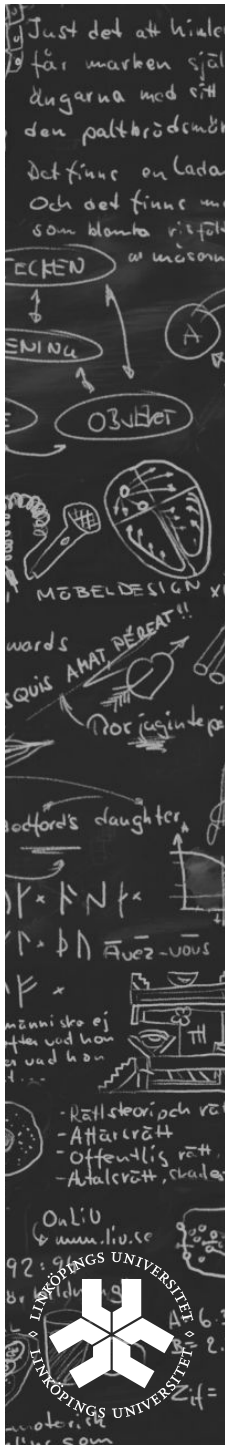
OWL Constructs - Reminder

All slides and exercises available at:

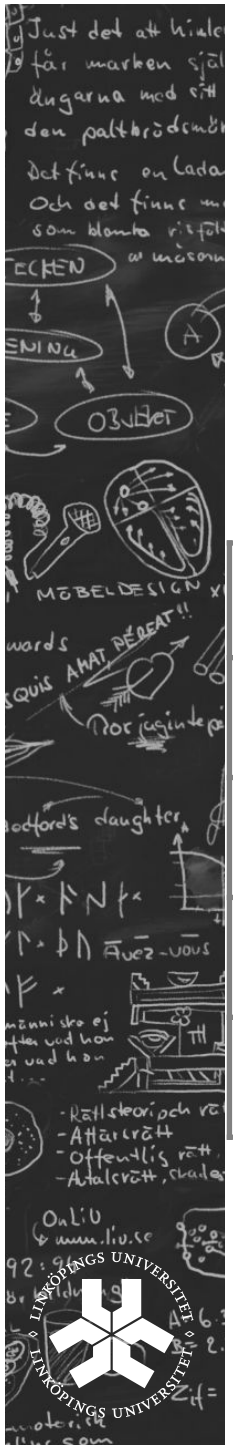
http://ontologydesignpatterns.org/wiki/Training:Advanced_Ontology_Engineering_at_FOI_-_2011

LiU expanding reality



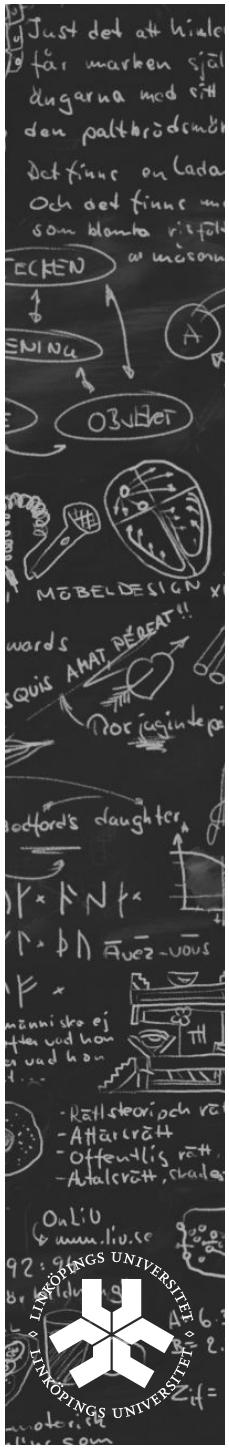
OWL Class Constructors

Constructor	Example	Interpretation
<Classes>	Human	*I(Human)*
intersectionOf	intersectionOf(Human Male)	*I(Human) ∩ I(Male)*
unionOf	unionOf(Doctor Lawyer)	*I(Doctor) ∪ I(Lawyer)*
complementOf	complementOf(Male)	*Δ \ I(Male)*
oneOf	oneOf(john mary)	*{I(john), I(mary)}*



