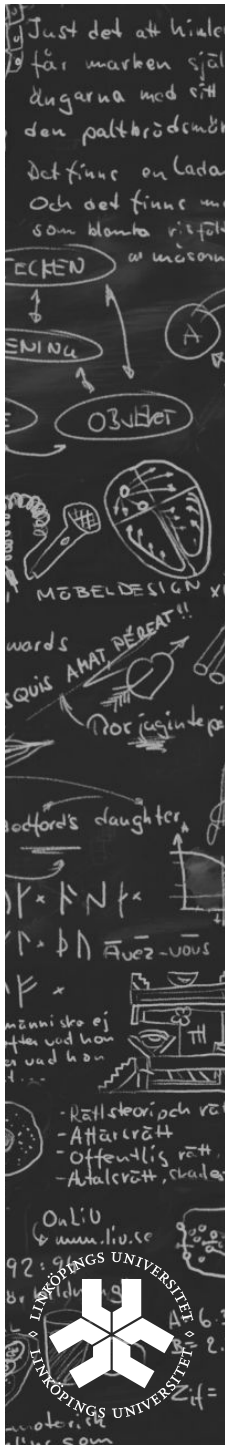
Requirements Engineering

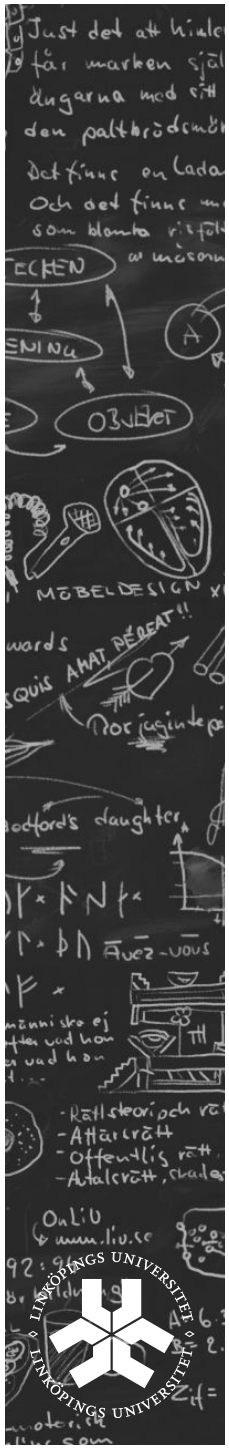
Eva Blomqvist
eva.blomqvist@liu.se
Department of Computer and Information Science (IDA)
Linköpings universitet
Sweden

LiU expanding reality



Why do we need requirements?

- Our focus is on “computational ontologies”
  - An ontology is usually a part of a software system, performing some specific tasks (through query- or inference engines)
- Two main perspectives
  - *Coverage* oriented ontologies
    - The important thing is to cover all the terms of the domain
    - Example: Formalizing a domain vocabulary, ontologies used in IR systems, CYC etc.
  - *Task* oriented ontologies
    - The important thing is to support particular queries or inferences
    - We have a software in mind when creating the ontology
    - Example: Ontology as a model for querying a DB, ontology for performing certain inferences etc.

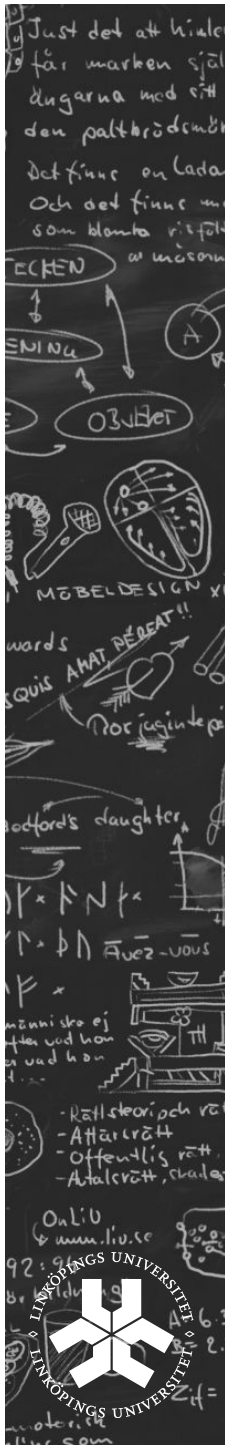


What are “requirements”?

- Viewing an ontology as a black box… what should that box provide?
- Functional requirements
  - Query results?
  - Inferences?
  - Error checking?
  - …
- Non-functional requirements
  - Coverage
  - Efficiency
  - Documentation
  - Changeability – extendibility
  - …

Internal structure, and content

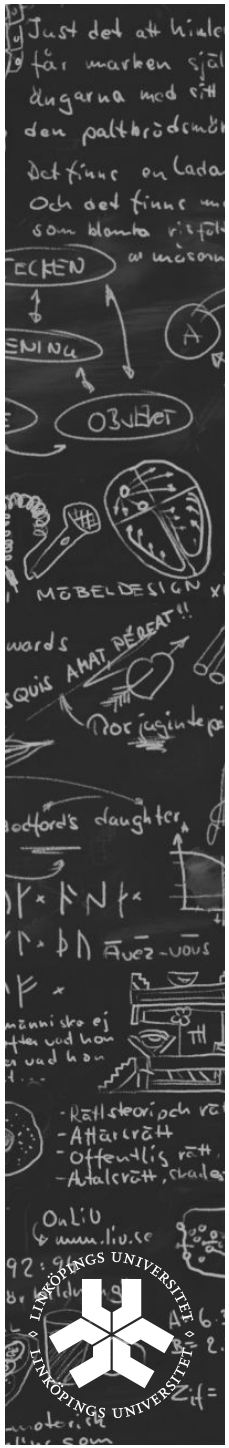
Overall structure, acceptance ➔ Guidelines and rules for development



Requirements Engineering – Competency Questions

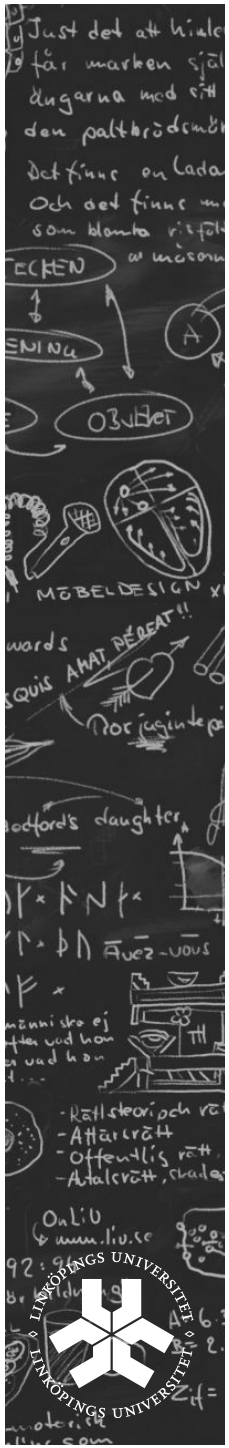
- Competency Questions (CQ) = Natural language questions that ask for information the ontology should be able to provide to a user (or system)
  - Functional requirements
  - Related to software requirements – “input” and “output” of the “ontology component” (including query engine, reasoner…)

- Different kinds
  - Simple lookup queries
    - Who are the participants of a certain course?
  - Expressing inferences or constraints
    - Given that people may have children, is a specific person a grandparent or not?
    - Is a person married to two people valid according to Swedish law?



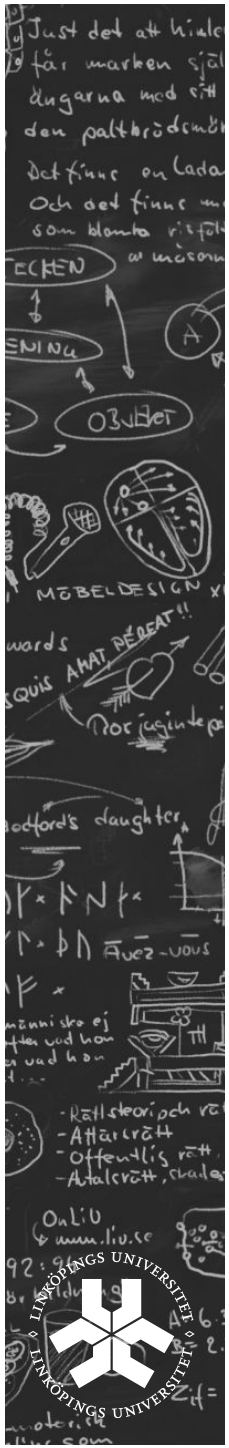
Requirements Engineering – Competency Questions (cont.)

- To clarify complex CQs we use
  - Contextual statements
  - Inference (reasoning) requirements
- A contextual statement expresses an axiom that needs to hold in the ontology, in natural language
  - Every course has at least one participant
  - A grandparent is someone who has a child who in turn also has a child
  - In Sweden you can only marry one person
- Reasoning requirements specify the input and output data for a reasoning task
  - We would like to be able to query directly for all the grandparents – classification based on the axiom above



Requirements Engineering – Non-functional Requirements

- Coverage
  - How important is the coverage of the domain?
    How will the ontology be updated?
- Efficiency
  - What OWL profile to use?
  - Reasoning off-line or online?
  - Query optimization, e.g. not requiring inferences
- Documentation
  - Labels and comments?
  - Naming conventions
- Changeability – extendibility
  - Should future extensions be prepared for?
  - Alignment to online ontologies, standards?



Trade-off: Software vs. Ontology

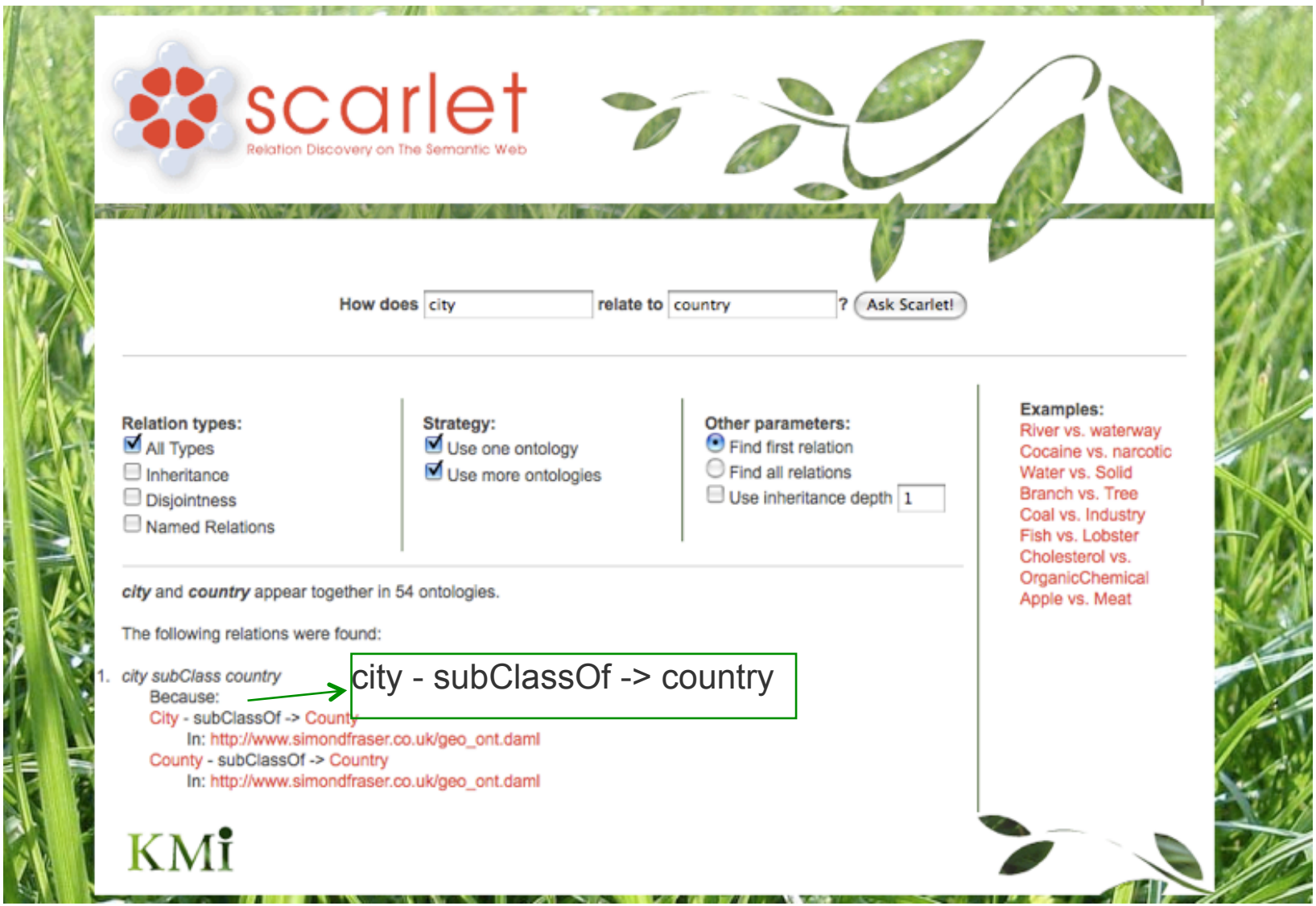
- What functionality is going to be put into the software and which is going to be part of the ontology?
  - An OWL reasoner is nothing more than general-purpose code for processing data – why not use specific code in our system instead?
- Ontology pro:s
  - The ontology makes assumptions explicit
  - The ontology can be changed at runtime without changing the code (or with minimal changes)
  - The reasoning procedures are sound and well-defined, and they are reused for all inferences
- Software pro:s
  - More efficient

