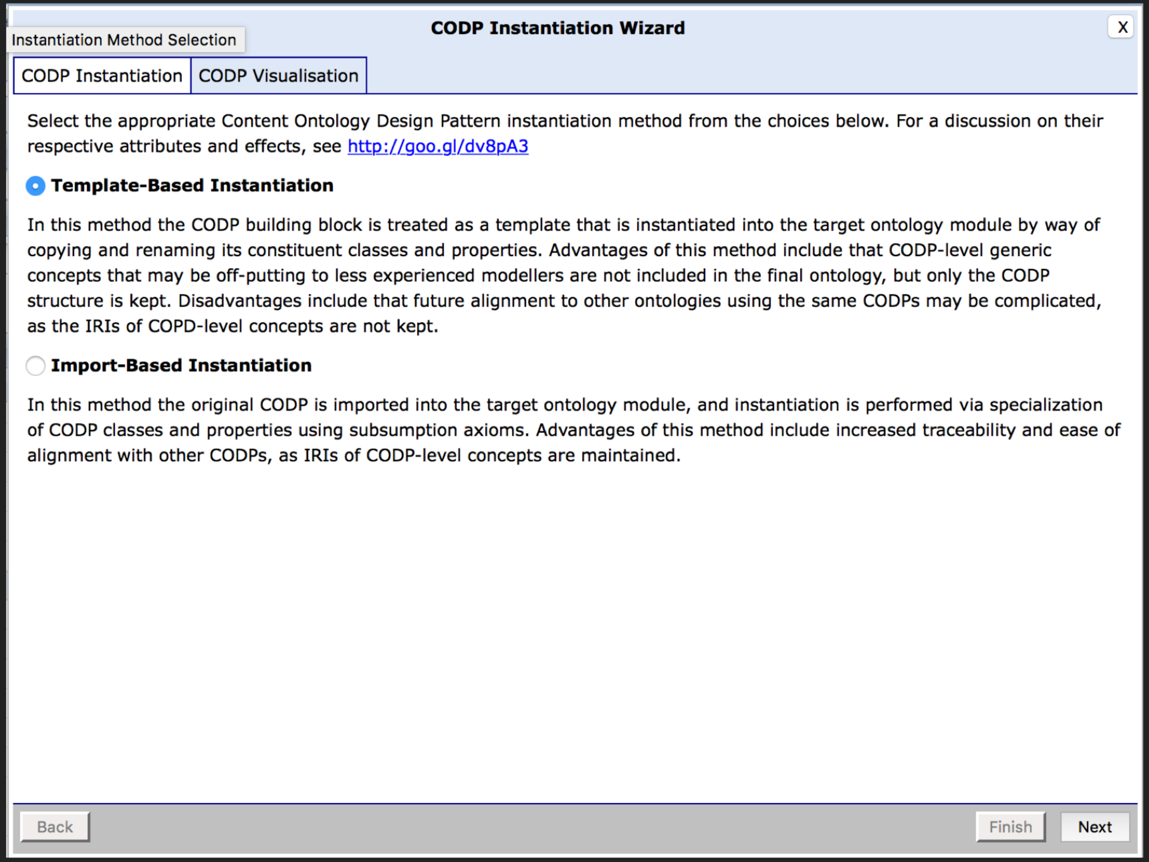
Solution description Two classes are defined, the :Object class and the :Record class. Instances of the former represent the certainly known or verified instances of the entities being modelled, whereas instances of the latter represent records or snapshots of things that have been stated about the former, but which do not reach the same level of certainty. Instances of the two classes are linked by the :isRecordOf and :hasRecord object properties, which are defined to be functional and inverse functional, respectively. Object- or datatype properties carrying the specific domain knowledge linkage or facts are defined as subproperties of :objectOrRecordLinkage (object properties) or of :objectOrRecordAttribute (datatype properties). The domain of both of these properties is the union of :Object and :Record, i.e., all datatype or object properties can be applied to either a :Record or :Object. To differentiate between the certainly known and the less certainly known facts at query time, the user checks for explicit typing of the node using the property as either an :Object or a :Record.

Consequences Modellers using this pattern need to fit their property subsumption hierarchies beneath the two top-level objectOrRecordLinkage and objectOrRecordAttribute properties. An alternate solution is to skip that part of the pattern, and instead use separate sets of properties for the Object and Record classes, possibly using property chaining the simplify querying for Record-level attributes associated with some particular Object.

Competency Questions
- Which addresses have been recorded for Robert de Niro?
- What are the given dates of birth of Seneca the Younger?
- What color hair have witnesses reported for the bank robber?

Scenarios
- Different witnesses giving conflicting information about the perpetrators of a crime.
- Different historical sources giving different (possibly conflicting) information about a person or event.
- Persons having multiple addresses.

Domains
- Vocabulary
- Identity



Select the appropriate Content Ontology Design Pattern instantiation method from the choices below. For a discussion on their respective attributes and effects, see http://goo.gl/dv8pA3

● **Template-Based Instantiation**

In this method the CODP building block is treated as a template that is instantiated into the target ontology module by way of copying and renaming its constituent classes and properties. Advantages of this method include that CODP-level generic concepts that may be off-putting to less experienced modellers are not included in the final ontology, but only the CODP structure is kept. Disadvantages include that future alignment to other ontologies using the same CODPs may be complicated, as the IRIs of COPD-level concepts are not kept.

○ **Import-Based Instantiation**

In this method the original CODP is imported into the target ontology module, and instantiation is performed via specialization of CODP classes and properties using subsumption axioms. Advantages of this method include increased traceability and ease of alignment with other CODPs, as IRIs of CODP-level concepts are maintained.

CONCLUSIONS

- Existing approaches to CODP instantiation are not palatable to all classes of users.
- Template-based instantiation is a promising approach to satisfying these users' preferences.
- Template-based instantiation also has other benefits, (e.g., self-containedness providing stability and simplifying tooling development), as well as disadvantages (e.g., interoperability with other CODP instantiations)
- Steps for implementing template-based instantiation in practice have been developed and shown to work.