Computational Ontologies
Content Ontology Design Patterns

Aldo Gangemi
Valentina Presutti
Laboratory for Applied Ontology (ISTC-CNR), Roma
{aldo.gangemi, valentina.presutti}@istc.cnr.it

Course Outline

- Ontologies and the Semantic Web
- Ontology Design and Ontology Design Patterns
- Content Ontology Design Patterns
- Design by Re-Engineering

CPs: some theory

Content OPs (CPs)

- CPs encode conceptual, rather than logical design patterns.
  - Logical OPs solve design problems independently of a particular conceptualization
  - CPs are patterns for solving design problems for the domain classes and properties that populate an ontology, therefore they address content problems

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

- Modeling problems solved by CPs have two components: domain and requirements.
  - A same domain can have many requirements (e.g. different scenarios in a clinical information context)
  - A same requirement can be found in different domains (e.g. different domains with a same “expert finding” scenario)
  - A typical way of capturing requirements is by means of *competency questions* [11]

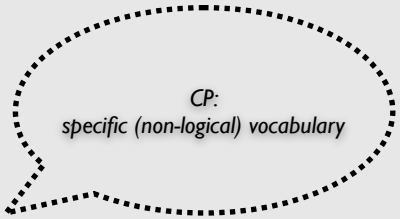
Peter Clark's idea

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

CPs vs. Logical OPs

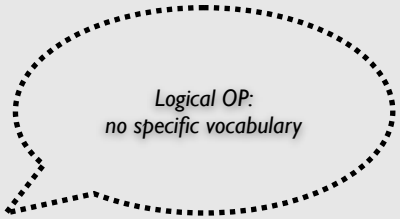
$\forall c((consumer(c) \wedge \exists p(producer(p) \wedge connects(c,p))) \rightarrow supplied(c))$

SubClassOf
((intersectionOf
Consumer
(restriction(connects someValuesFrom(Producer))))
Supplied)



$\forall c((\phi(c) \wedge \exists p(\psi(p) \wedge \rho(c,p))) \rightarrow \chi(c))$

SubClassOf
((intersectionOf
owl:Class:ϕ
(restriction(owl:ObjectProperty:ρ someValuesFrom(owl:Class:ψ))))
owl:Class:χ)



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

Formal characteristics of OWL CPs

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

Characteristics of CPs

- Requirement-covering components
  - They are defined in terms of the requirements (or cqs) they satisfy
- Computational components.
  - CPs are language-independent, and should be encoded in a high-order representation language.
    - Nevertheless, their (sample) representation in a computational logic that can be processed by parsers and automatic reasoners is needed in order to (re)use them as building blocks in ontology design.
- Small, autonomous components.
  - A CP is a small, autonomous ontology and ensures a certain set of inferences to be enabled on its corresponding knowledge base.
  - Smallness and autonomy of CPs facilitate ontology designers: composing CPs enables them to govern the complexity of the whole ontology.
  - CPs require a critical size, so that their diagrammatical visualizations are aesthetically acceptable and easily memorizable.

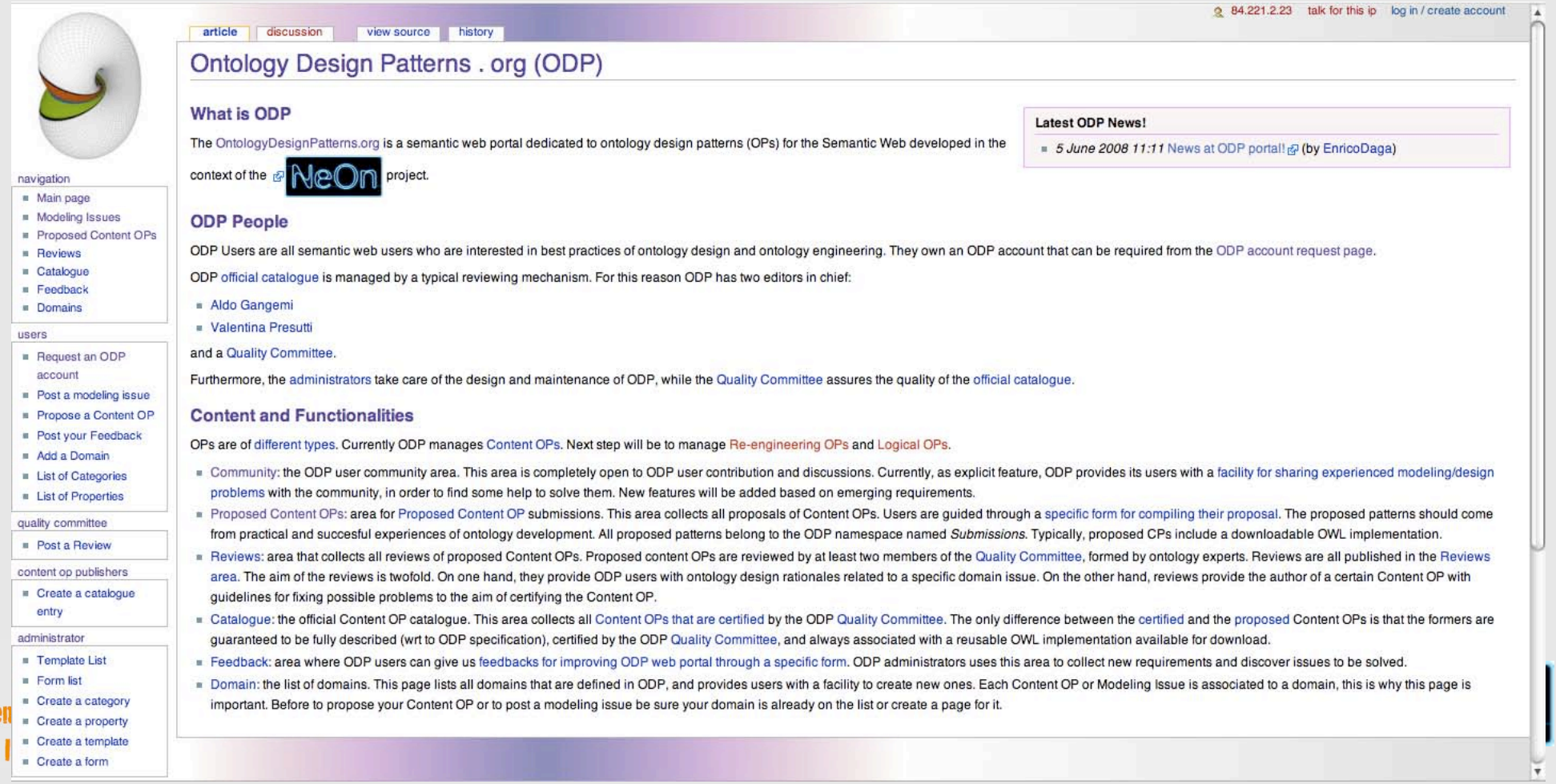
- Hierarchical components.
  - A CP can be an element in a partial order, where the ordering relation requires that at least one of the classes or properties in the pattern is specialized.
  - A hierarchy of CPs can be built by specializing or generalizing some of the elements (either classes or relations).
  - For example, the agent-role pattern can be specialized to the person-taking-a-musician-role pattern.
- Cognitively relevant components.
  - CP visualization must be intuitive and compact, and should catch relevant, “core” notions of a domain.
  - An interesting result from cognitive learning is that the development of expert skills typically “selects” patterns of concepts that are richly interconnected, and in normal cases, these patterns are applied without an explicit reference to the underlying detailed knowledge acquired during the training period.
  - A CP must contain the central notions that “make rational thinking move” for an expert in a given domain for a given task.

