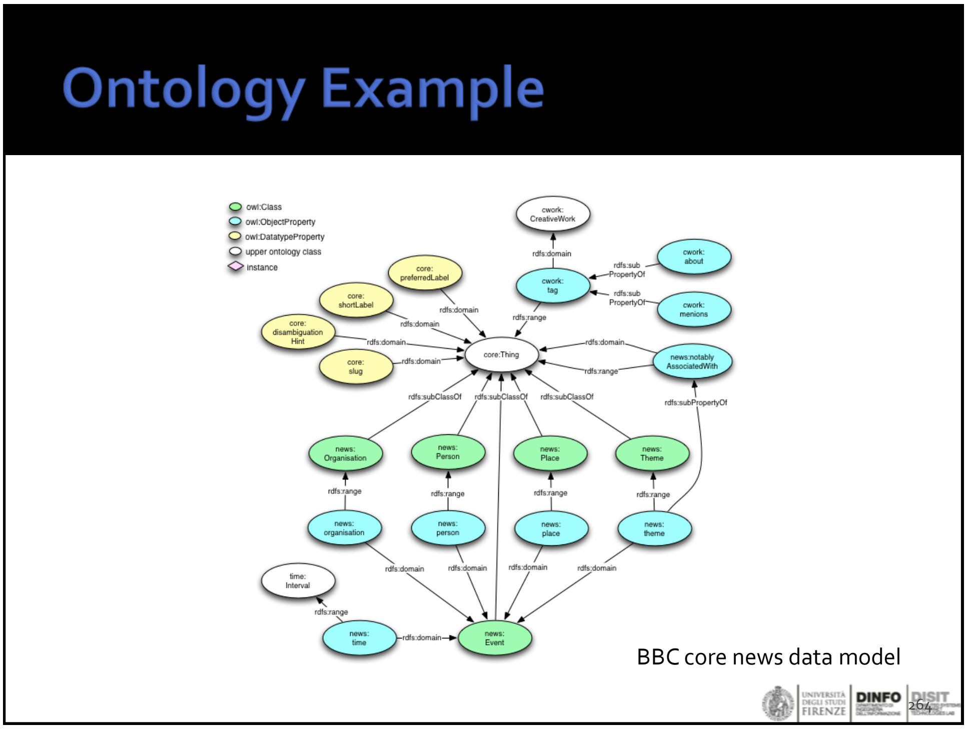
Ontology Engineering

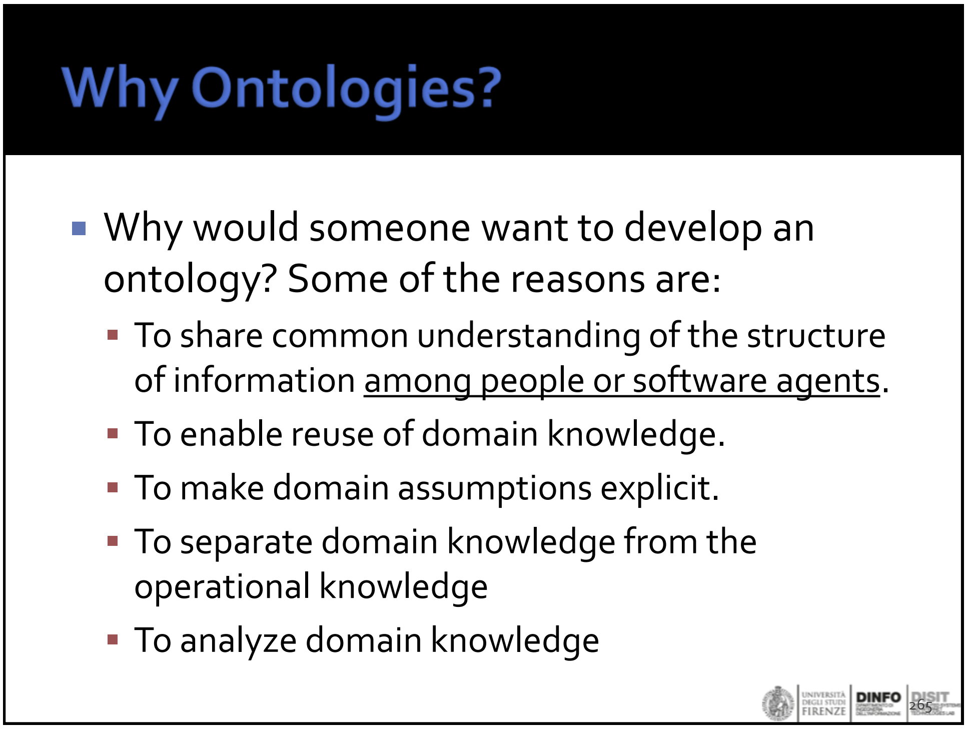
Ontology

- In practical terms, developing an ontology includes:
  - defining classes in the ontology,
  - arranging the classes in a taxonomic (subclass–superclass) hierarchy,