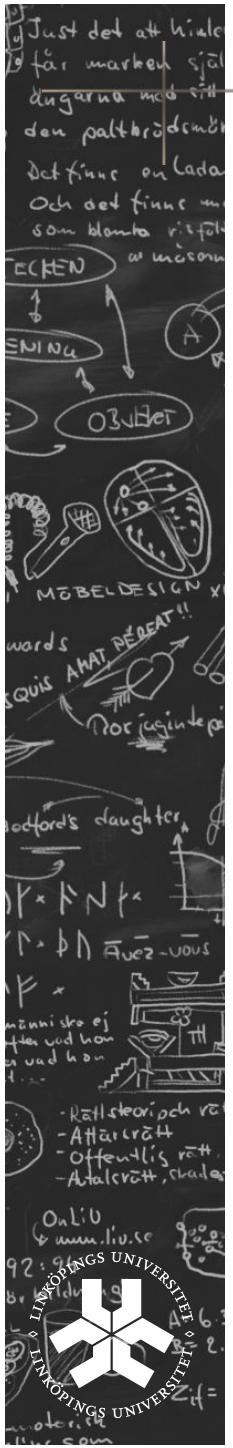
Ontology Design Patterns

Eva Blomqvist
eva.blomqvist@liu.se
Department of Computer and Information Science (IDA)
Linköpings universitet
Sweden

Slides partly by Valentina Presutti, STLab, ISTC-CNR, Italy

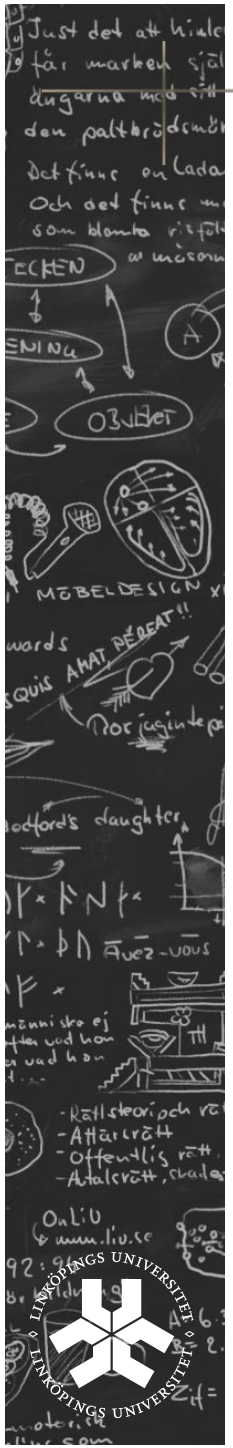
LiU expanding reality


scarlet
Relation Discovery on The Semantic Web
How does city relate to country ? Ask Scarlet!
Relation types:
All Types
Inheritance
Disjointness
Named Relations
Strategy:
Use one ontology
Use more ontologies
Other parameters:
Find first relation
Find all relations
Use inheritance depth 1
Examples:
River vs. waterway
Cocaine vs. narcotic
Water vs. Solid
Branch vs. Tree
Coal vs. Industry
Fish vs. Lobster
Cholesterol vs. OrganicChemical
Apple vs. Meat
city and country appear together in 54 ontologies.
The following relations were found:
1. city subClass country
Because:
City - subClassOf -> County
In: http://www.simondfraser.co.uk/geo_ont.daml
County - subClassOf -> Country
In: http://www.simondfraser.co.uk/geo_ont.daml
city - subClassOf -> country
KMi



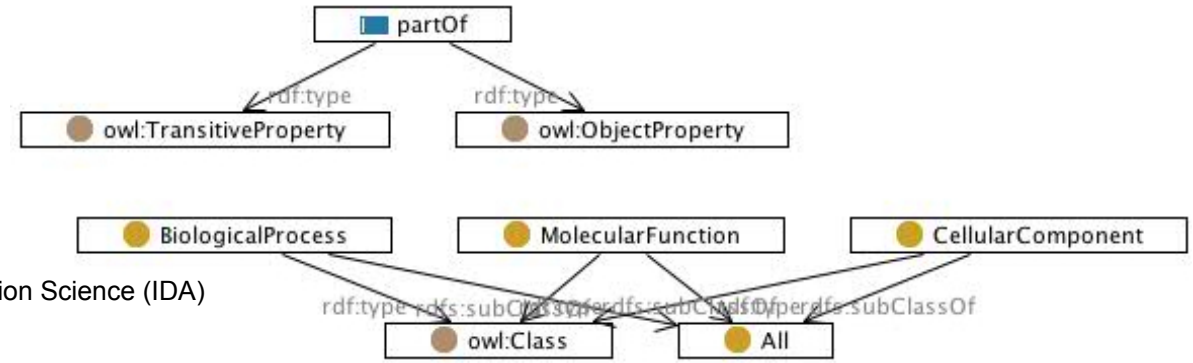
What we can do with OWL

- ... (maybe) we can check the consistency, classify, and query our knowledge base
- ... but, remember the Scarlet example
  - City subClassOf Country
- Logical consistency is not the main problem
  - e.g. owl:sameAs can be wrongly used and still we have consistency
- Why is OWL not enough?
  - OWL gives us logical language constructs, but does not give us any guidelines on how to use them in order to solve our tasks.
  - E.g. modeling something as an individual, a class, or an object property can be quite arbitrary

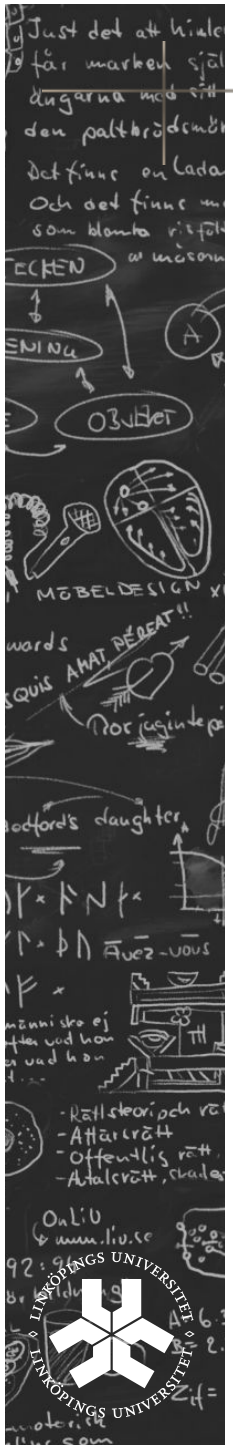


Solutions?

- OWL is not enough for building a good ontology, and we cannot ask all web users neither to learn logic, or to study ontology design
- Reusable solutions are here through Ontology Design Patterns, which help reducing arbitrariness without asking for sophisticated skills ...
- ... provided that tools are built for any user ☺

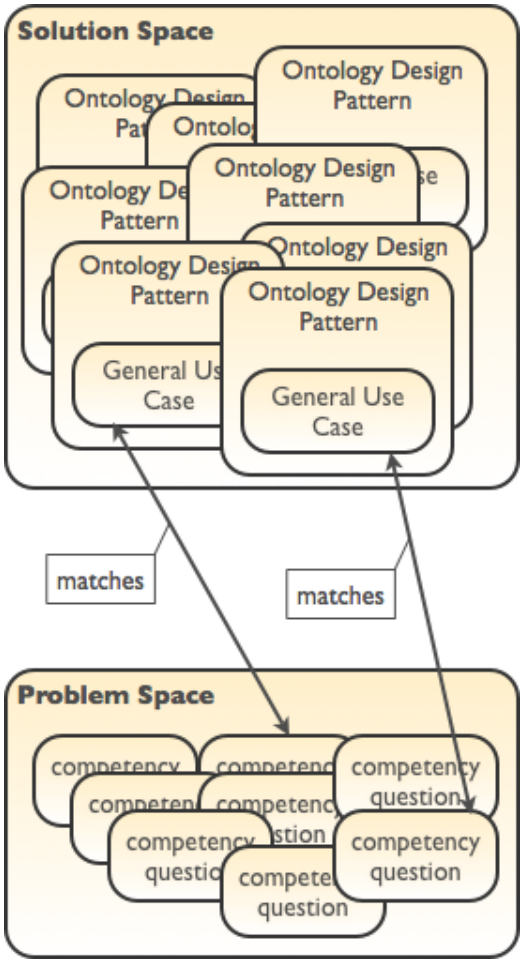


Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden



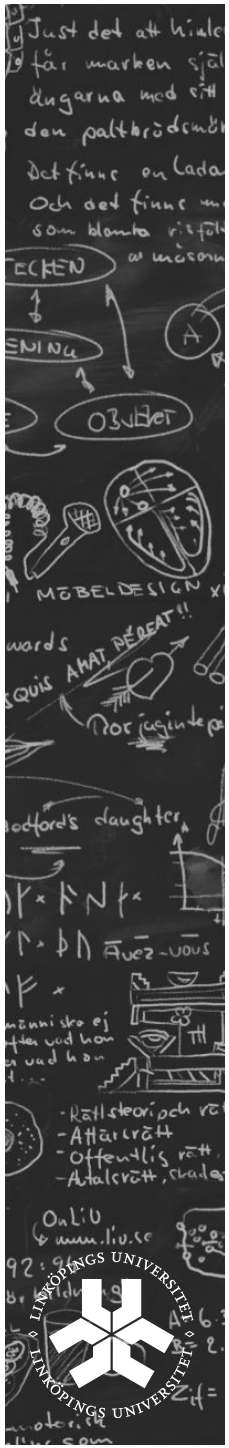
Ontology Design Patterns

An ontology design pattern is a reusable successful solution to a recurrent modeling problem



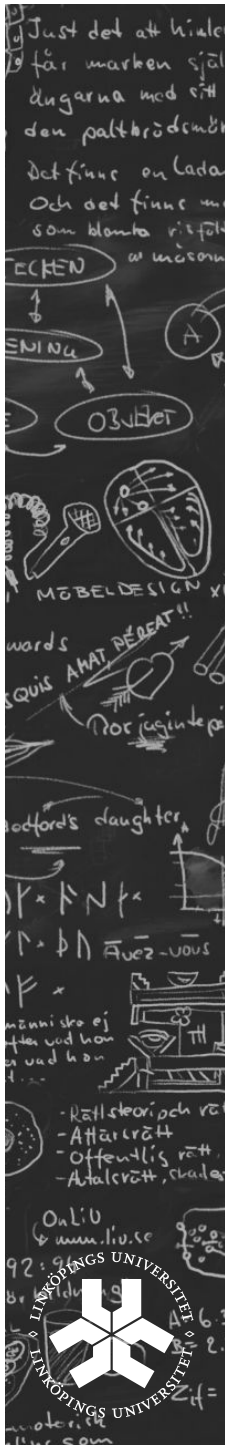
Logical Ontology Design Patterns

LiU expanding reality



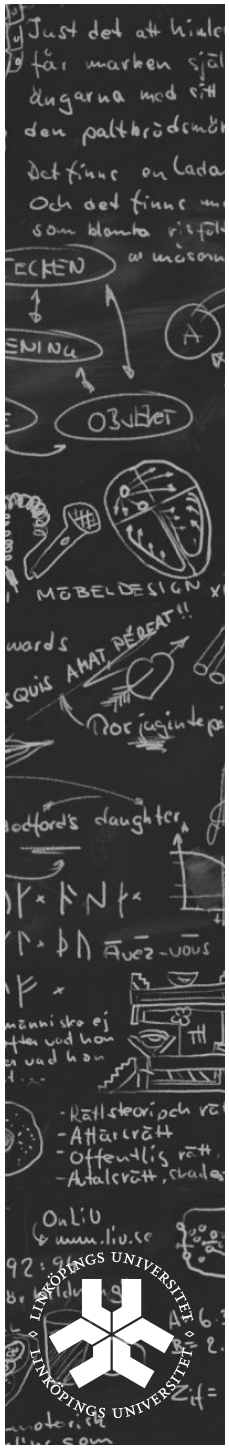
Logical ODPs

- *Definition*
  - A Logical ODP is a formal expression, whose only parts are expressions from a logical vocabulary e.g., OWL, that solves a problem of expressivity
- Logical ODPs are independent from a specific domain of interest
  - i.e. they are content-independent