Characteristics of CPs

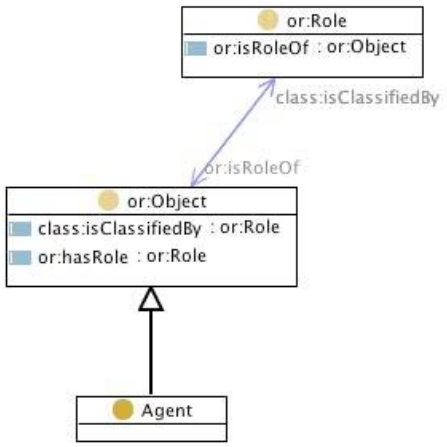
- Reasoning-relevant components
  - They allow some form of inference (*minimal axiomatization*, e.g. not an isolated class)
- Linguistically relevant components.
  - Many CPs nicely match linguistic patterns called frames.
  - A frame can be described as a lexically founded ontology design pattern.
    - Frames typically encode argument structures for verbs, e.g. the frame Desiring associates elements (or “semantic roles”) such as Experiencer, Event, FocalParticipant, LocationOfEvent, etc. The richest repository of frames is FrameNet.
  - Frames can be used for validating CPs with respect to lexical coverage, for lexicalizing them,and can be reengineered in order to populate the CP catalogue
- Best practice components.
  - A CP should be used to describe a “best practice” of modelling.
  - Best practices are intended as local, thus derived from experts.
  - The quality of CPs is currently based on the personal experience and taste of the proposers, or on the provenance of the knowledge resource where the pattern comes from.

Presentation

- A catalogue of CPs
  - http://www.ontologydesignpatterns.org (odp-web)
  - catalogue entry
- Annotation properties:
  - http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl
  - annotation of OWL implementation of CPs



An example of CP: Agent Role



Elements

*The **AgentRole** Content OP locally defines the following ontology elements:*

Agent (owl:Class)

Any agentive Object, either physical, or social.

Agent page

Reviews about AgentRole

The are no reviews.