  - defining properties and describing allowed values for these properties,
  - filling in the values for properties for instances.



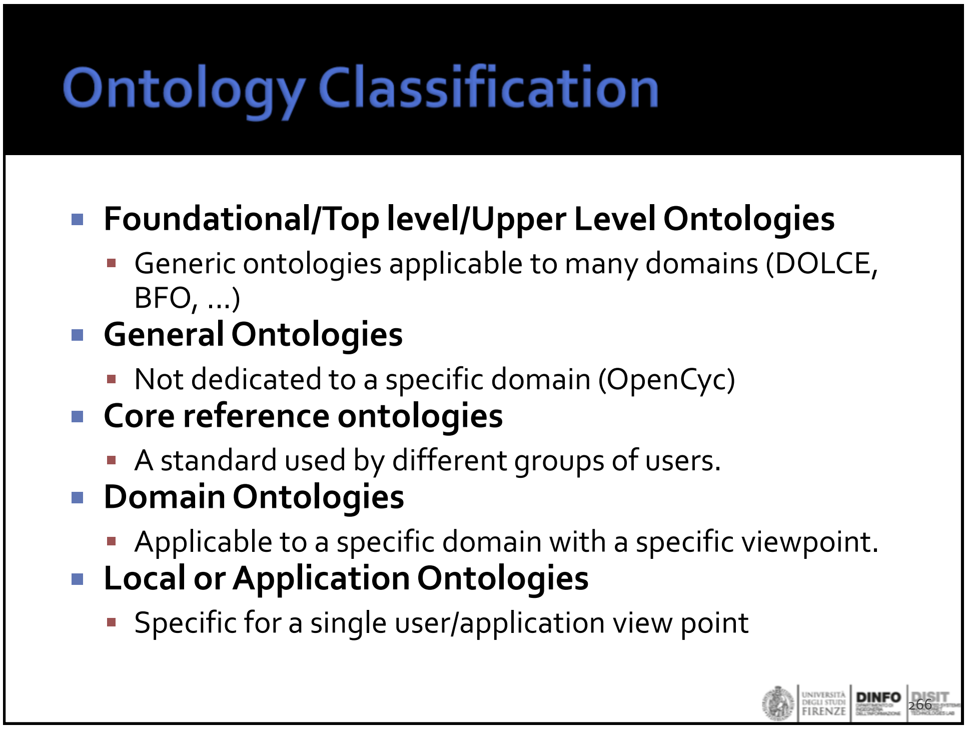
Ontology Example

BBC core news data model



Why Ontologies?