Logical ODPs

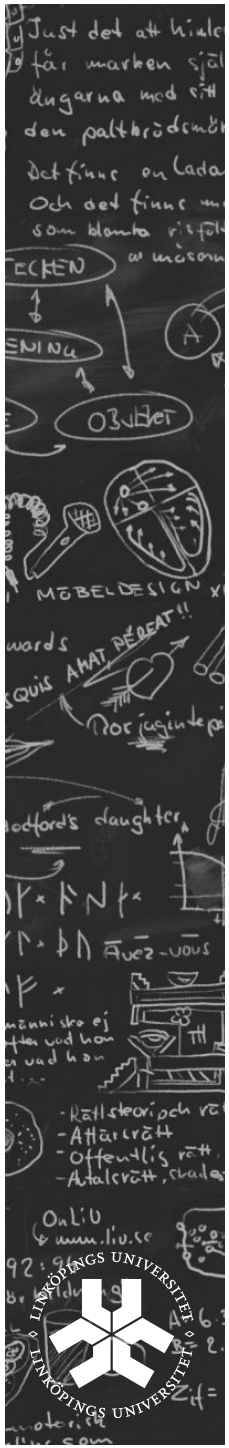
- A Logical ODP describes a formal expression that can be *exemplified, morphed, and instantiated* in order to solve a domain modeling issue
- **owl:Class:_:x rdfs:subClassOf owl:Restriction:_:y**
- *Inflammation rdfs:subClassOf (localizedIn some BodyPart)*
- *Colitis rdfs:subClassOf (localizedIn some Colon)*
- *John's_colitis localizedIn John's_colon*



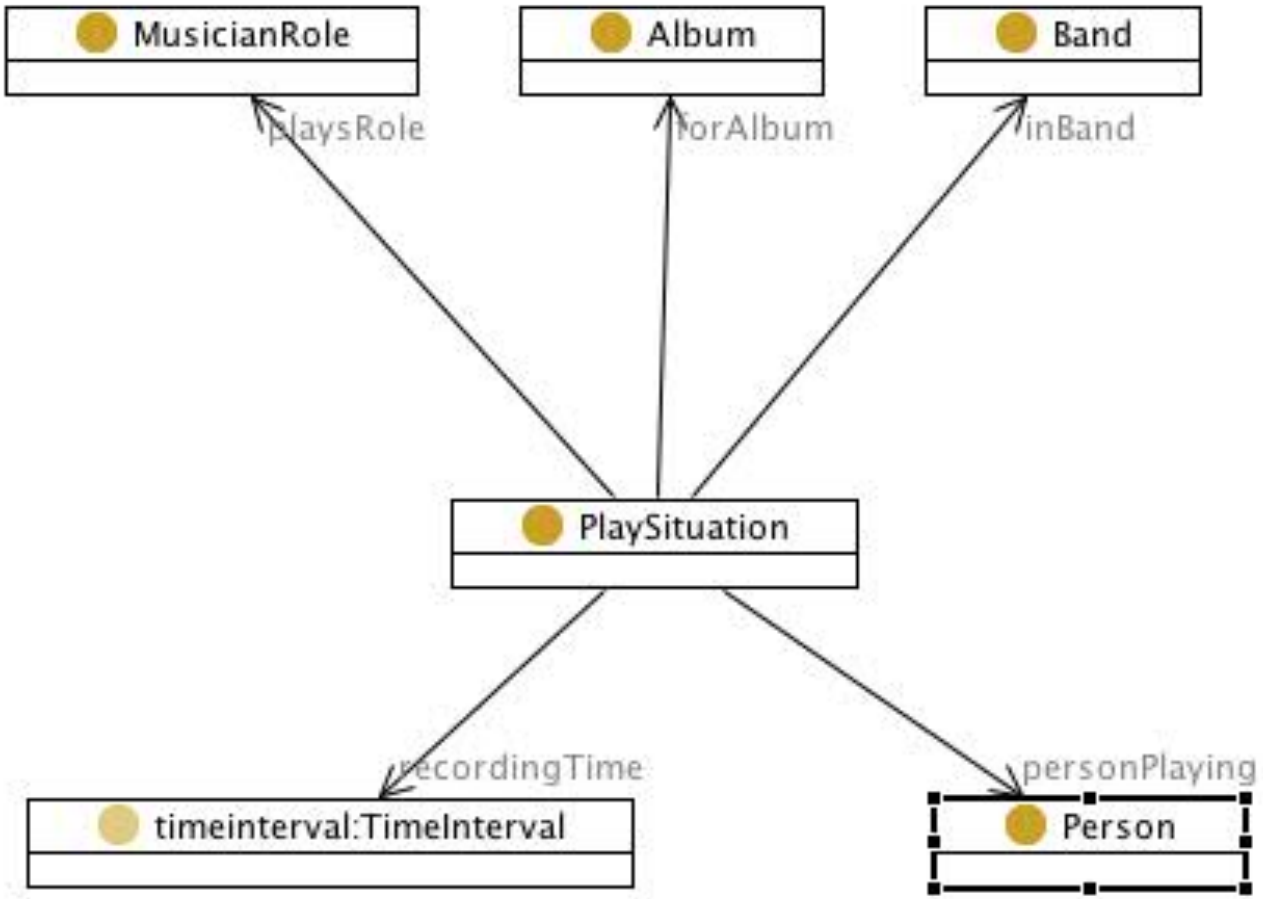
Example: N-ary relation

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

- N-ary relation:
  - PlaySituation(Person, MusicianRole, Band, Album, TimeInterval)
  - How can we express this in OWL with only binary relations?



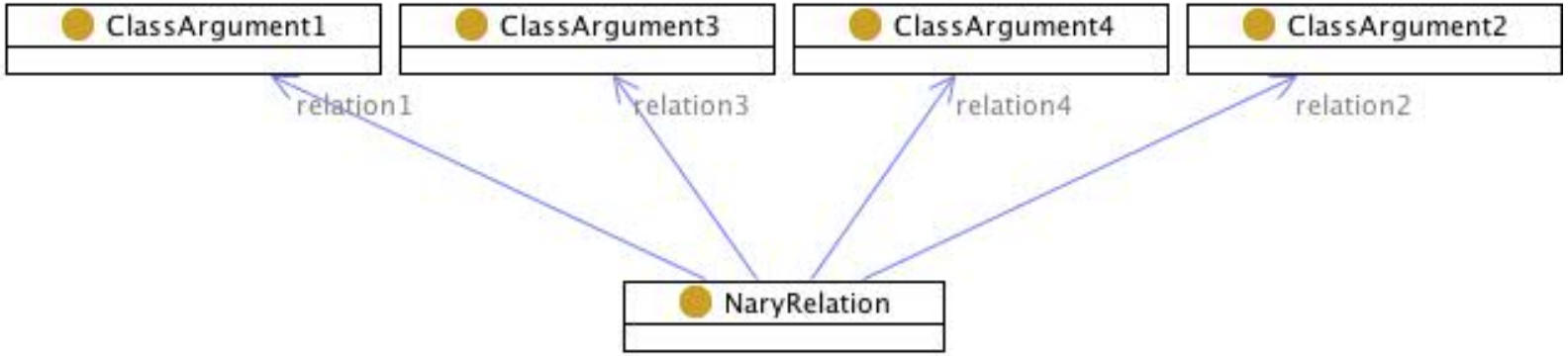
Transformation ODPs
Example: N-ary relation



Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

August 17, 2012

Transformation ODPs
Example: N-ary relation

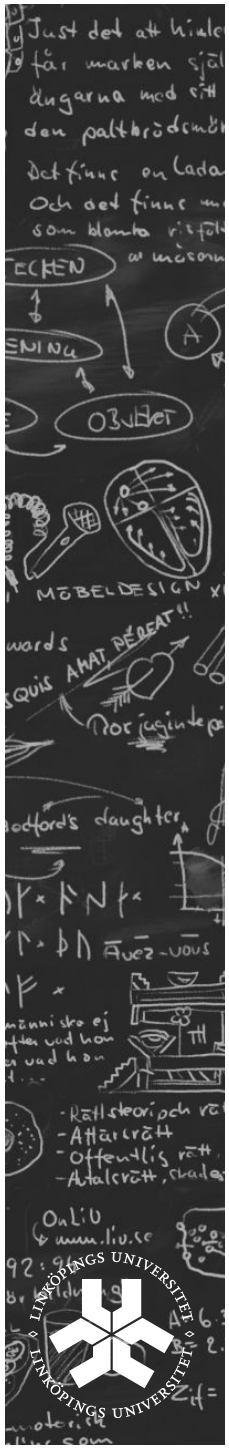


Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

August 17, 2012

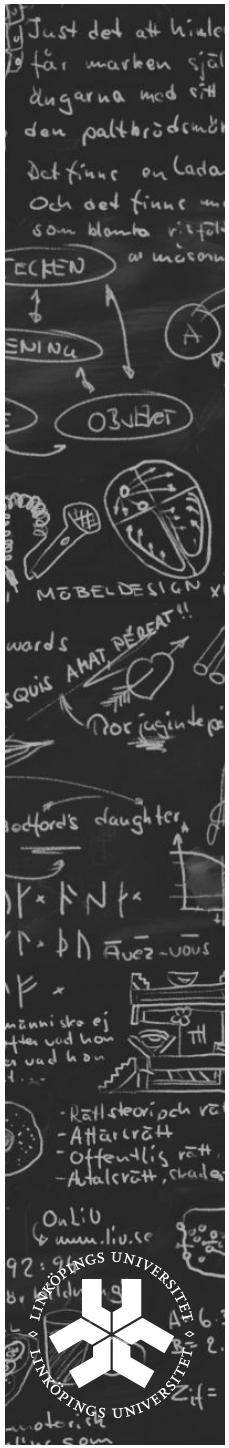
Content ODPs

LiU expanding reality



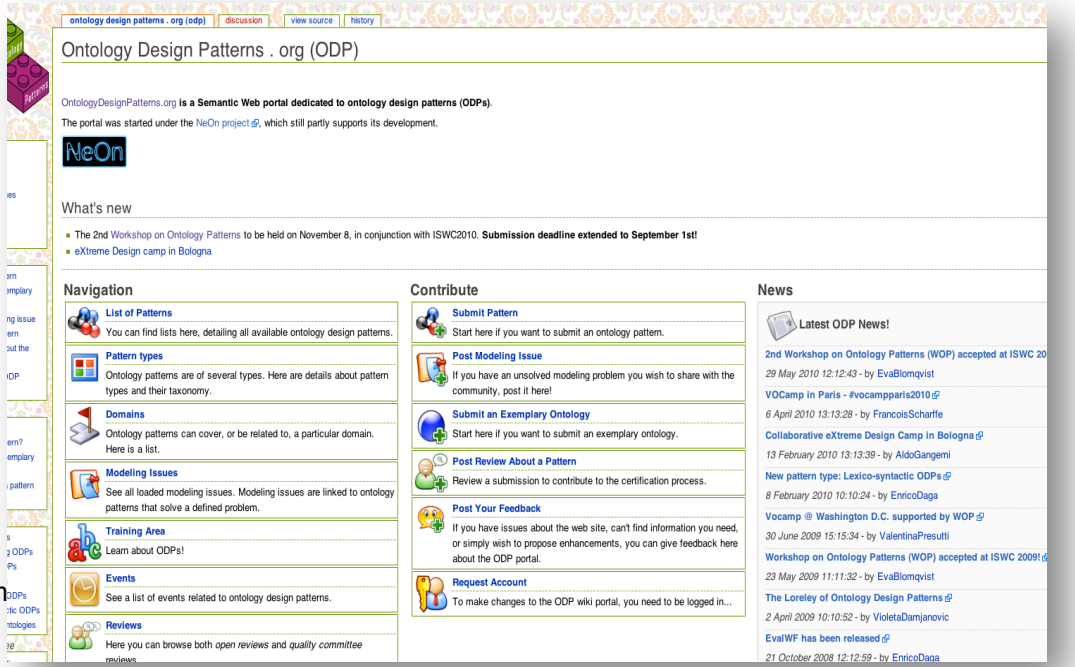
Content ODPs (CPs)

- CPs encode conceptual, rather than logical design patterns.
  - Logical ODPs solve design problems independently of a particular conceptualization
  - CPs are patterns for solving design problems for the domain classes and properties that populate an ontology, therefore they address content problems
- CPs are instantiations of Logical ODPs (or of compositions of Logical ODPs), featuring a non-empty signature
  - Hence, they have an explicit non-logical vocabulary for a specific domain of interest, i.e. they are content-dependent