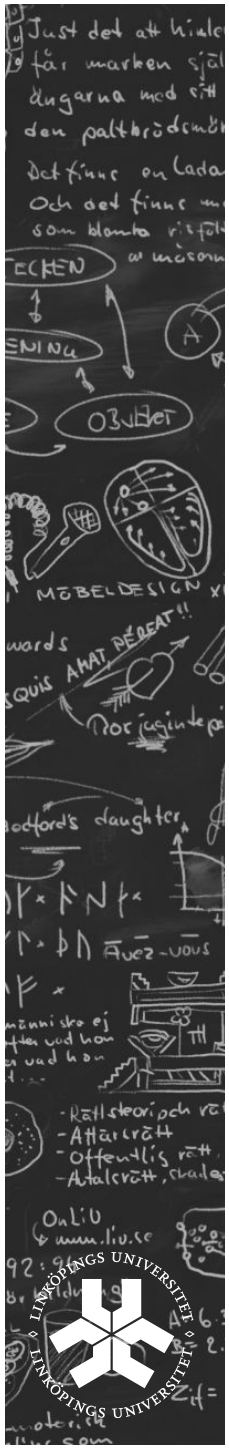
OWL Class Constructors

Constructor	Example	Interpretation
someValuesFrom	restriction(hasChild someValuesFrom Lawyer)	$\{x \mid \exists y.\ \langle x,y\rangle \in I(hasChild) \wedge y \in I(Lawyer)\}$
allValuesFrom	restriction(hasChild allValuesFrom Doctor)	$\{x \mid \forall y.\ \langle x,y\rangle \in I(hasChild) \Rightarrow y \in I(Doctor)\}$
minCardinality	restriction(hasChild minCardinality(2)))	$\{x \mid \#\langle x,y\rangle \in I(hasChild) \geq 2\}$
maxCardinality	restriction(hasChild maxCardinality (2))	$\{x \mid \#\langle x,y\rangle \in I(hasChild) \leq 2$



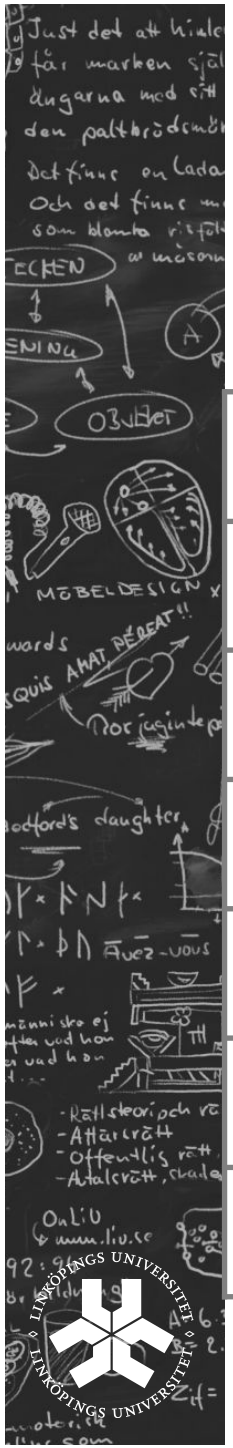
OWL Axioms

Axiom	Example	Interpretation
SubClassOf	SubClassOf(Human Animal)	$I(Human) \subseteq I(Animal)$
EquivalentClasses	EquivalentClass(Man intersectionOf(Human Male))	$I(Man) = I(Human) \cap I(Male)$
DisjointClasses	DisjointClass(Animal Plant)	$I(Animal) \cap I(Plant) = \varnothing$