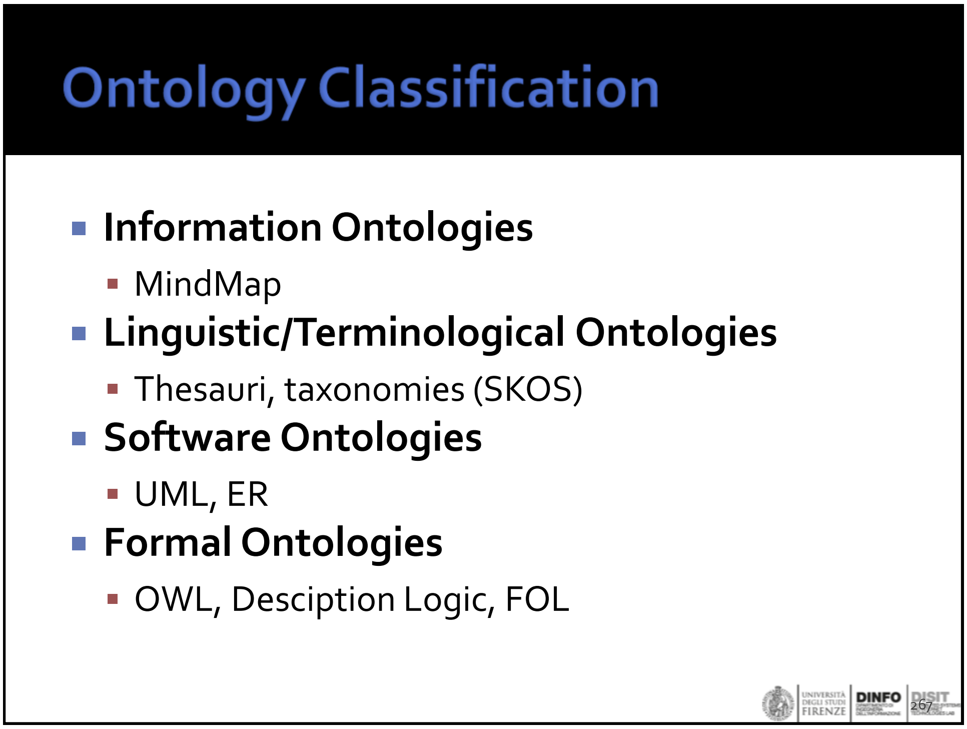
- Why would someone want to develop an ontology? Some of the reasons are:
  - To share common understanding of the structure of information among people or software agents.
  - To enable reuse of domain knowledge.
  - To make domain assumptions explicit.
  - To separate domain knowledge from the operational knowledge
  - To analyze domain knowledge



Ontology Classification

- **Foundational/Top level/Upper Level Ontologies**
  - Generic ontologies applicable to many domains (DOLCE, BFO, …)
- **General Ontologies**
  - Not dedicated to a specific domain (OpenCyc)
- **Core reference ontologies**
  - A standard used by different groups of users.
- **Domain Ontologies**
  - Applicable to a specific domain with a specific viewpoint.
- **Local or Application Ontologies**
  - Specific for a single user/application view point