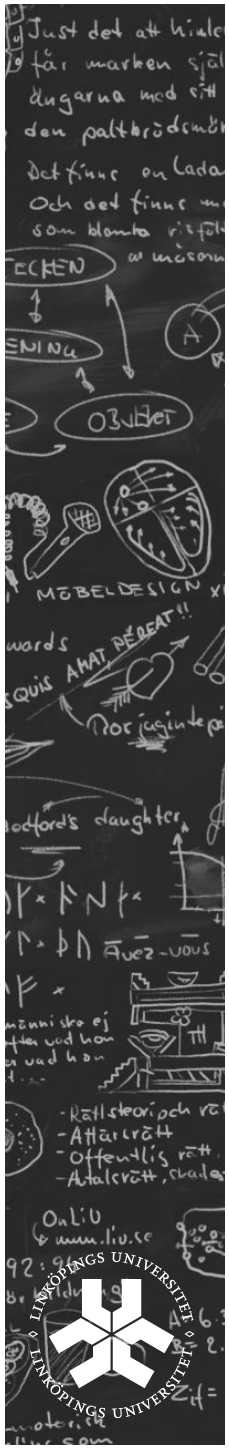
Catalogues of CPs

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



Department of Computer and Inform
Linköpings universitet, Sweden

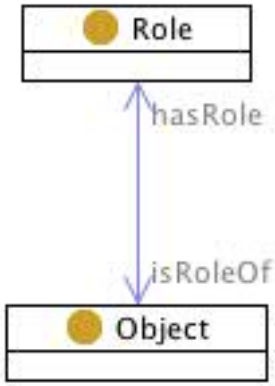
August 17, 2012

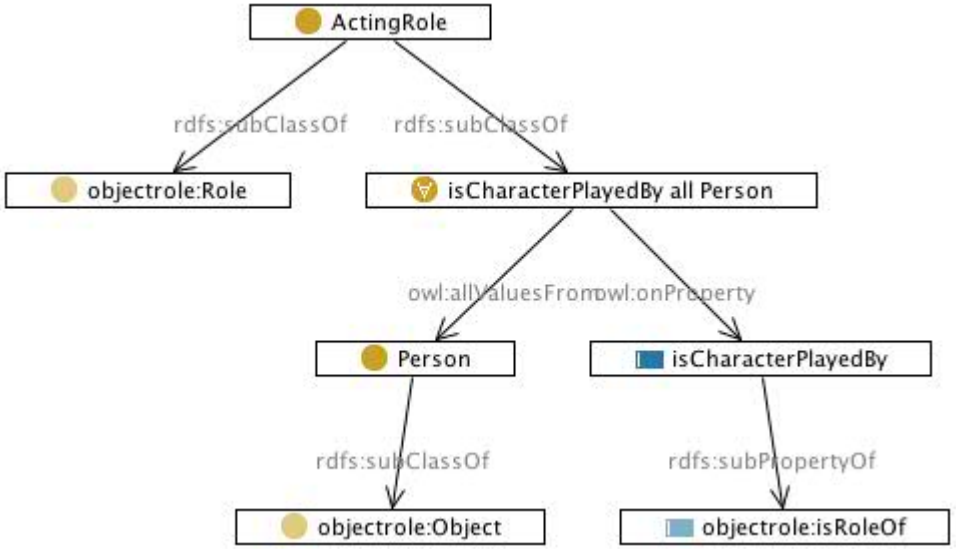


Pragmatic characteristics of CPs

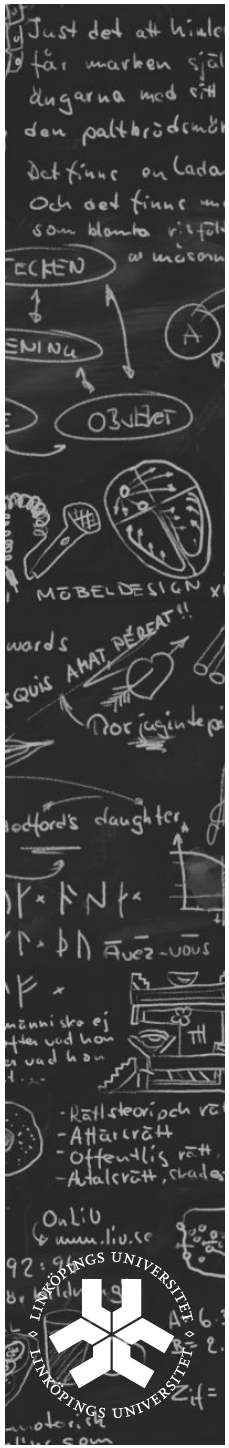
- Domain-dependent
  - Expressed with a domain-specific (non-logical) vocabulary
- Requirement-covering
  - Solve domain modeling problems (expressible as use-cases, tasks or “competency questions”), at a typical maximum size (cf. blink)
- Reasoning-relevant components
  - Allow some form of inference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

Sample Specialization



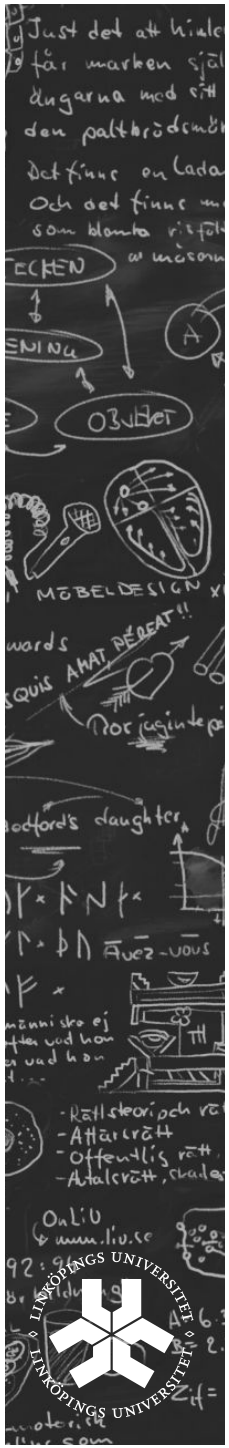


- A content pattern CP_2 specializes CP_1 if at least one ontology element of CP_2 is subsumed by an ontology element of CP_1
  - i.e., either by *rdfs:subClassOf* or *rdfs:subPropertyOf*



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

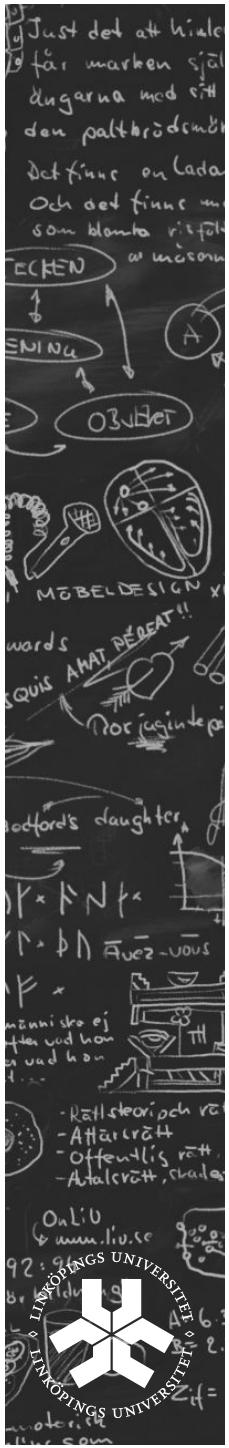


General Content ODPs

- Roles of objects
- Classification
- Part-whole relationships
- Membership
- Information and its realization
- Sequences
- Topics
- Time
- Places
- Moving
- Plans
- Events
- Descriptions and Situations

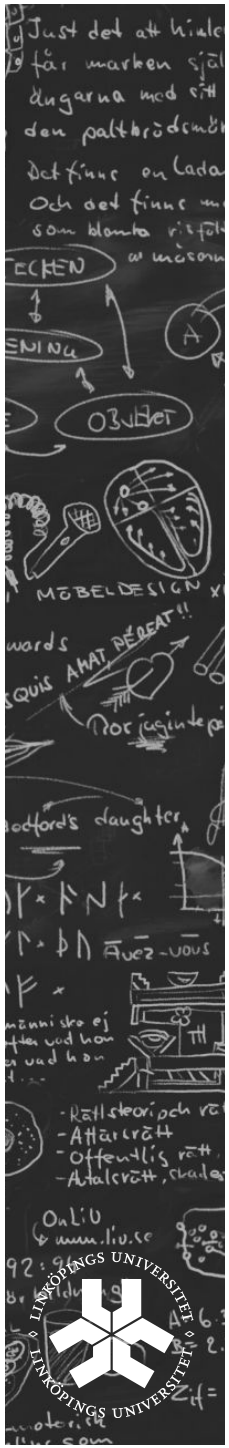
Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

August 17, 2012



Example: Roles of objects

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

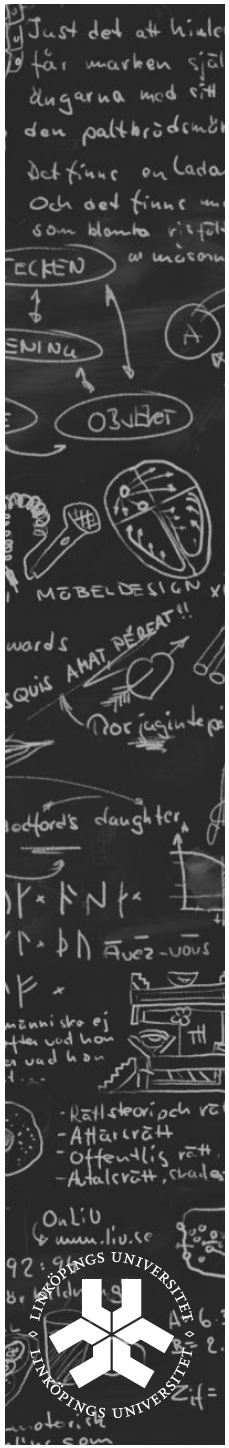


Roles of objects

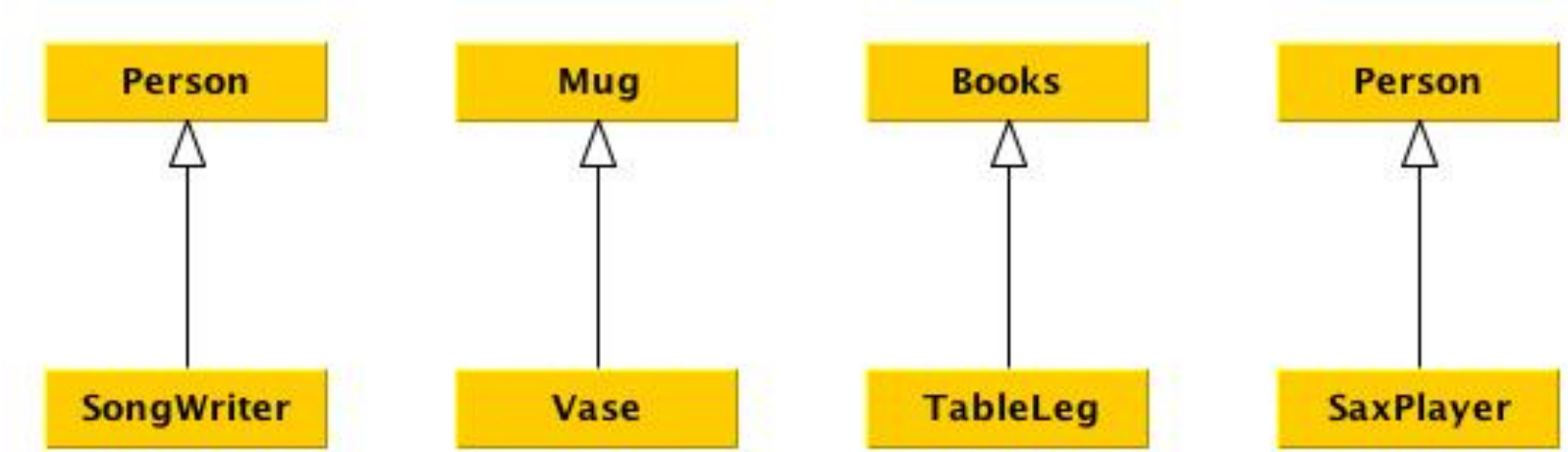
- A beer mug used as vase
- Books used as table's legs
- A sax player (person)
- A song writer (person)

Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

August 17, 2012

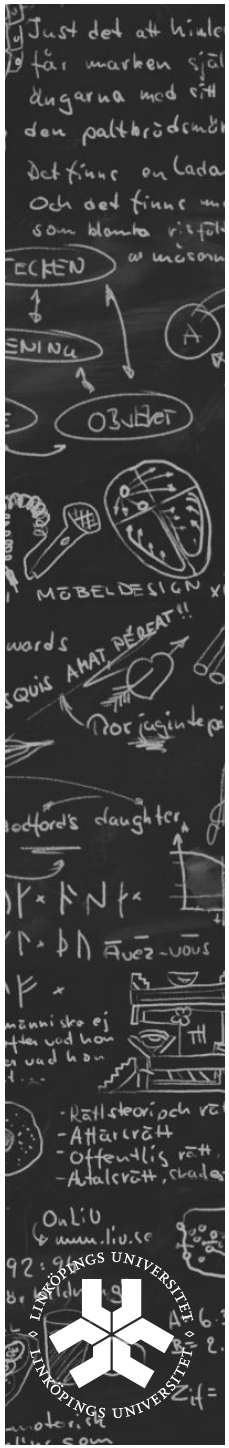


1st ODP: Roles as classes



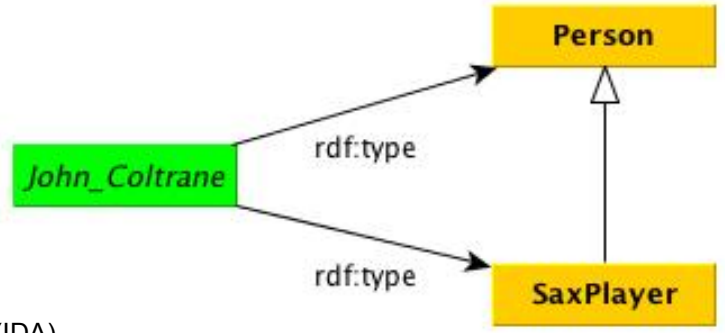
Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