Go back to the List of Content OP proposals

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

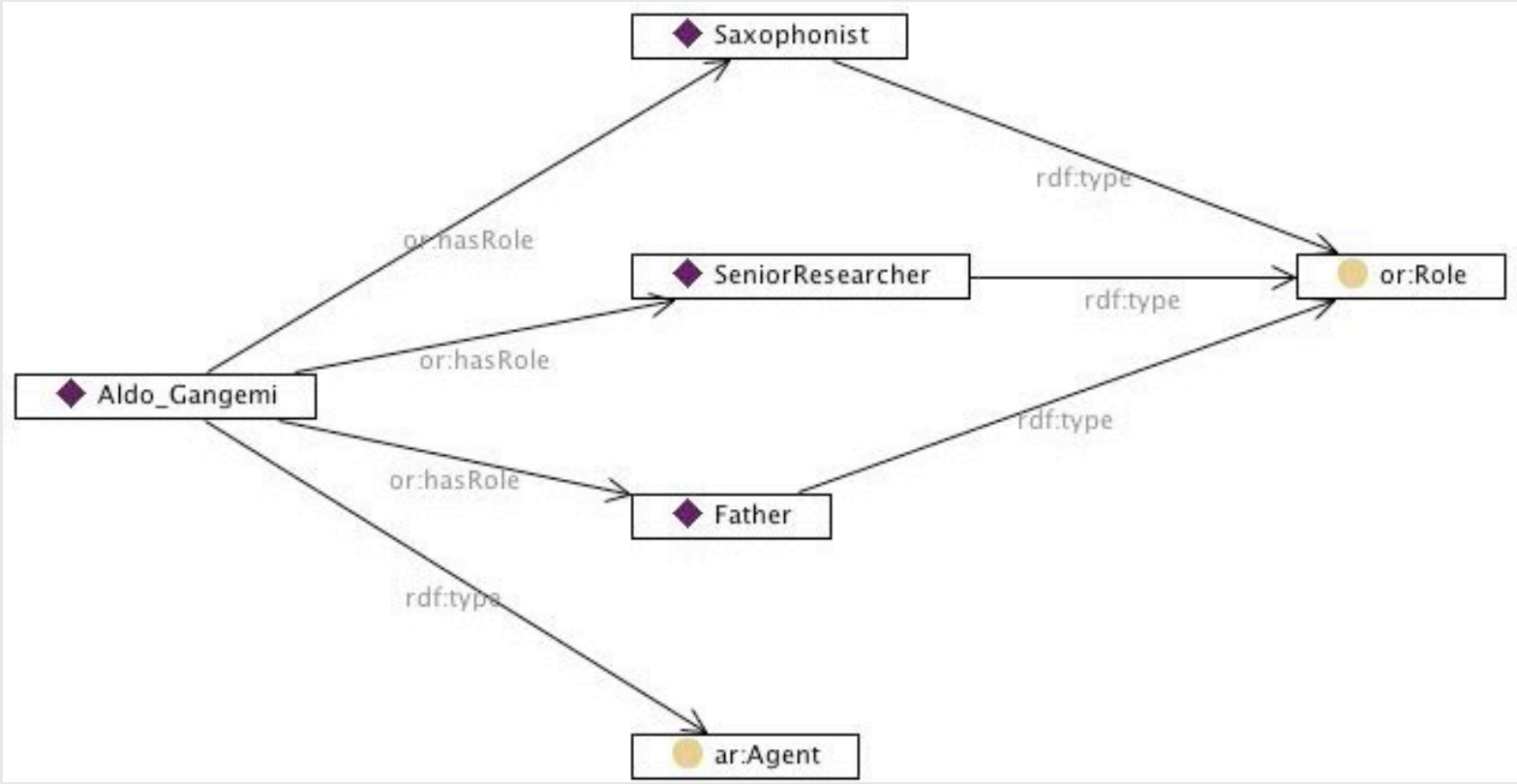
The elements of this Content OP are added with the elements of its components and/or the elements of the Content OPs it is a specialization of.

AgentRole

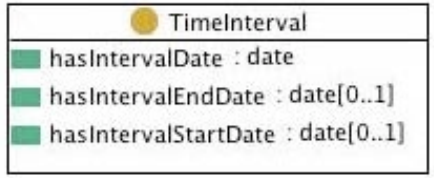
Submitted by	ValentinaPresutti
Name	agent role
Also Known As	
Intent	To represent agents and the roles they play.
Domains	Management, Organization, Scheduling
Competency Questions	which agent does play this role?, what is the role that played by that agent?
Reusable OWL Building Block	http://www.ontologydesignpatterns.org/cp/owl/agentrole.owl
Consequences	This CP allows designers to make assertions on roles played by agents without involving the agents that play that roles, and vice versa. It does not allow to express temporariness of roles.
Scenarios	She greeted us all in her various roles of mother, friend, and daughter.
Known Uses	
Web References	
Other References	
Examples (OWL files)	http://www.ontologydesignpatterns.org/cp/examples/agentrole/ex1.owl
Extracted From	http://www.loa-cnr.it/ontologies/DUL.owl
Reengineered From	
Has Components	
Specialization Of	Submissions:Objectrole
Related CPs	

An example of CP: Agent Role Instantiation

- Scenario: Aldo Gangemi is a senior researcher. He is also father and saxophonist.



An example of CP: Time Interval



Elements

*The **TimeInterval** Content OP locally defines the following ontology elements:*

Time Interval (owl:Class)

Any region in a dimensional space that represents time.

TimeInterval page

has interval date (owl:DatatypeProperty)

A datatype property that encodes values from xsd:date for a time interval; a same time interval can have more than one xsd:date value: begin date, end date, date at which the interval holds, as well as dates expressed in different formats: xsd:gYear, xsd:dateTime, etc.

hasIntervalDate page

has interval start date (owl:DatatypeProperty)

The start date of a time interval.

hasIntervalStartDate page

has interval end date (owl:DatatypeProperty)

The end date of a time interval.