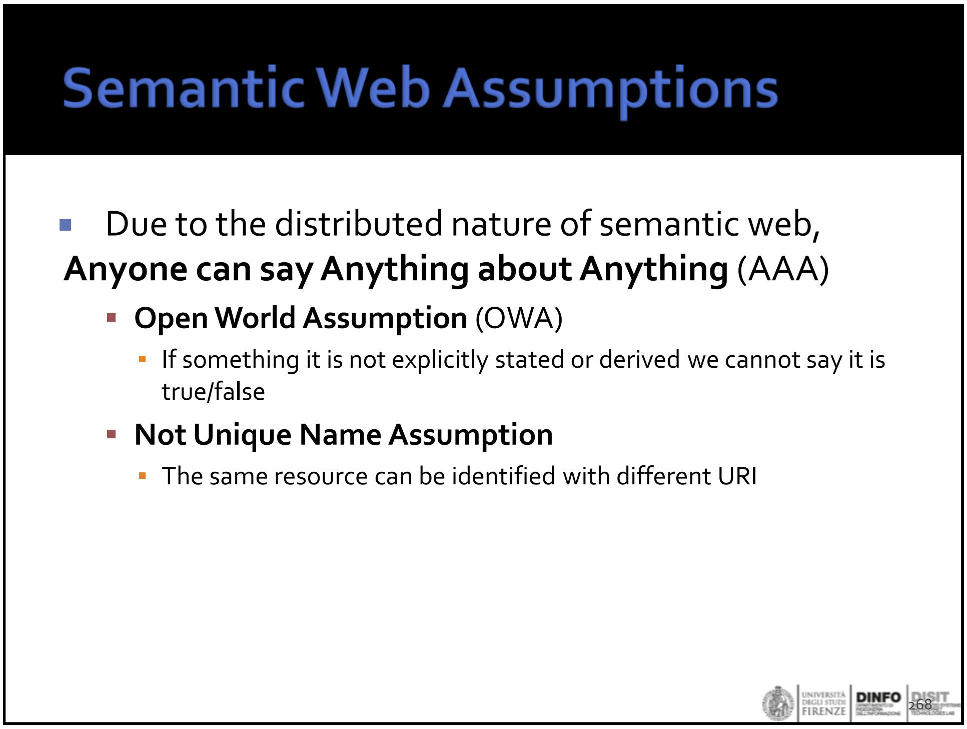
266



Ontology Classification

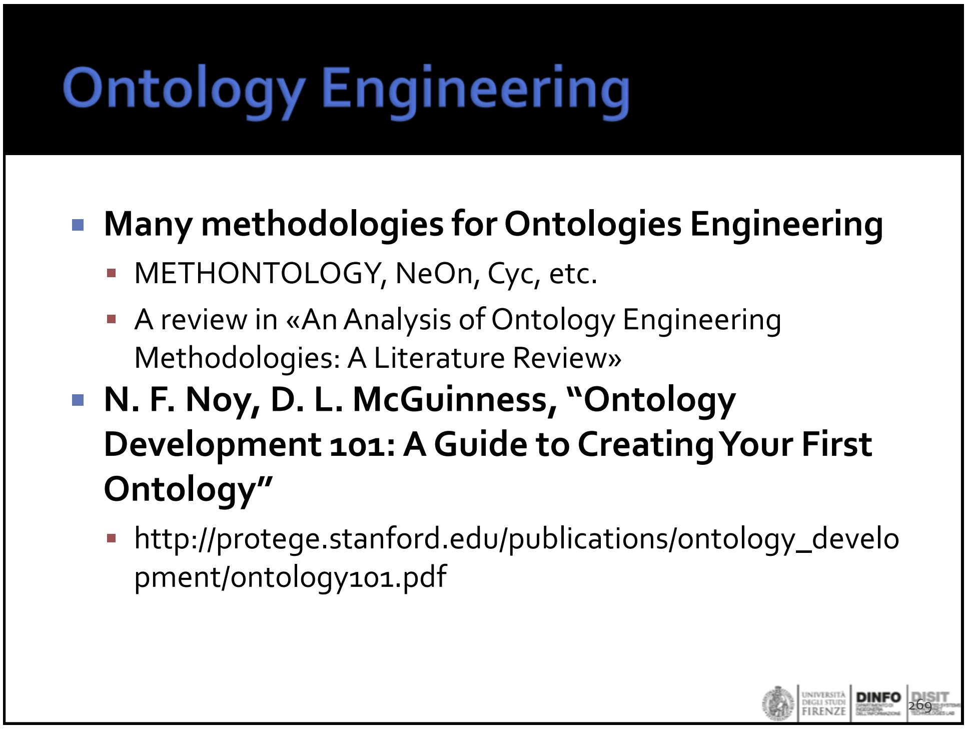
- **Information Ontologies**
  - MindMap
- **Linguistic/Terminological Ontologies**
  - Thesauri, taxonomies (SKOS)
- **Software Ontologies**
  - UML, ER
- **Formal Ontologies**
  - OWL, Desciption Logic, FOL