August 17, 2012

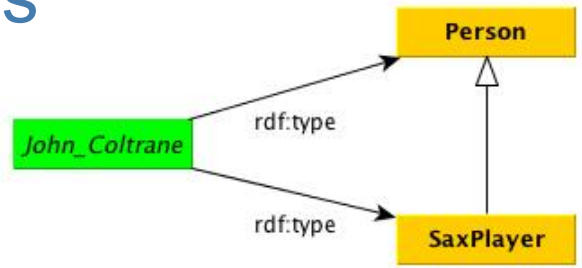


1st ODP: Roles as classes

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

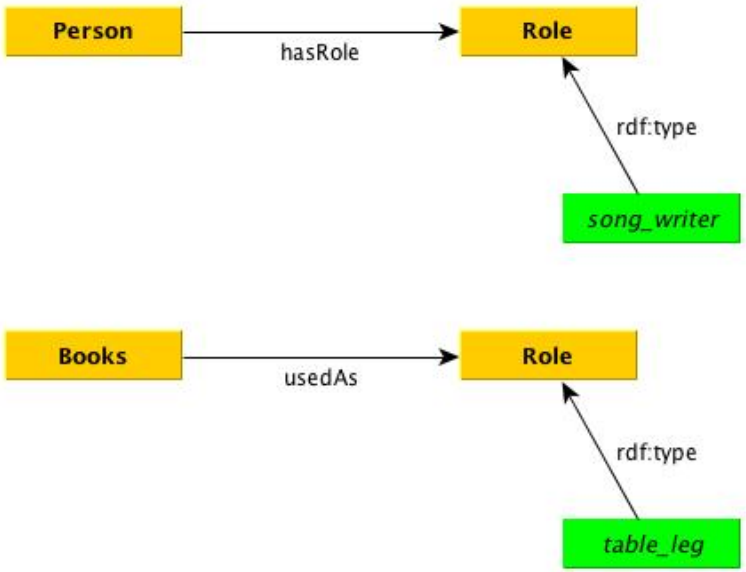


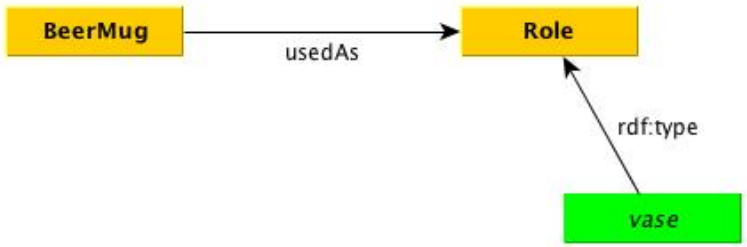
1st ODP: Roles as classes

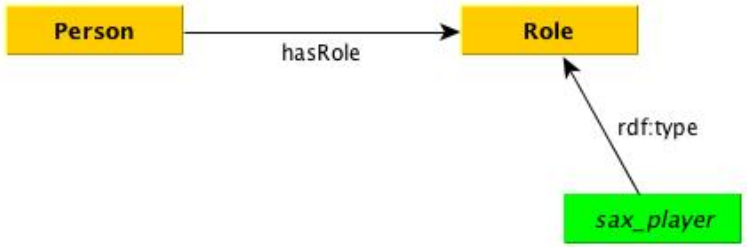


- Consequences
  - Low expressivity
  - Roles are described at TBox level
  - Class taxonomy is bigger - a class for each role
  - Class taxonomy is entangled - multi-typing
  - ABox is smaller – same individual, several (role) types
  - Automatic classification of individuals through rdfs:subClassOf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

2nd ODP: Roles as individuals





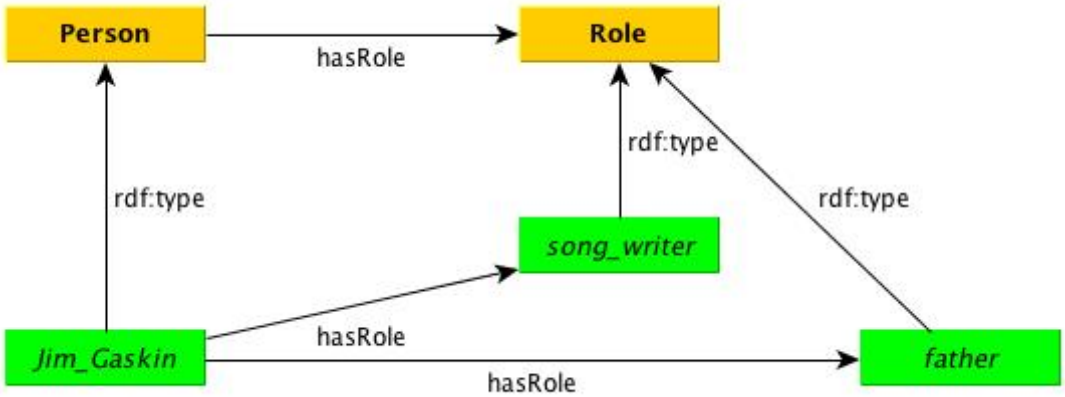


Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

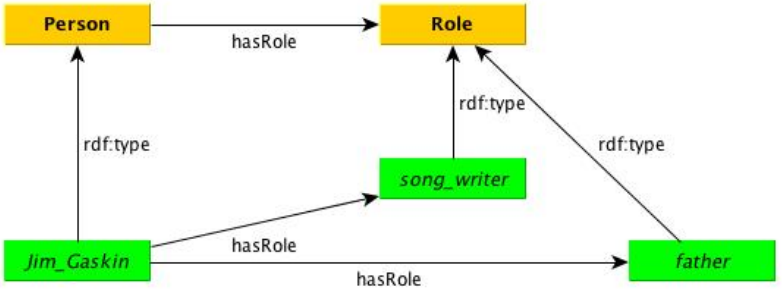
August 17, 2012

2nd ODP: Roles as individuals

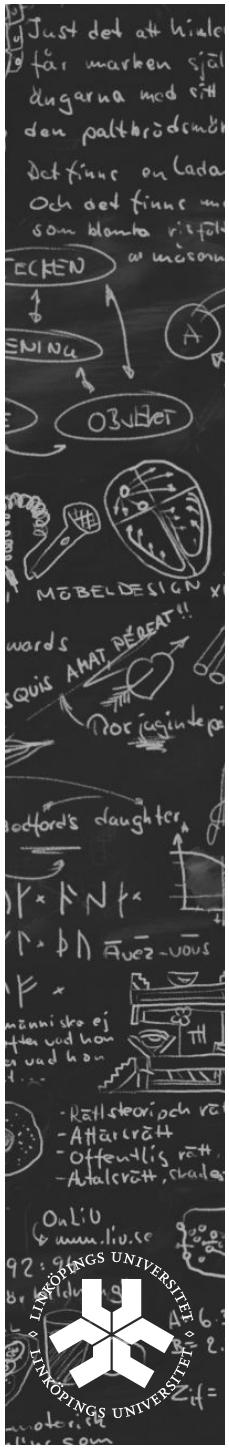
- An object and its roles are related through domain-specific relations
- Relations between an object and its roles have to be asserted
- Automatic inference of relations between an object and its roles can be obtained through property subsumption



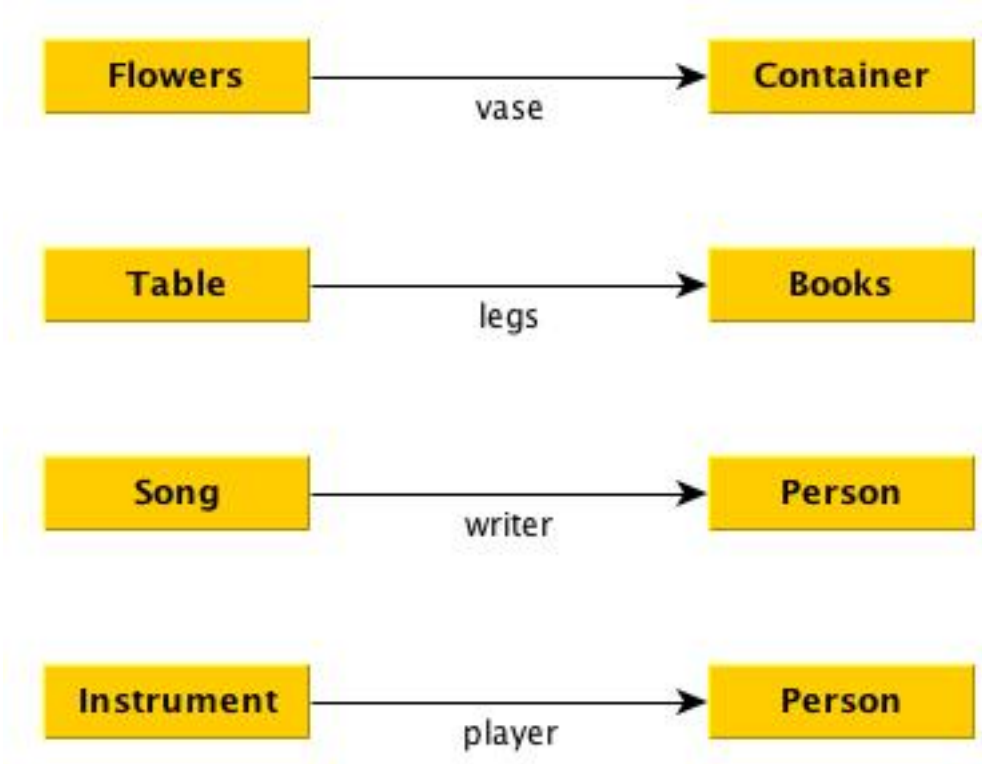
2nd ODP: Roles as individuals



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

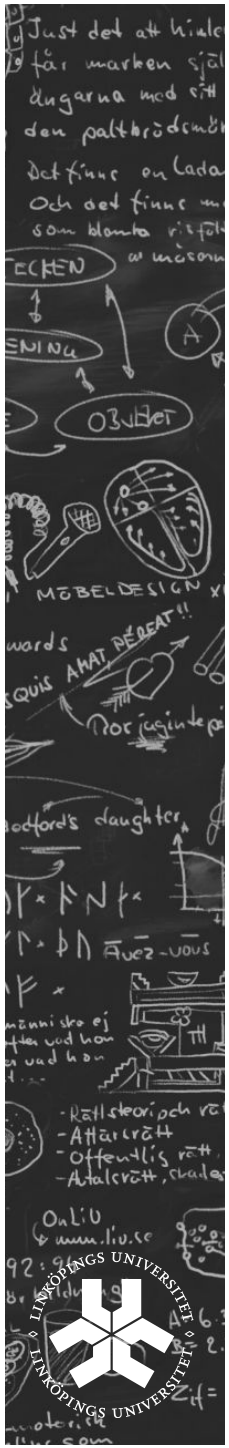


3rd ODP: Roles as properties



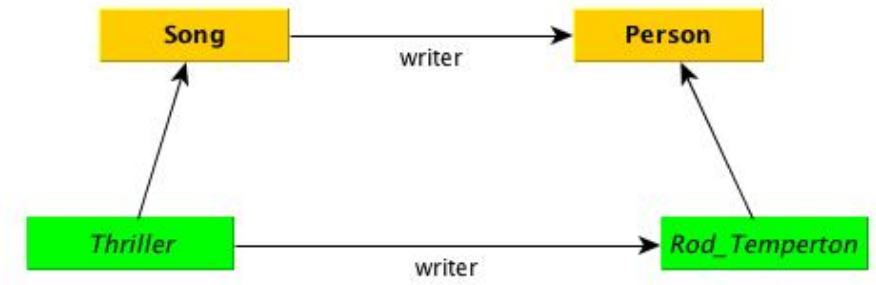
Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