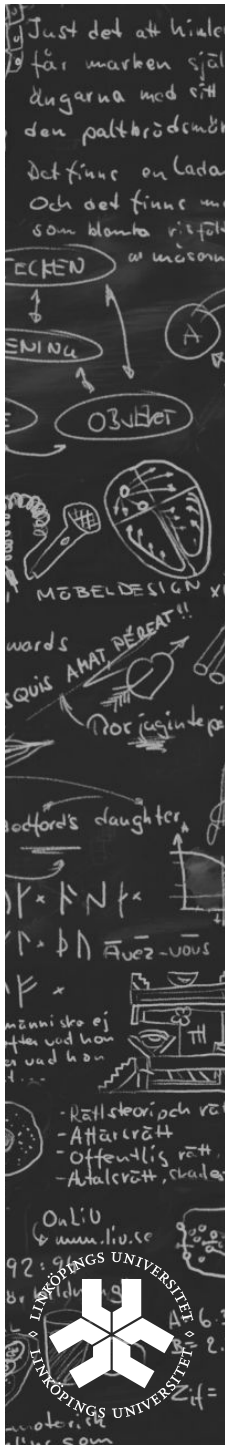
OWL Individuals

Axiom	Example	Interpretation
<Individual>	Individual(Valentina Type(Human))	*I(Valentina) ∈ I(Human)*
<Fact>	Individual(Valentina value(worksWith Aldo))	*I⟨Valentina,Aldo⟩∈ I(worksWith)*
DifferentIndividuals	DifferentIndividuals(Valentina Aldo)	*I(Valentina) ≠ I(Aldo)*
SameIndividualAs	SameIndividualAs(AldoGangemi GangemiAsTutor)	*I(*AldoGangemi*) = I(*GangemiAsTutor*)*



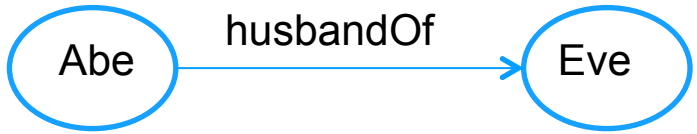
OWL Property Axioms

Axiom	Example	Interpretation
SubPropertyOf	SubPropertyOf(hasMother hasParent)	$I(hasMother) \subseteq I(hasParent)$
domain	ObjectProperty(owns domain(Person))	$\forall x.\langle x,y\rangle \in I(owns) \Rightarrow x \in I(Person)$
range	ObjectProperty(employs range(Person))	$\forall x.\langle x,y\rangle \in I(employs) \Rightarrow y \in I(Person)$
symmetric	ObjectProperty(connects Symmetric)	$\forall x,y.\langle x,y\rangle \in I(connects) \Rightarrow \langle y,x\rangle \in I(connects)$
transitive	ObjectProperty(hasPart Transitive)	$\forall x,y,z.\langle x,y\rangle \in I(hasPart) \wedge \langle y,z\rangle \in I(hasPart) \Rightarrow \langle x,z\rangle \in I(hasPart)$
inverseOf	ObjectProperty(hasChild inverseOf(hasParent))	$I(hasChild) = I(hasParent^{-})$

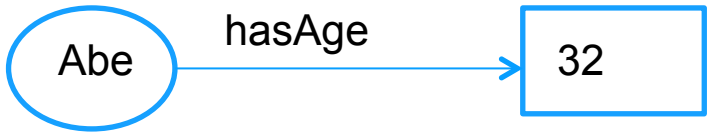


Types of properties

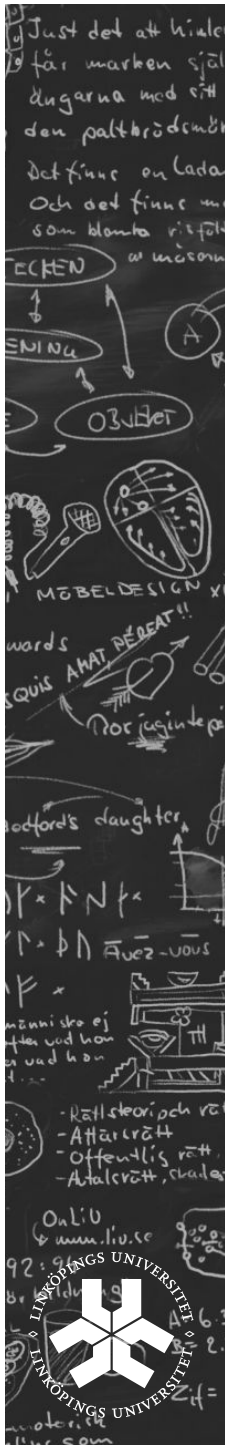
- Yesterday: owl:ObjectProperty
  - A property for creating triples where both subject and object is an instance (individual of some class)



- Datatype properties (owl:DatatypeProperty)
  - A property that holds between an instance and a literal, e.g. number, string, date… (we can use xsd-datatypes)