267



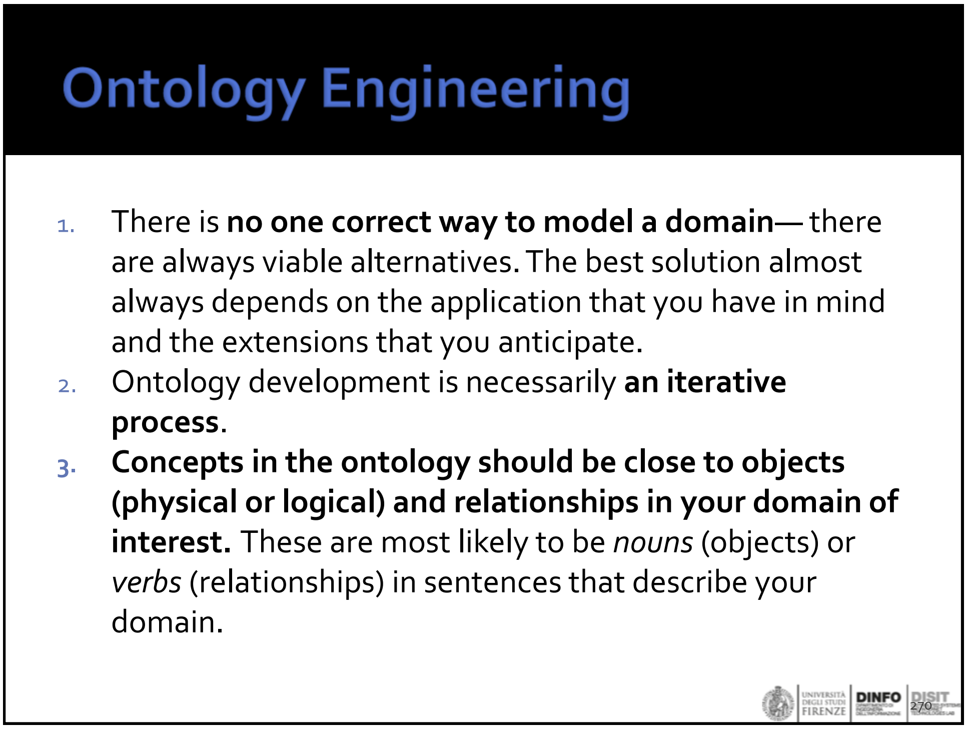
Semantic Web Assumptions

- Due to the distributed nature of semantic web, **Anyone can say Anything about Anything** (AAA)
  - **Open World Assumption** (OWA)
    - If something it is not explicitly stated or derived we cannot say it is true/false
  - **Not Unique Name Assumption**
    - The same resource can be identified with different URI