August 17, 2012



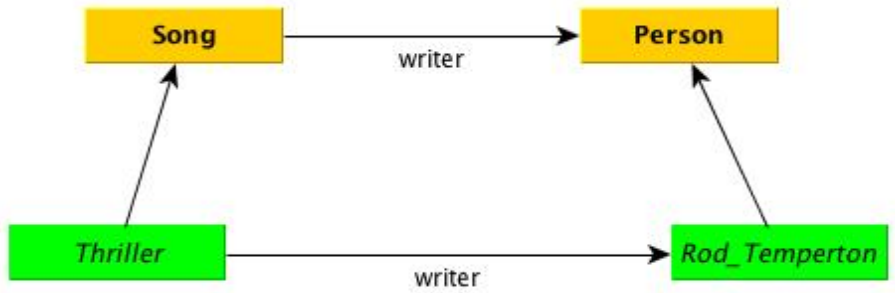
3rd ODP: Roles as properties

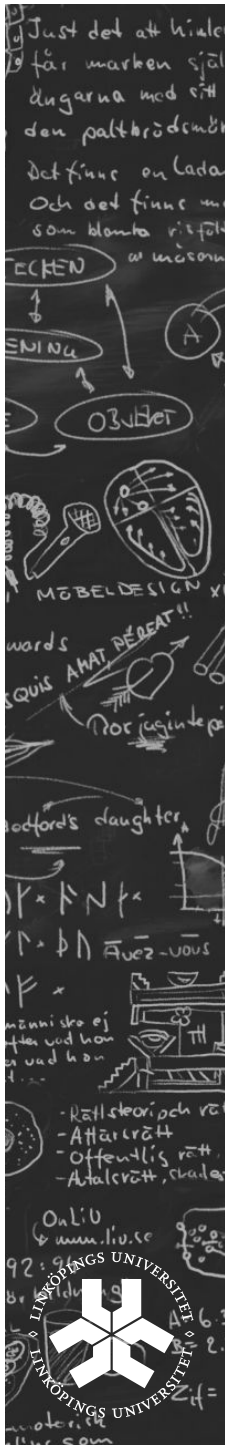
- The semantics of “having a role” is embedded in the name of a property
- Objects are not explicitly related to their roles, they are related to other things through a property expressing an action they perform, a role they play
- Most common pattern in the web of data for modeling roles



3rd ODP: Roles as properties

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



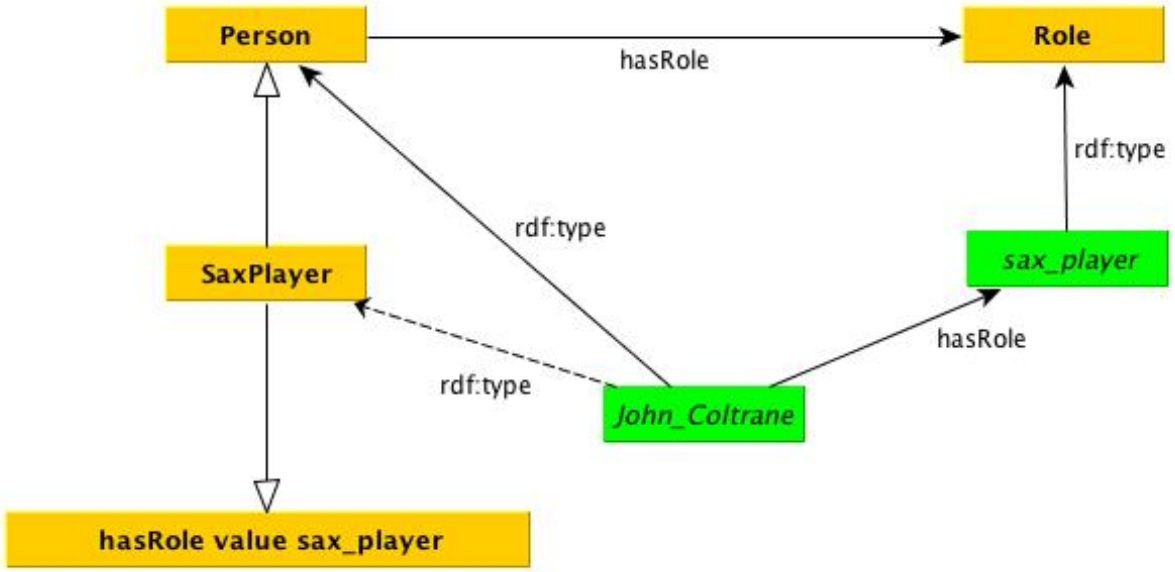


ODPs for Roles of objects - Summary

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

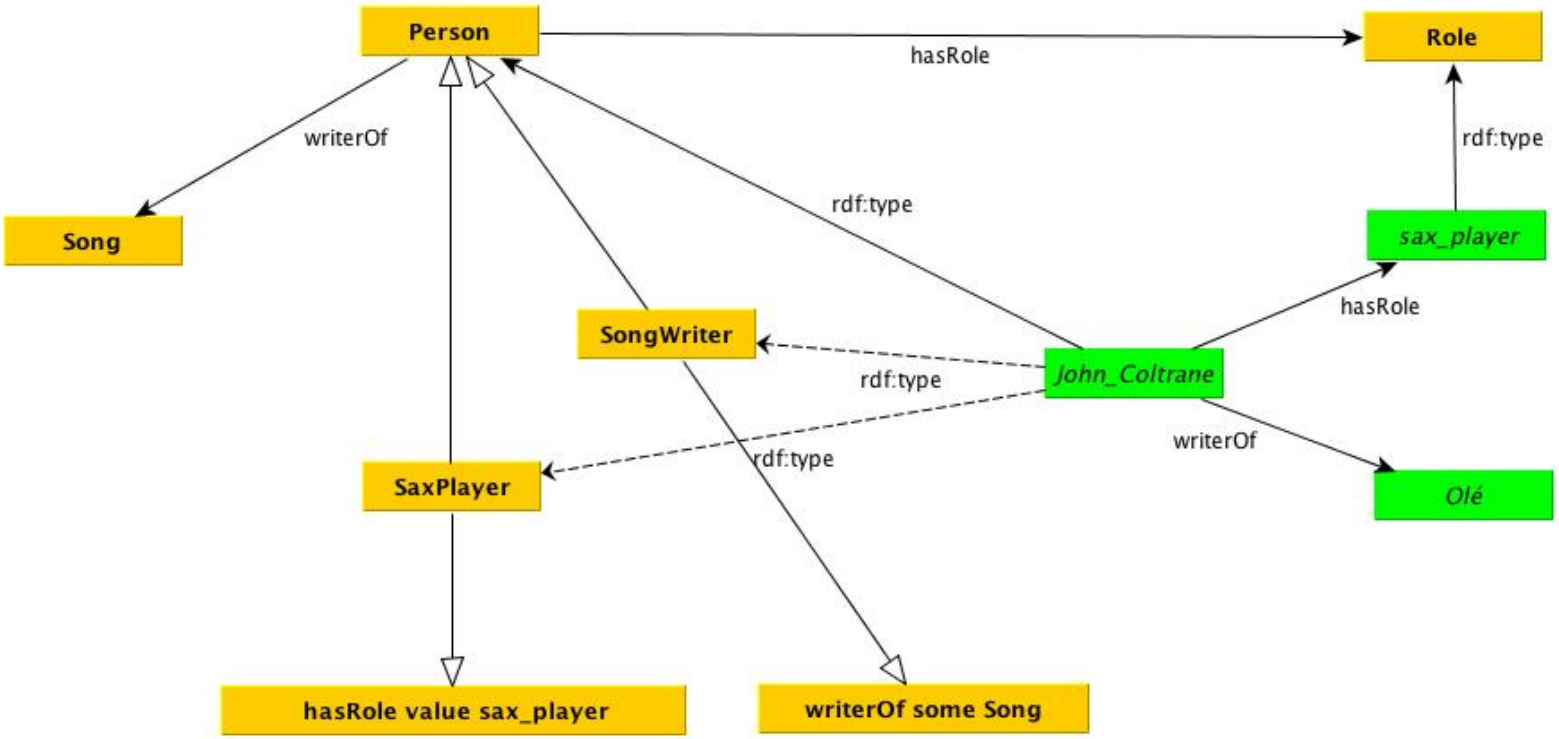
Combining roles as instances with roles as classes

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

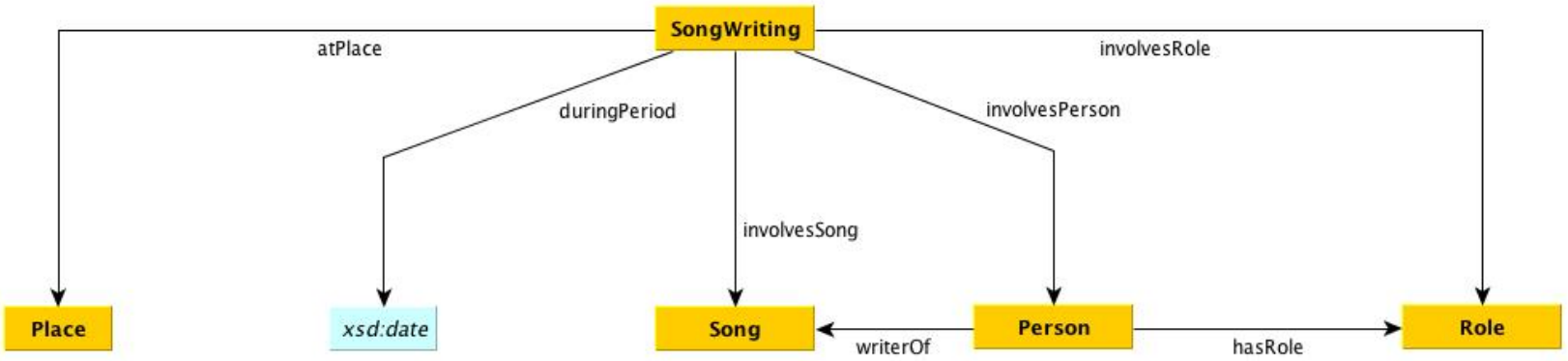


…and add roles as properties

- Note the restriction on property writerOf



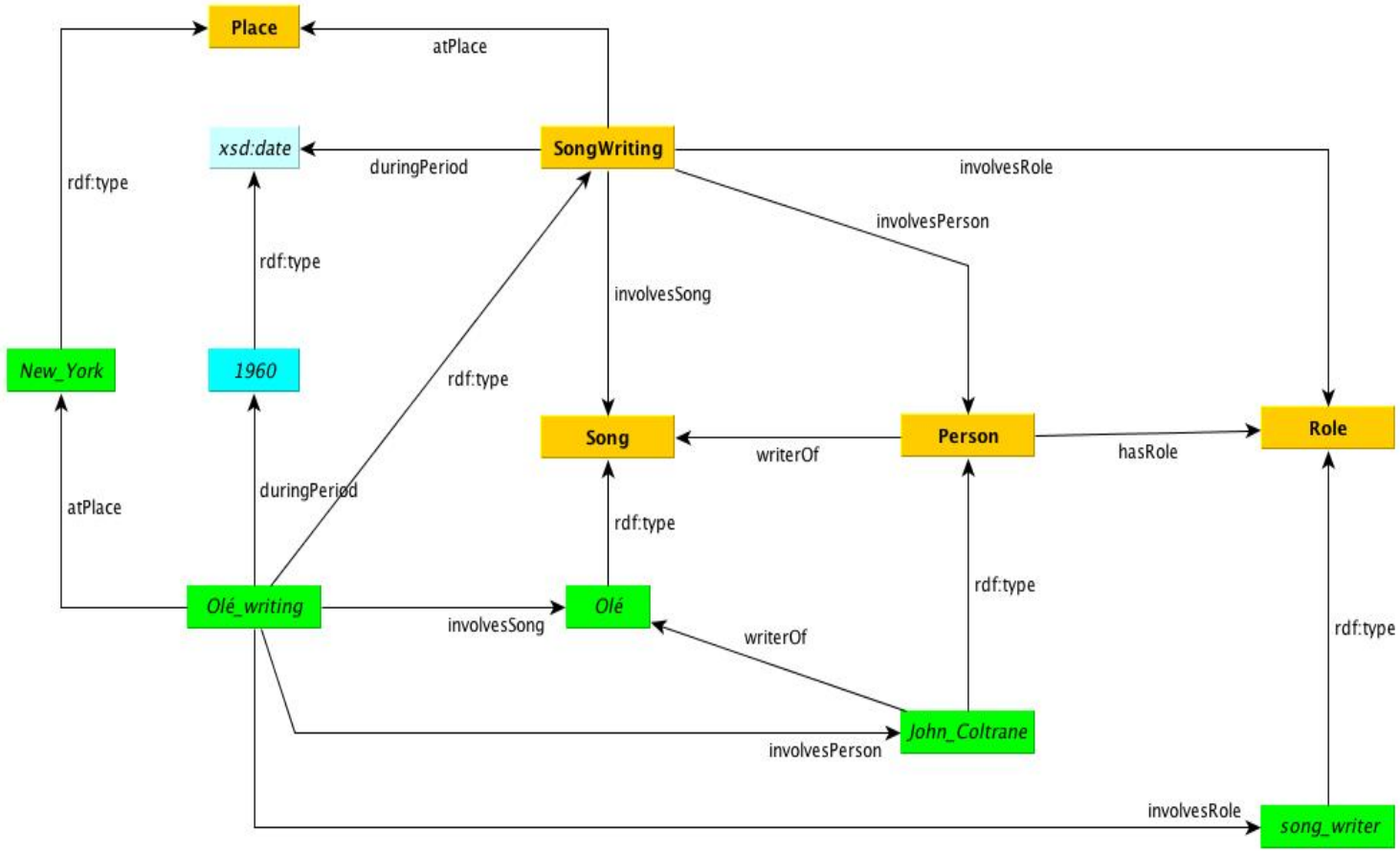
Indexing roles in terms of time and space



Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

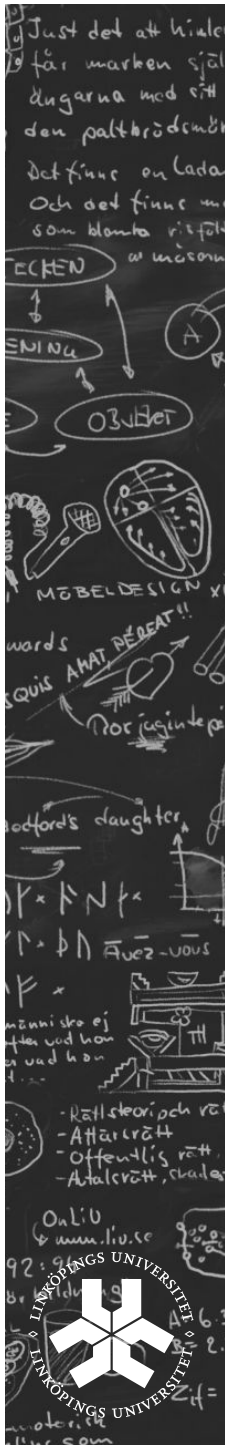
August 17, 2012

Indexing roles in terms of time and space



Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

August 17, 2012



Content ODPs for roles of objects

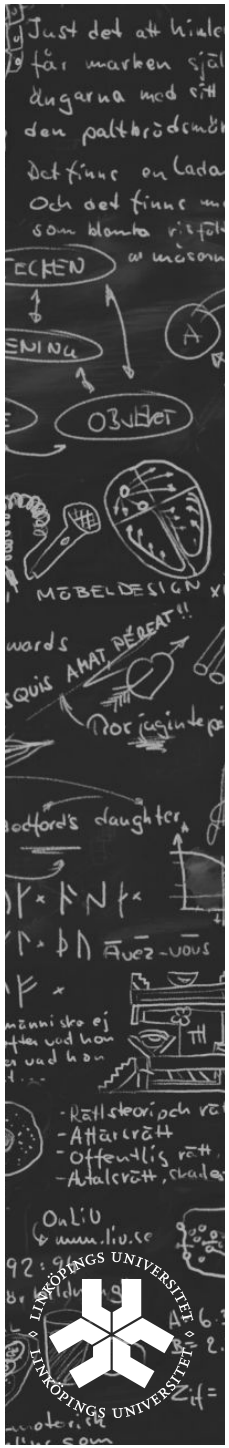
- The general pattern is called “classification”
- Object-Role and Agent-Role
  - OWL pattern representing roles as individuals
  - http://ontologydesignpatterns.org/wiki/Submissions:Objectrole
  - http://ontologydesignpatterns.org/wiki/Submissions:AgentRole
- Time-indexed person role-pattern
  - http://ontologydesignpatterns.org/wiki/Submissions:Time_indexed_person_role
- Time-place-indexed object-role
  - N-ary relation representing an objects, the roles it plays at a certain date in a certain place
  - http://www.ontologydesignpatterns.org/cp/owl/dul/timeplaceindexedobjectrole.owl

Methodologies
- Exemplified through XD

Eva Blomqvist
eva.blomqvist@liu.se
Department of Computer and Information Science (IDA)
Linköpings universitet
Sweden

Slides partly by Valentina Presutti, STLab, ISTC-CNR, Italy

LiU expanding reality

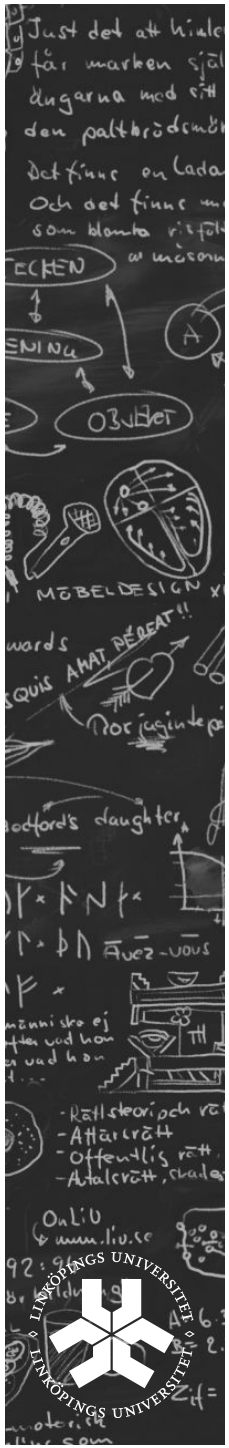


Ontology Engineering Methodologies

- Mostly focus has been on overall life-cycle and "model" of the methodology – rather than *how* to actually perform it
- Few are focused on reuse and the networked nature of web ontologies

- One of the most cited:
  - Ontology development 101 – Noy & McGuinnes (2001)
    - Pre-OWL methodology
    - Traditional in the sense
      - It doesn't have a specific task focus
      - It is a waterfall like method
    - Although detailed in some steps, no details on requirements or testing etc.
    - Basic steps for modelling

      (1) Domain an scope (2) Consider reuse (3) Enumerate terms
      (4) Develop class hierarchy (5) Define the properties
      (6) Define restrictions and constraints (7)Create instances

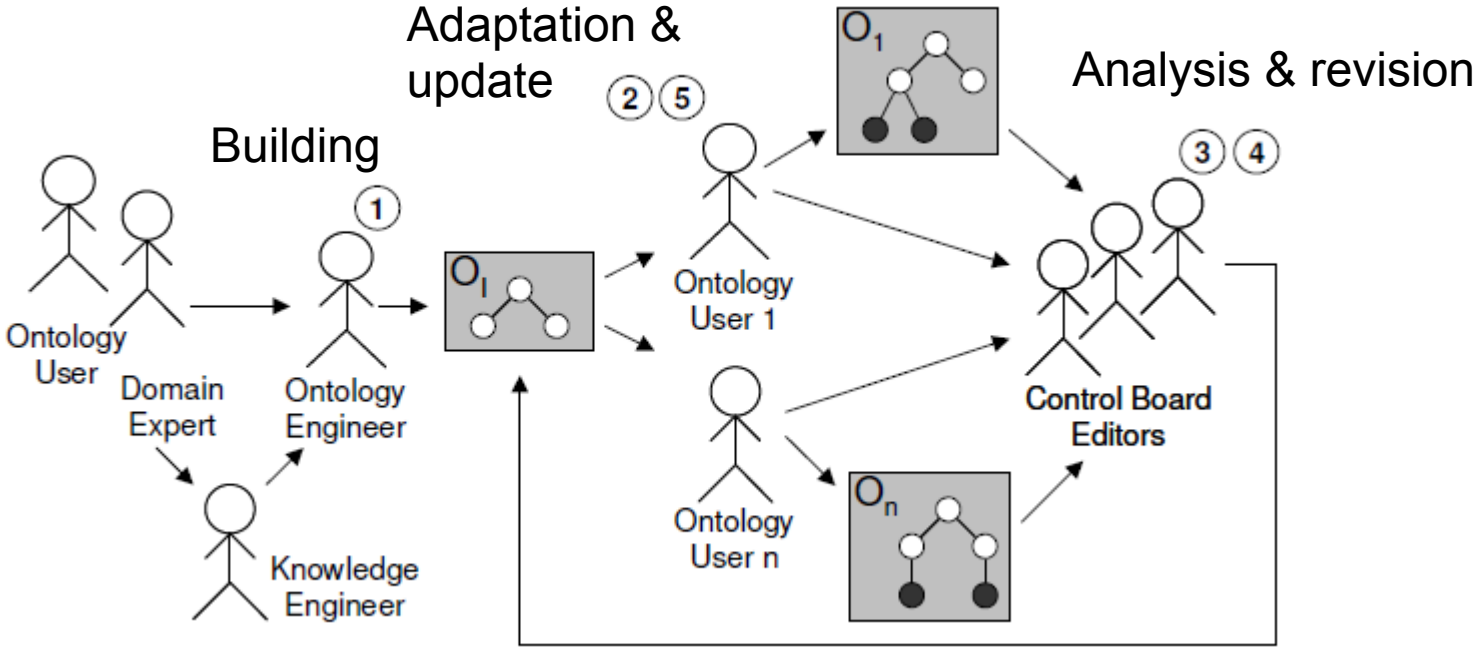


Example: METHONTOLOGY (~1997)

- Waterfall-like process consisting of (overlapping) phases
  1. Specification – document requirements, scope, level of formality etc.
  2. Knowledge Acquisition – gathering and studying sources of information
  3. Conceptualization – structure the terminology identified in 1, going from glossary to logical formulas
  4. Integration – find and select other ontologies to reuse
  5. Implementation – represent in formal language using tool
  6. Evaluation – verification and validation
  7. Documentation

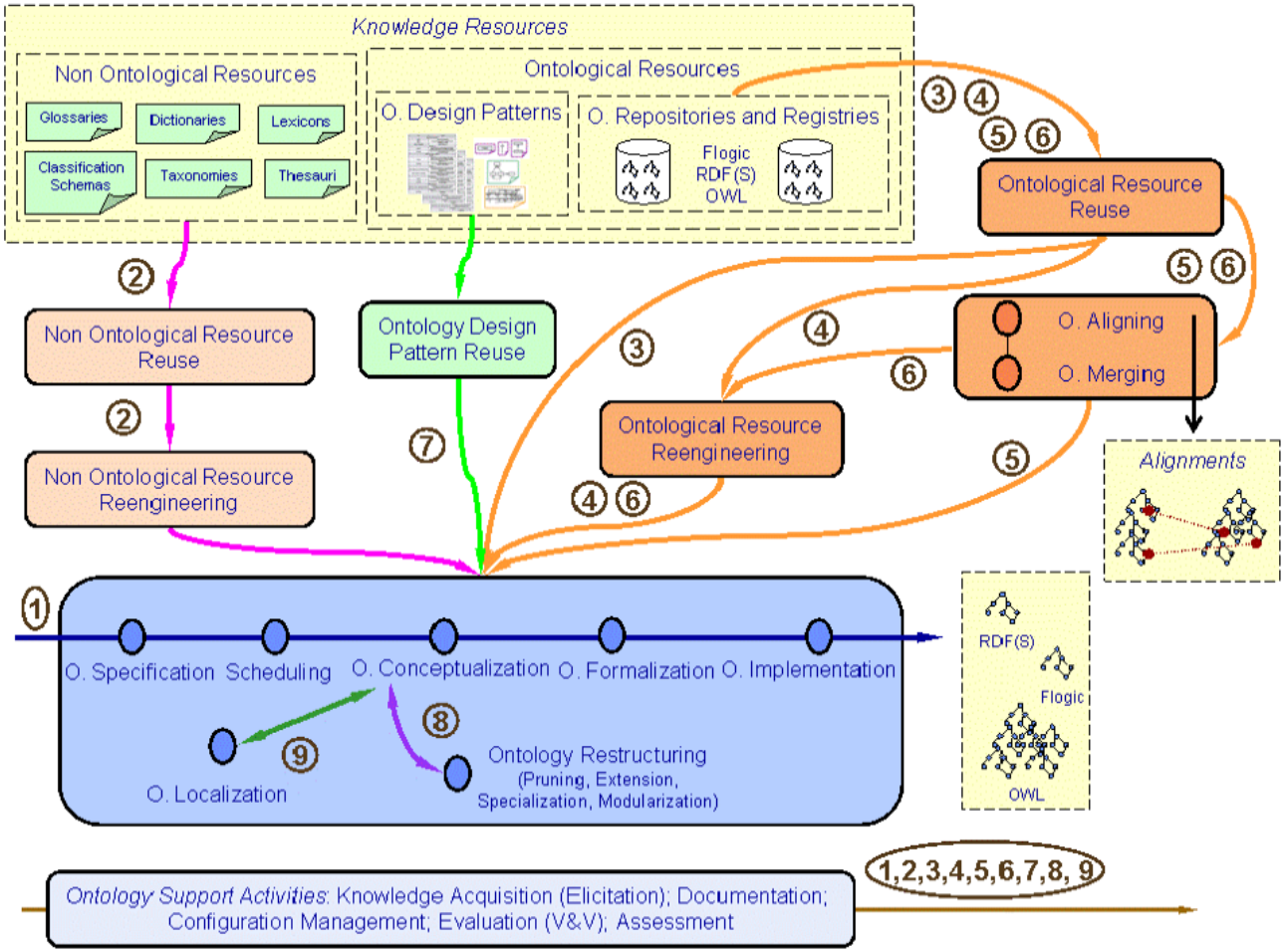
Example: DILIGENT (~2004)

- Based on theories for argumentation
- Intended for
  - Empowering domain experts in ontology engineering
  - Continous and distributed construction and update



The NeOn Methodology (2006-2010)

- Seven scenarios for ontology engineering



eXtreme Design

"Rapid Prototyping" based on ODPs

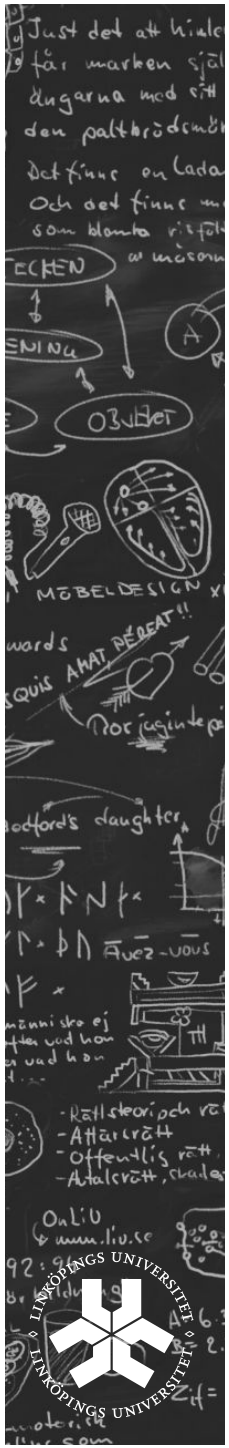
LiU expanding reality

Why the name “XD”?