- Annotation properties – for documentation



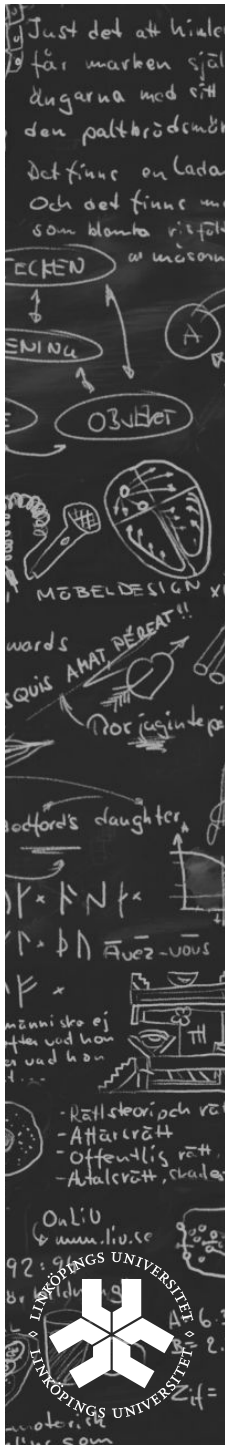
Additional useful things

- Everything is identified by URIs (the “web” part of OWL)
  - The ontology has a URI which is the namespace of its content
  - Each element in the ontology has a URI, based on the URI of the ontology
  - Example: ontology URI – http://mydomain.se/ontology1 person class in my ontology - http://mydomain.se/ontology1/person or http://mydomain.se/ontology1#person
  - Namespace prefixes, short names for URIs, e.g. “rdf:” actually means “http://www.w3.org/1999/02/22-rdf-syntax-ns#”
- owl:imports
  - A imports B means that all statements in B are included in A
- Annotation properties
  - rdfs:label
  - rdfs:comment

Requirements Engineering

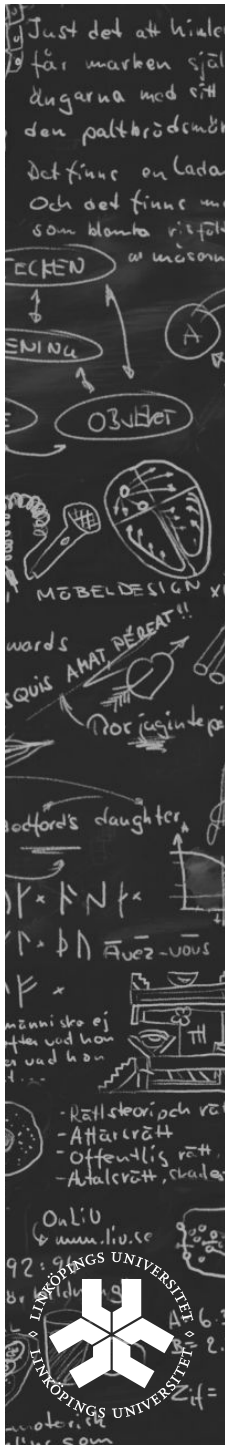
Eva Blomqvist
eva.blomqvist@liu.se
Department of Computer and Information Science (IDA)
Linköpings universitet
Sweden

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.

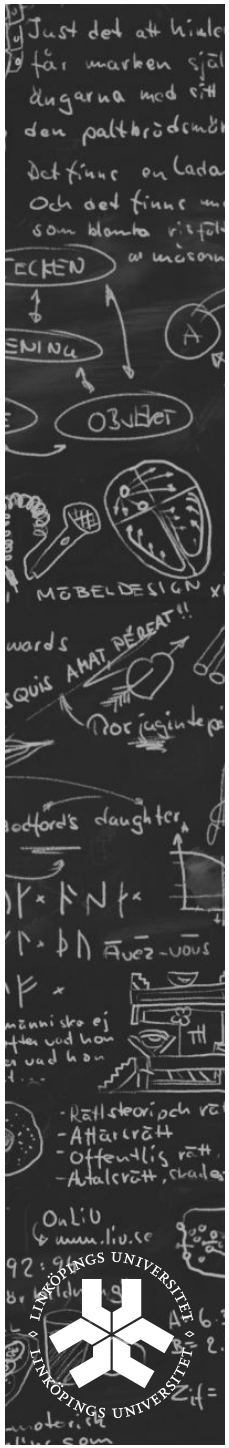


What are “requirements”?