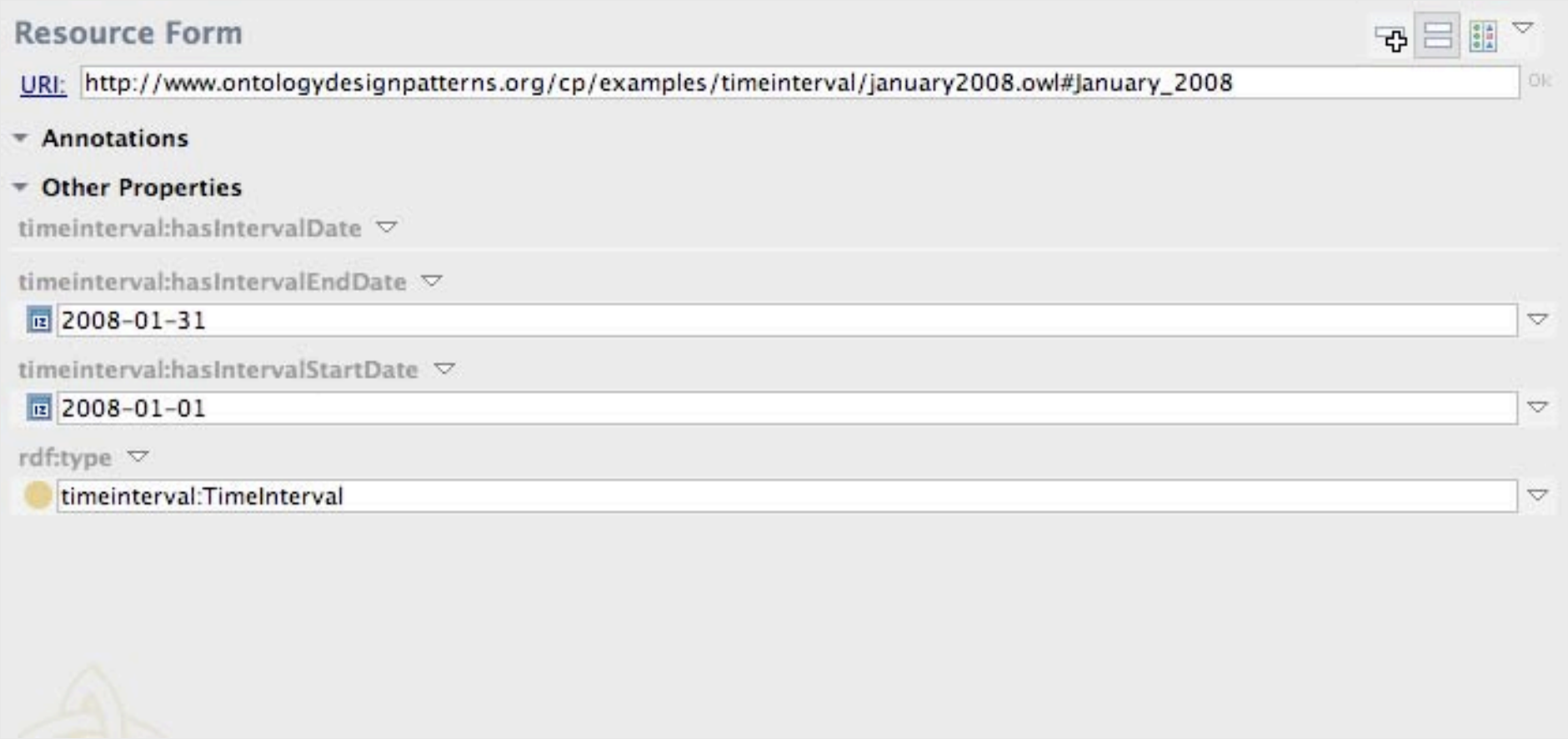
hasIntervalEndDate page

TimeInterval

Submitted by	ValentinaPresutti
Name	time interval
Also Known As	
Intent	To represent time intervals.
Domains	Time
Competency Questions	What is the end time of this interval?, What is the starting time of this interval?, What is the date of this time interval?
Reusable OWL Building Block	http://www.ontologydesignpatterns.org/cp/owl/timeinterval.owl
Consequences	The dates of the time interval are not part of the domain of discourse, they are datatype values. If there is the need of reasoning about dates this Content OP should be used in composition with the region Content OP.
Scenarios	The time interval "January 2008" starts at 2008-01-01 and ends at and ends at 2008-01-31.
Known Uses	
Web References	
Other References	
Examples (OWL files)	http://www.ontologydesignpatterns.org/cp/examples/timeinterval/january2008.owl
Extracted From	
Reengineered From	
Has Components	
Specialization Of	
Related CPs	

An example of CP: Time Interval Instantiation

- Scenario: January 2008 starts at 2008-01-01 and ends at 2008-01-31



Covering

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

  cov(CP, CQ)

- A cq_i can be transformed to a query q_i to be submitted to a knowledge base.

- A CP covers *CQ* if it is as expressive as it is needed to store the necessary knowledge for answering q_1, . . . ,q_n .

(Re)use situations:
matching CPs covering against local problems

Representing local problems

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

What we mean by *matching cqs to CPs*

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

Summary of reuse situations and examples

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

Precise or redundant matching

- Precise matching
  - the set of relevant elements of the cqs completely matches the set of CP ontology elements
- Redundant matching
  - the set of relevant elements of the cqs completely matches a subset of CP ontology elements
- For example consider the following local scenario:
  - an agent plays a certain role.
- It can be expressed by the cqs included in the following set:
  - CQ={which agent did play a specific role?}