- Inspired by XP ☺ with focus on design
- An agile methodology for web ontology design
- It is part of the NeOn methodology

Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

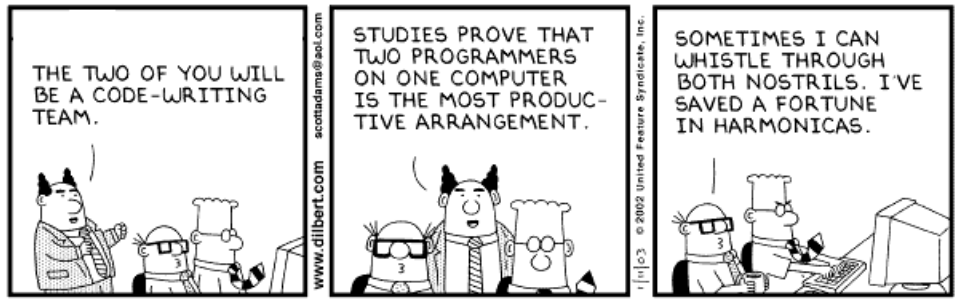
August 17, 2012



XD principles

- Customer involvement and feedback
- Customer stories to derive CQs (+ contextual statements, reasoning requirements)

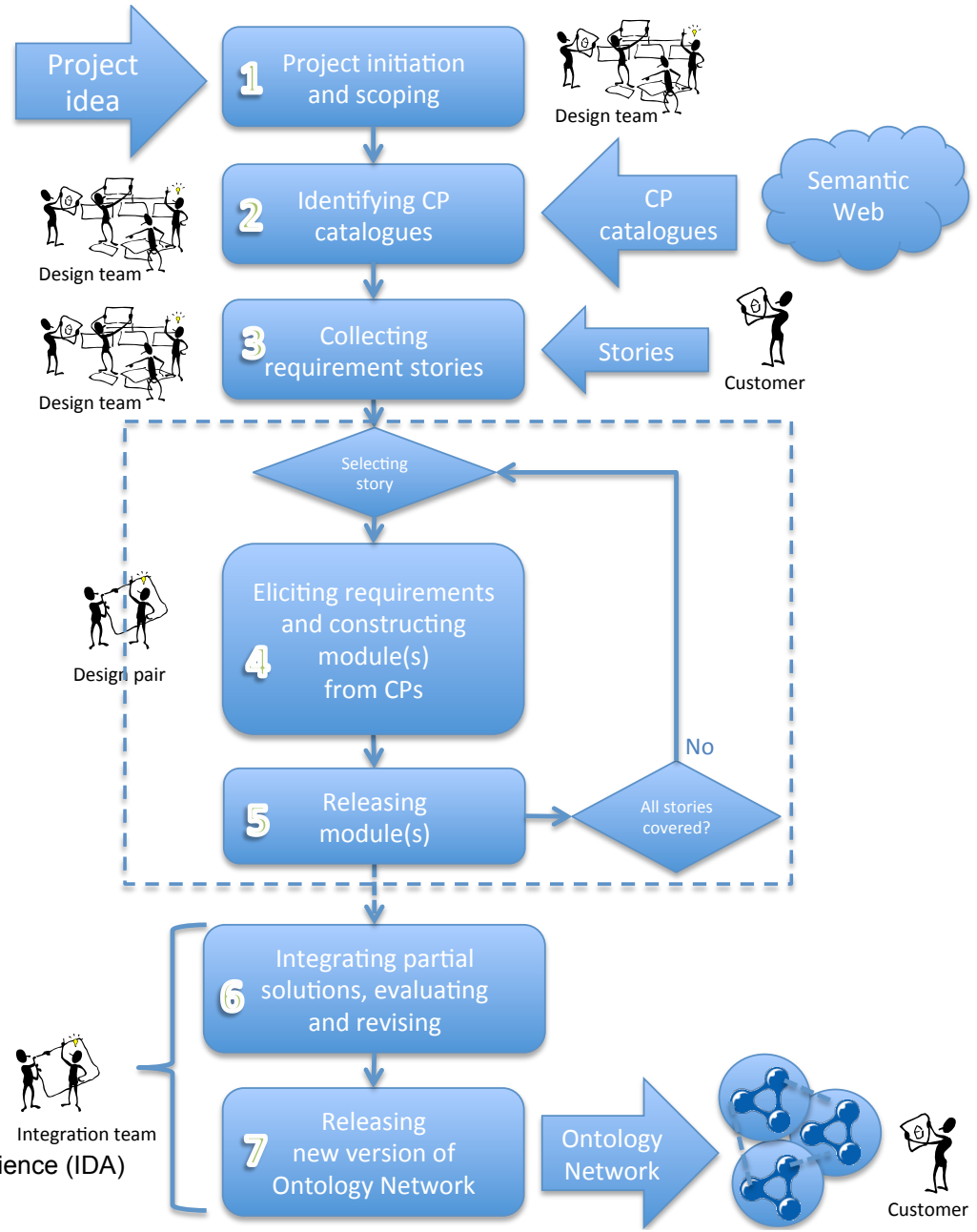
- CP reuse and modular design (ontology networks)
- Collaboration and integration
- Task-oriented design
- Test-driven design
- Pair design



Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

August 17, 2012

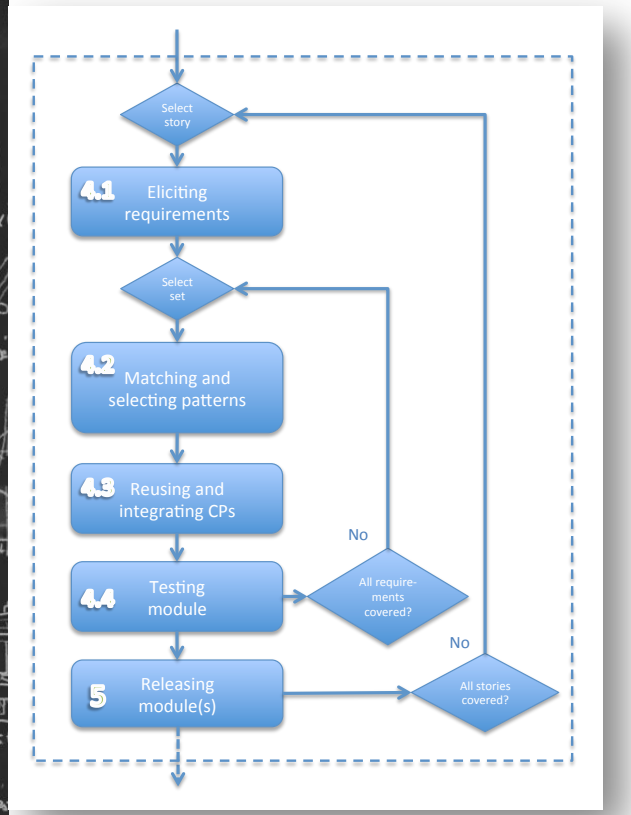
XD Iteration

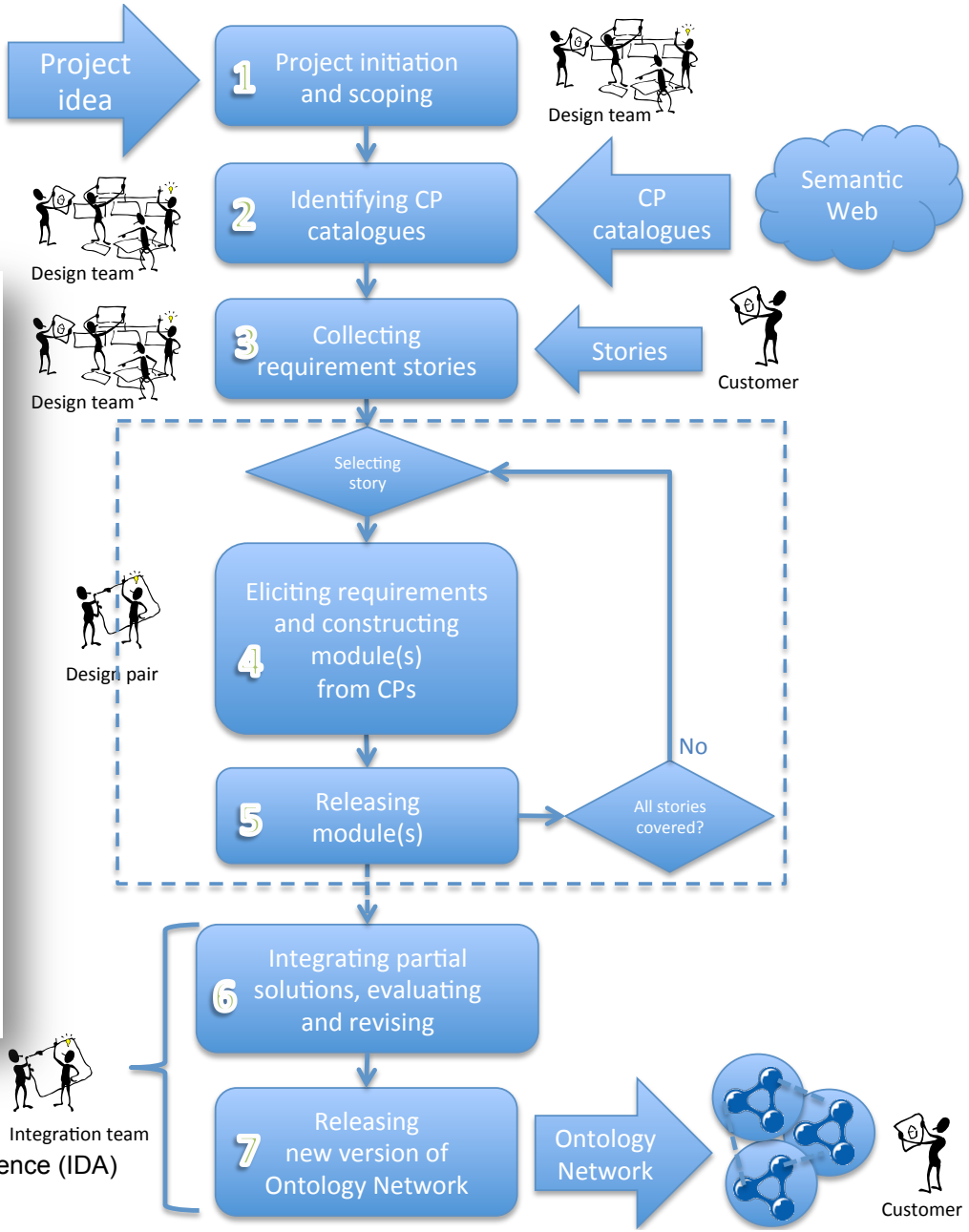


Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

August 17, 2012

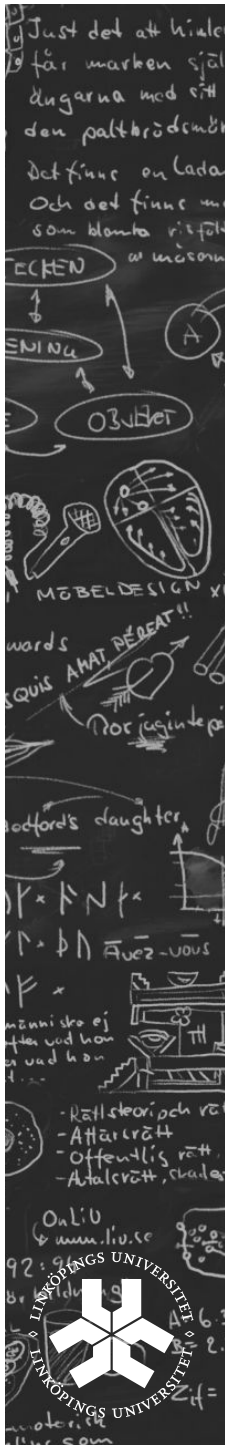
XD Iteration





Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden

August 17, 2012



XD Summary

- XD is an agile method – start building small modules that solve a few requirements, then add more, we don't decide on all the requirements at once
- Testing is essential – by figuring out the test you figure out how the model should work!
- Collaboration is essential
- Many problems are resolved in the integration phase – alignments or refactoring?
  - Need for good overall design policies

Department of Computer and Information Science (IDA)
Linköpings universitet, Sweden