- 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

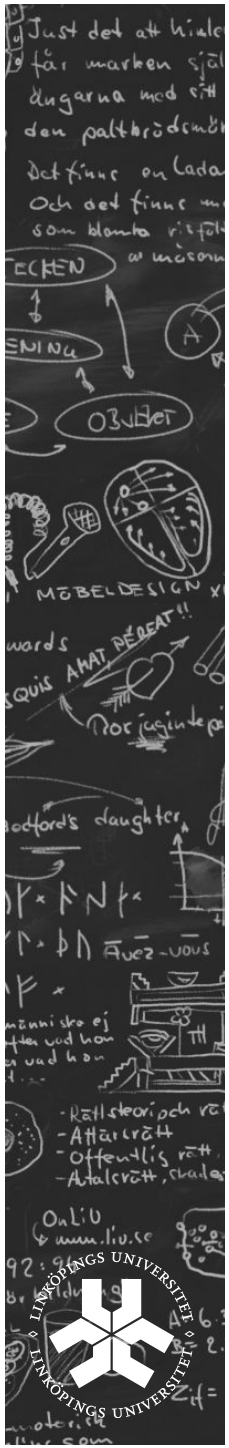
Overall structure, acceptance ➔ Guidelines and rules for development



Requirements Engineering – Competency Questions

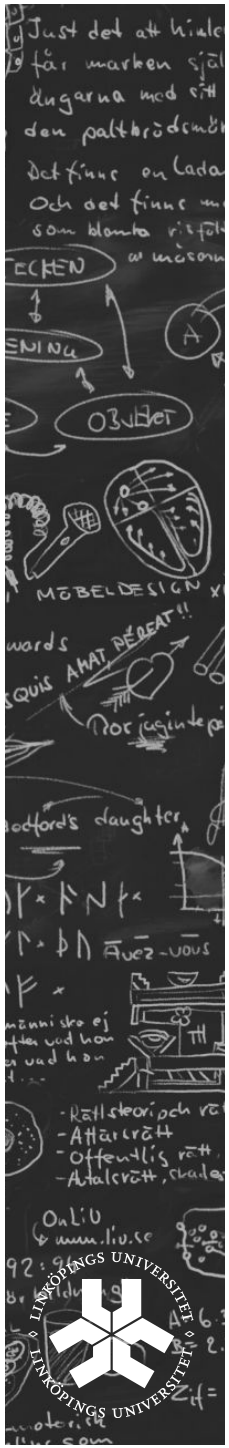
- Competency Questions (CQ) = Natural language questions that ask for information the ontology should be able to provide to a user (or system)
  - Functional requirements
  - Related to software requirements – “input” and “output” of the “ontology component” (including query engine, reasoner…)

- 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.)

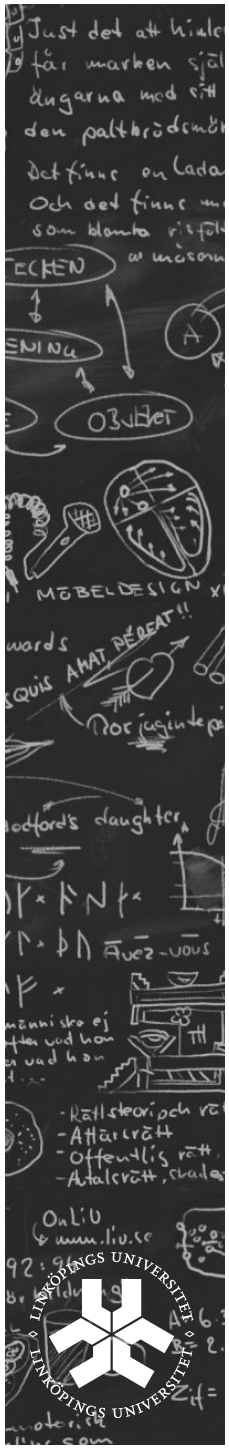
- To clarify complex CQs we use
  - Contextual statements
  - Inference (reasoning) requirements
- A contextual statement expresses an axiom that needs to hold in the ontology, in natural language
  - Every course has at least one participant
  - A grandparent is someone who has a child who in turn also 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 – Competency Questions (cont.)