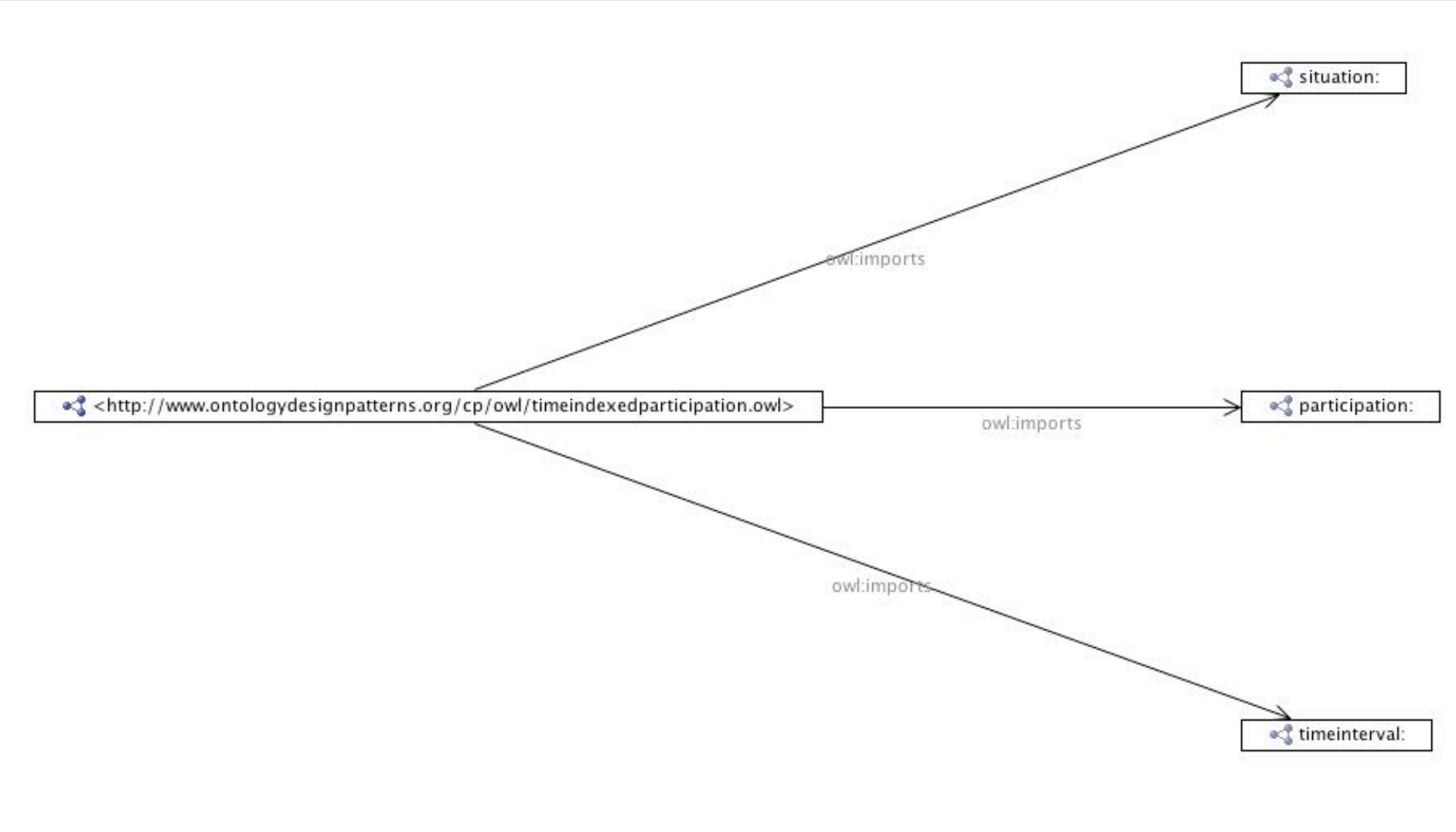
- From the previous CP examples we know that

  cov(agent role, CQ)
- The CP can be reused as it is by **importing** it in the ontology
- A usage operation is identified
  - import

Import

- Import is the basic mechanism for ontology reuse.
- It is also the only one directly supported in the OWL vocabulary
  - i.e., *owl:import*.
- Import is applicable to ontologies, hence also to CPs.
- If an ontology O_2 imports an ontology O_1, all the ontology elements and OWL axioms from O_1 are included in O_2 .
- The imported ontology elements and axioms cannot be modified
  - i.e., the ontology elements and axioms are read-only entities for O_2.
- By importing a CP, an ontology ensures the set of inferences allowed by the CP in its corresponding knowledge base.

Sample import

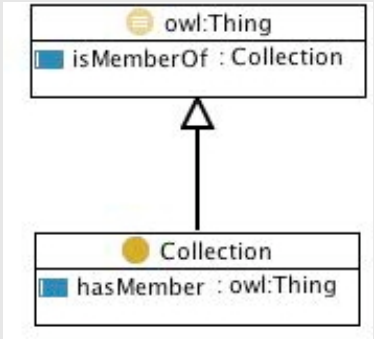


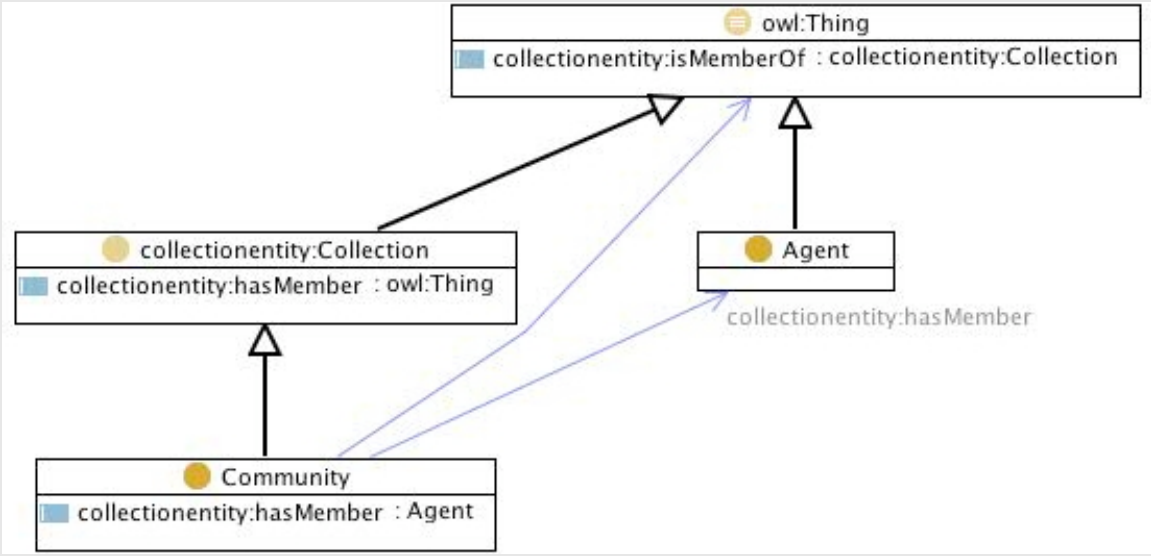
Broader/narrower matching

- Broader matching:
  - The cqs covered by a CP are more general than the local ones.
  - The CP has firstly to be **imported**, then it has to be **specialized** in order to cover the local scenarios.
- Narrower matching:
  - The cqs covered by a CP are more specific than the local ones.
  - The CP has firstly to be **imported**, then it has to be **generalized** in order to cover the local scenarios.
- Two usage operations are identified:
  - specialization
  - generalization

Specialization

- A content pattern CP_2 specializes CP_1 if at least one ontology element of CP_2 is subsumed by an ontology element of CP_1
  - i.e., either by *rdfs:subClassOf* or *rdfs:subPropertyOf*





Broader matching example

- Consider the following scenario:
  - a person plays a certain role.
- it can be expressed by the competency question included in the following set:
  - CQ_1={who did play a certain role?}
- From the previous example we know that

  cov(agent role, Req)
- where *CQ* is more general than CQ_1
- We can import agent role (prefix *ar:*) and define the class *Person* in the following way:

  Person rdfs:subClassOf ar:Agent

Specialization and Generalization of CPs

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

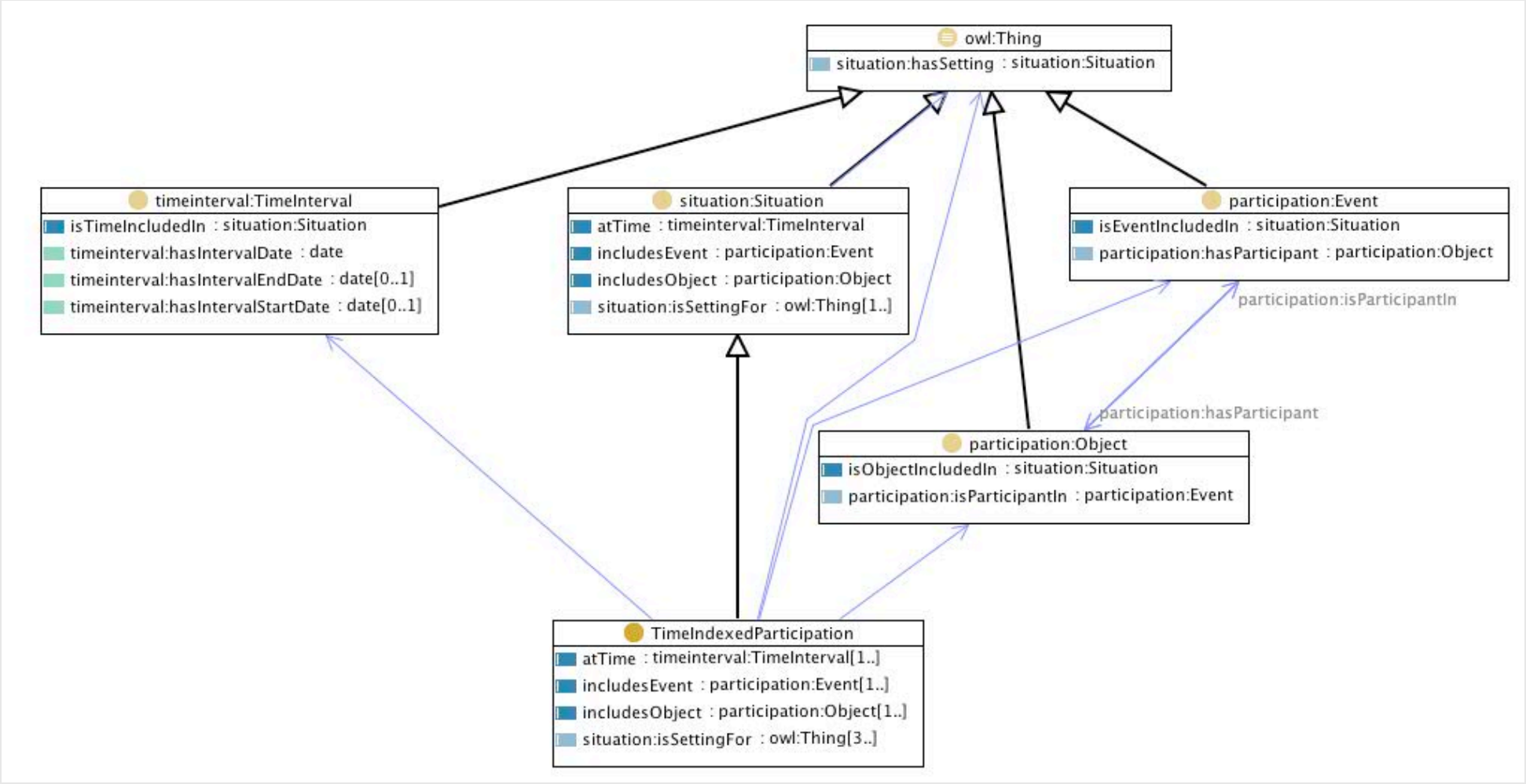
Partial matching

- The CP does not cover all aspects of the local cqs
- The local use case has to be partitioned into smaller pieces.
- One of these pieces will be covered by the selected CP.
- For the other pieces, other CPs have to be selected.
- All selected CPs have to be imported and **composed**.
- One additional usage operation is identified:
  - composition

Composition

- The composition operation relates two CPs and results into a new ontology
- The resulting ontology is composed of the union of the ontology elements and axioms from the two CPs, plus the axioms (e.g. disjointness, equivalence, etc.) that are added in order to link the CPs
- The composition of CP_1 and CP_2 consists of creating a semantic association between CP_1 and CP_2 by adding at least one new axiom, which involves ontology elements from both CP_1 and CP_2
- Typically, also new elements ("expansion") are added when composing

Sample composition



Partial matching example

- For example, consider the following competency questions:
  - cq_1 : who did play a specific role in a certain period?
  - cq_2 : which role does a certain person have at a certain time?

- From previous examples we know that
  - **agent role** covers partially cq_1 and cq_2, as it allows to represent agents and the role they play
  - **time interval** covers partially cq_1 and cq_2, as it allows to represent time intervals

- The ontology resulting from the composition of these two CPs covers both cq_1 and cq_2

- Is that true?

Expansion

- Given that a CP is associated with two unique sets:
  - $OE_{cp1}=\{oe_1,...,oe_n\}$ of all ontology elements from CP_1
  - $AX_{cp1}=\{ax_1,...,ax_n\}$ of all axioms from CP_1. The expansion operation relates a CP to a set of ontology elements, and a set of axioms.
- *Expansion* consists of adding new ontology elements and axioms to a CP.
- The resulting ontology is composed of the ontology elements and axioms of the CP, plus the added ontology elements and axioms.
- The added ontology elements and axioms do not match any CP, otherwise we would have a composition of CPs.
- Given a CP_1 such that *cov(CP_1, CQ_1)*, a set of ontology elements OE, and a set of OWL axioms AX, the expansion of CP_1 by means of OE and AX consists of creating a new ontology O which contains all the ontology elements and axioms of CP_1, OE, and AX, and such that *cov(O, CQ_1)*.

Where do CPs come from?