- Summary: Three types of functional requirements
  - CQs
    - Give me all the grandparents of the KB.
  - Contextual statements
    - A grandparent is someone who has a child who also has a child
  - Reasoning requirements
    - Being a grandparent is derived from the relation to children

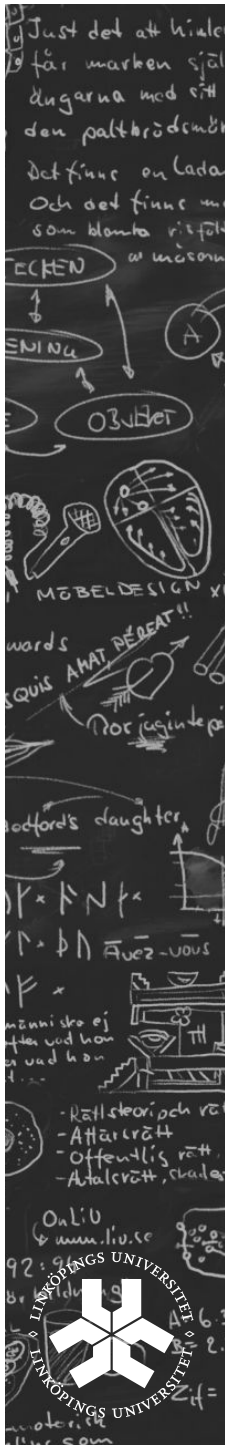
- We need to collect such requirements so that they cover the complete domain of the ontology!



Requirements Engineering - Deriving CQs

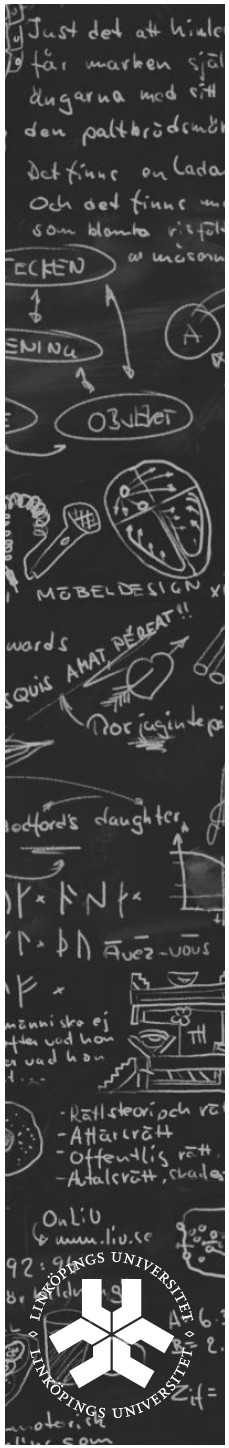
- How do we get the CQs?
  - Domain experts are good at telling stories and giving examples - CQs can be derived from such examples
    - Coverage
    - End-user tasks
  - Software requirements
    - Tasks

- Customer says: “All companies have a name and a tax registration number”
  - Requirements
    - Contextual statement!
    - Possible CQ: Retrieve a company with a certain name or tax registration number?



Requirements Engineering - Deriving CQs (cont.)

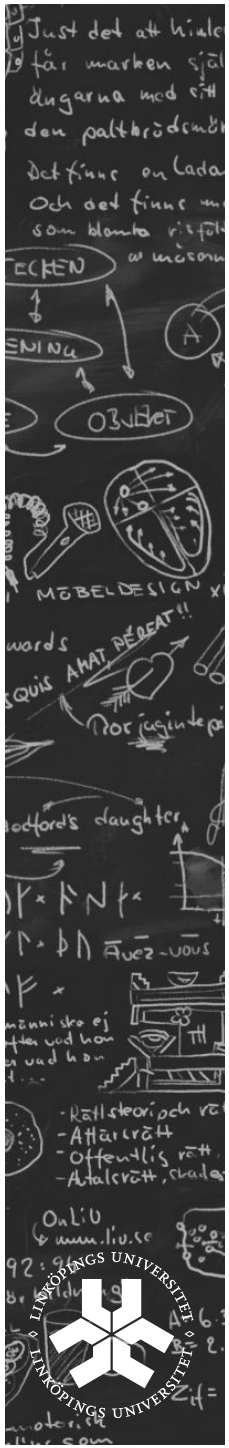
- Text analysis - possible procedure
  - Collect stories from the customer/domain expert
  - Divide into pieces (one or a couple of sentences) dealing with *one* aspect or topic
  - Create instance-free sentences
  - What CQs can be derived from the instance-free sentences? “Complex” CQs?
  - What possible contextual statements could apply?
  - What reasoning tasks do we need to support?
  - Check & revise
    - With customer
    - Against software requirements



Requirements Engineering - Deriving CQs (cont.)