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

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

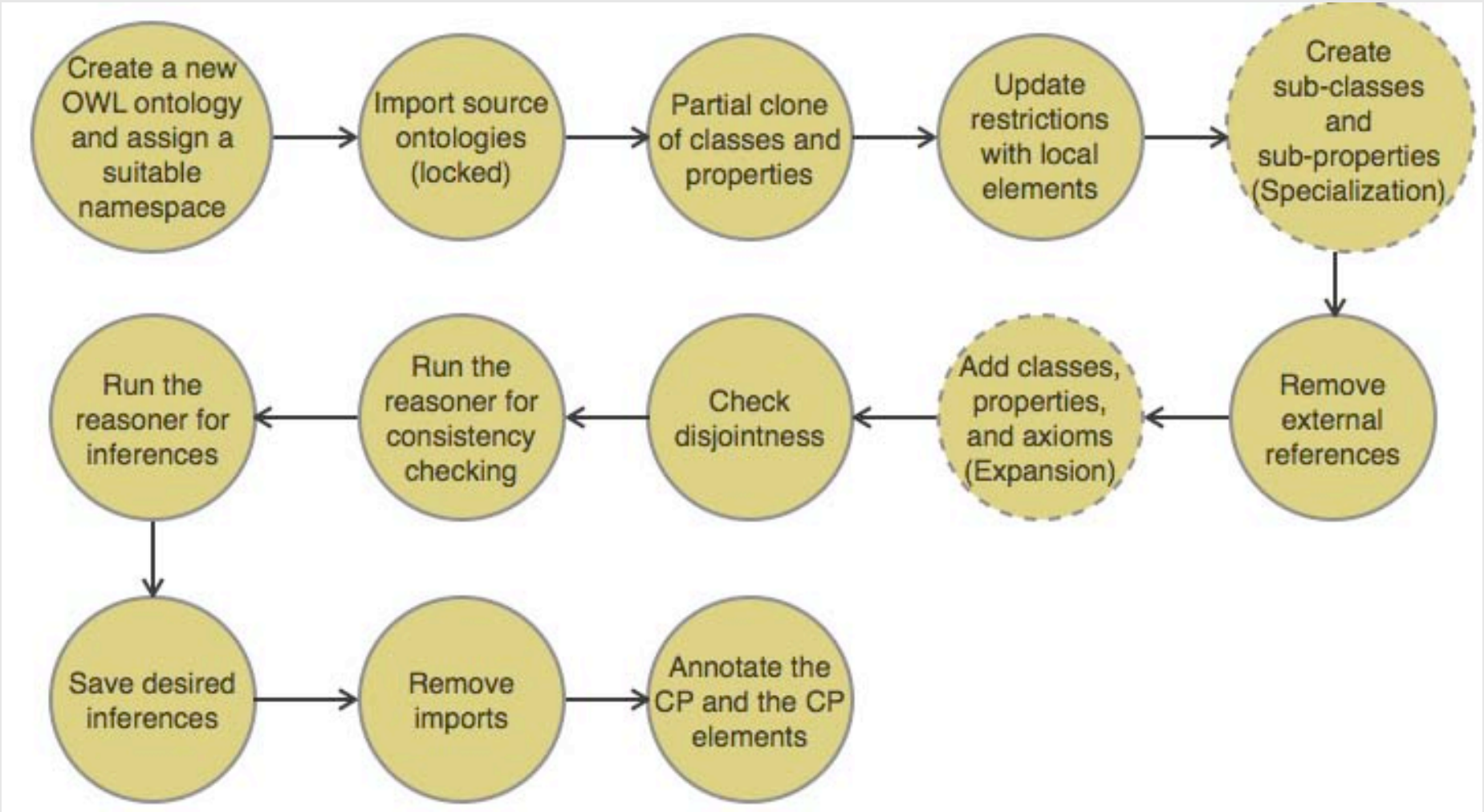
Clone

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

Types of clone operation

- Shallow clone
  - consists of creating a new ontology element oe_2 by duplicating an existing ontology element oe_1 . OWL restrictions of and axioms defined for oe_1 and oe_2 will be exactly the same
- Deep clone:
  - consists of creating a new ontology element oe_2 by duplicating an existing ontology element oe_1 , and by deep-cloning a new ontology element for each one that is referred in oe_1 's axiomatization, recursively
- Partial clone:
  - consists of deep-cloning an ontology element, but by keeping only a subset of its axioms, and of partial-cloning the kept elements, recursively.
- Some ontology design tools support the shallow clone operation
  - e.g., TopBraid Composer
- Deep clone and partial clone are not yet supported by any existing tool.

The extraction process



CP definition (finally!)

Definition

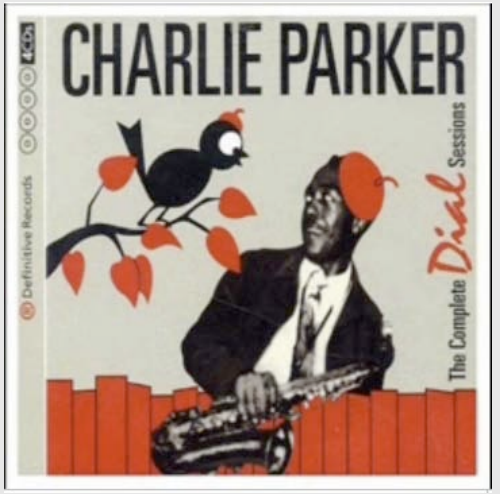
- CPs are distinguished networked ontologies and have their own namespace
- They cover a specific set *CQ* of competency questions (requirements), which represent the problem they provide a solution for
- A CP emerges from existing conceptual models and can be **extracted from** a reference ontology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

Pattern-based ontology design method: eXtreme ontology Design (XD)

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

Sample XD iteration

- Sentence: *Charlie Parker is the alto sax player on Lover Man, Dial, 1946*
  - Charlie Parker (person) *Alternative abstractions do exist!*
  - the alto sax player (player role)
  - on Lover Man (tune)
  - Dial (publisher)
  - 1946 (recording year)
- CQs
  - what persons do play a musical instrument?
  - on what tune?
  - for what publisher?
  - in what recording year?
- Queries
  - SELECT ?x ?y WHERE { ?x ?r ?y . ?x a :Person . ?y a :PlayerRole }
  - SELECT ?x ?z WHERE { ?x ?r ?y . ?x a :Person . ?x ?s ?z . ?z a :Tune }
  - SELECT ?z ?w WHERE { ?z ?t ?w . ?z a :Tune . ?w a :Publisher }
  - SELECT ?z ?k WHERE { ?z :recordingYear ?k . ?z a :Tune . ?k a xsd:gYear }



cont.d

- Retrieve/Match cqs to CPs, or possibly propose new ones
  - agentrole.owl, timeindexedpersonrole.owl, timeinterval.owl, ...
- Specialize/Compose/Expand CPs to local cq terminology
  - person-playerrole, playing-instrument-on-a-tune, playing-on-a-tune-in-recordingyear
- Populate ABox
  - Person(CharlieParker), PlayerRole(AltoSaxPlayer), Tune(LoverMan), Session(LoverManWithParkerOnDial), ...
- Run unit test/Iterate until fixed
  - SELECT ?x ?y ?z ?w ?k
  - WHERE {
    - ?x ?r ?y .
    - ?x a :Person .
    - ?y a :PlayerRole .
    - ?x ?s ?z .
    - ?z a :Tune .
    - ?z ?t ?w .
    - ?w a :Publisher .
    - ?z :recordingYear ?k .
    - ?k a xsd:gYear }
  - ?x=CharlieParker ?y=AltoSaxPlayer ?z=LoverMan ?w=Dial ?k=1946

XD iteration with Content OPs

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

FOR EACH SENTENCE s_i

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

END FOR

- Integrate...