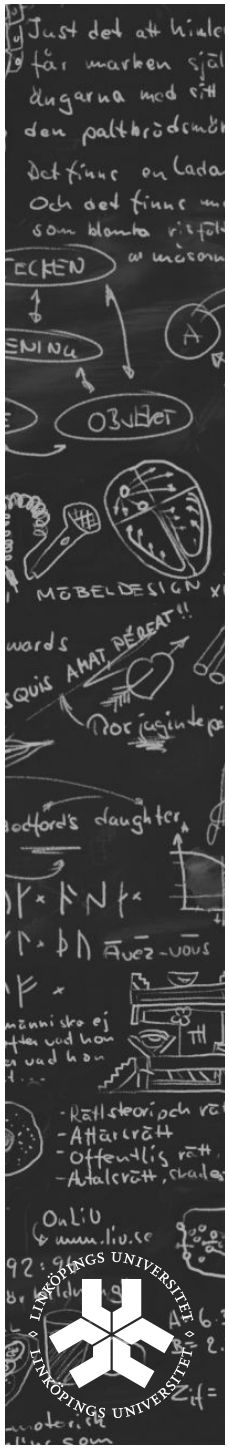
- What is an “instance-free” sentence?
  - What do you expect to become the facts, “data”, in the knowledge base?
  - Abstraction!

- Example
  - Eva is teaching a course at FOI.
    - A person is teaching a course in an organization.
      or
      A person is participating in an event at an organization with a certain role.
    - How to decide? – Think about what is the data, what we enter at runtime and query for! Do we need to query for the type of event? For the role of a certain person in an event?



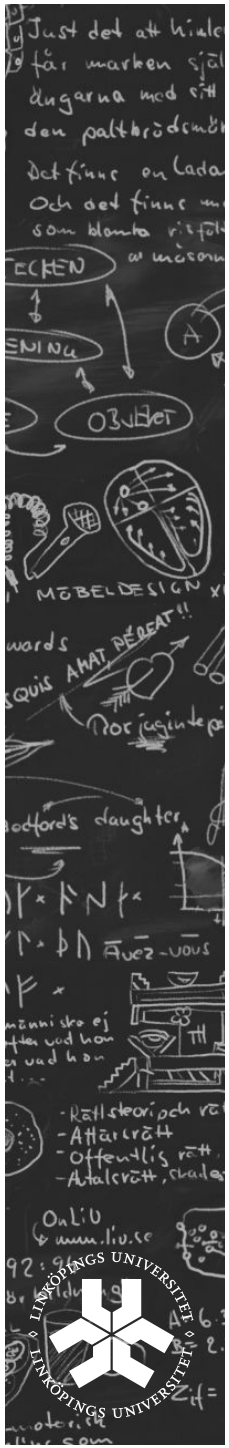
Requirements Engineering - Deriving CQs (cont.)

- Derive CQs
  - What are the things that domain experts, i.e. end-users, would expect to be able to do or see in the interface based on the KB?
  - What does the overall software require from the ontology?
- Examples
  - A person is participating in an event at an organization with a certain role.
    - What are the roles of the people involved in a certain event?
    - What organization is hosting the event?
  - A person lives in a city.
    - Where does this person live? Who lives in this city?



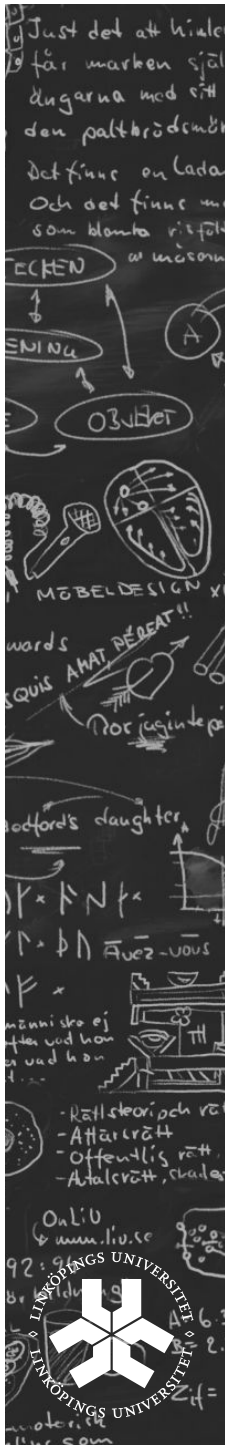
Requirements Engineering – Non-functional Requirements

- 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?



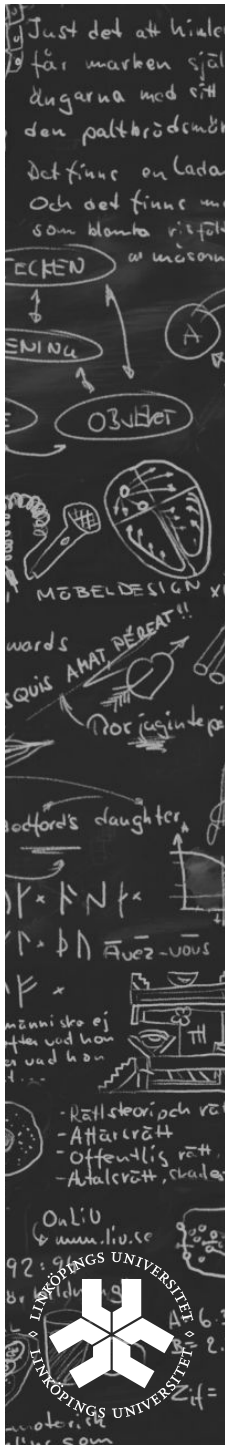
Trade-off: Software vs. Ontology

- 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



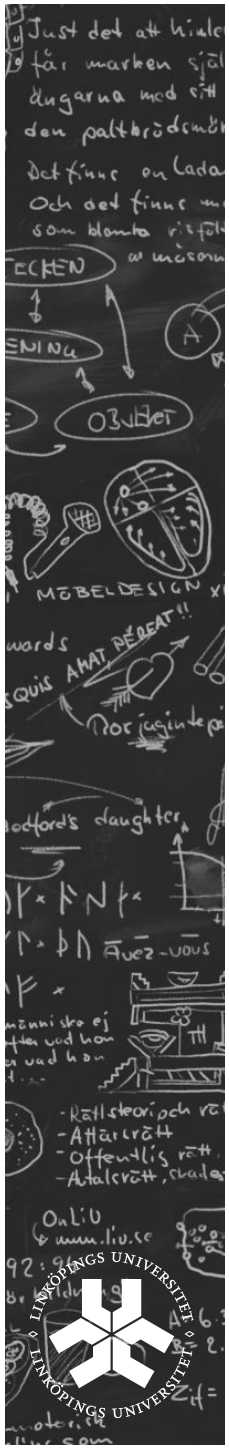
Exercise - Deriving possible CQs and other requirements from text

- Context:
  - An online music database wishes to semantically represent their data about musicians, albums, and performances, in order to be able to provide better search functions to their users, i.e. by querying the knowledge base instead of using keyword queries.
- User interface:
  - Search for songs and who has recorded/played a song
  - Search for artists and bands and get information about them
  - Visualize timeline of a band
  - Browse genres of songs, band and artists



Exercise - Deriving possible CQs and other requirements from text

- Take the following text and try to derive requirements from it (CQs + contextual statements & reasoning requirements)
  - The current configuration of the “Red Hot Chili Peppers” are: Anthony Kiedis (vocals), Flea (bass, trumpet, keyboards, and vocals), John Frusciante (guitar), and Chad Smith (drums). The line-up has changed a few times during they years, Frusciante replaced Hillel Slovak in 1988, and when Jack Irons left the band he was briefly replaced by D.H. Peligo until the band found Chad Smith. In addition to playing guitars for Red hot Chili Peppers Frusciante also contributed to the band “The Mars Volta” as a vocalist for some time. Red Hot Chili Peppers released the album “Stadium Arcadium” in 2006. It includes the song “Hump de Bump”, which was composed in January 2004. “Hump de Bump” is a punk rock song, but the band is also playing alternative rock, and hip hop on some records. The critic Crian Hiatt defined the album as "the most ambitious work in his twenty-three-year career".



Requirements Engineering - Method Reminder

- Text analysis - possible procedure
  - Collect stories from the customer/domain expert
  - Divide into pieces (one or a couple of sentences) dealing with *one* aspect or topic
  - Create instance-free sentences
  - What CQs can be derived from the instance-free sentences? “Complex” CQs?
  - What possible contextual statements could apply?
  - What reasoning tasks do we need to support?
  - Check & revise
    - With customer
    - Against software requirements