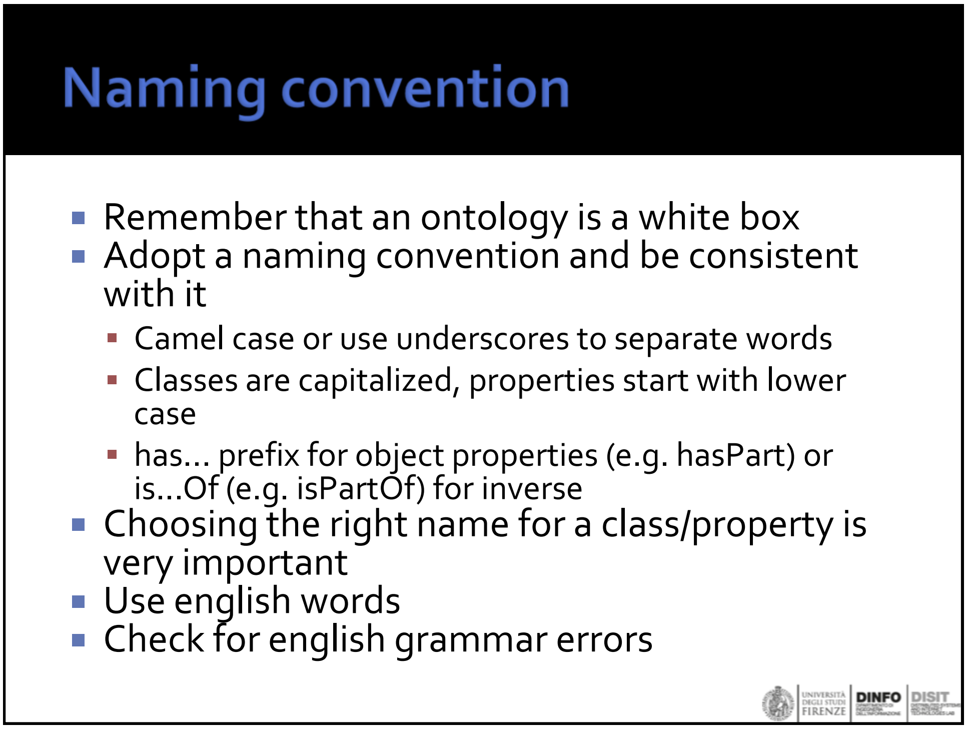
Ontology Engineering

- **Many methodologies for Ontologies Engineering**
  - METHONTOLOGY, NeOn, Cyc, etc.
  - A review in «An Analysis of Ontology Engineering Methodologies: A Literature Review»
- **N. F. Noy, D. L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology"**
  - http://protege.stanford.edu/publications/ontology_development/ontology101.pdf



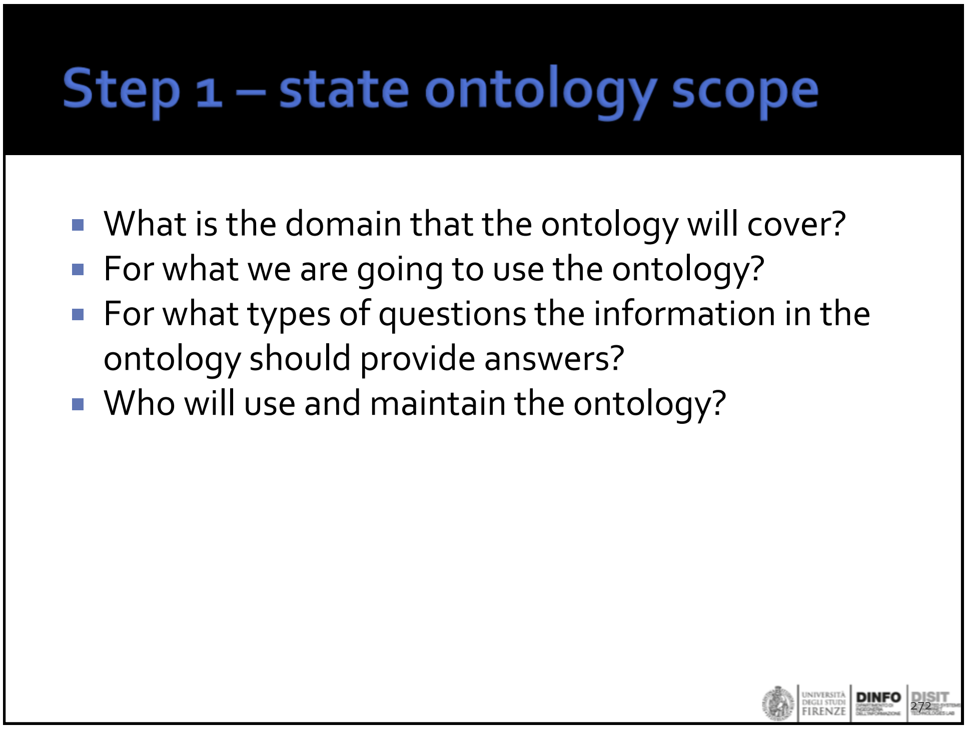
Ontology Engineering

1. There is **no one correct way to model a domain**— there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate.
2. Ontology development is necessarily **an iterative process**.
3. **Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest.** These are most likely to be *nouns* (objects) or *verbs* (relationships) in sentences that describe your domain.



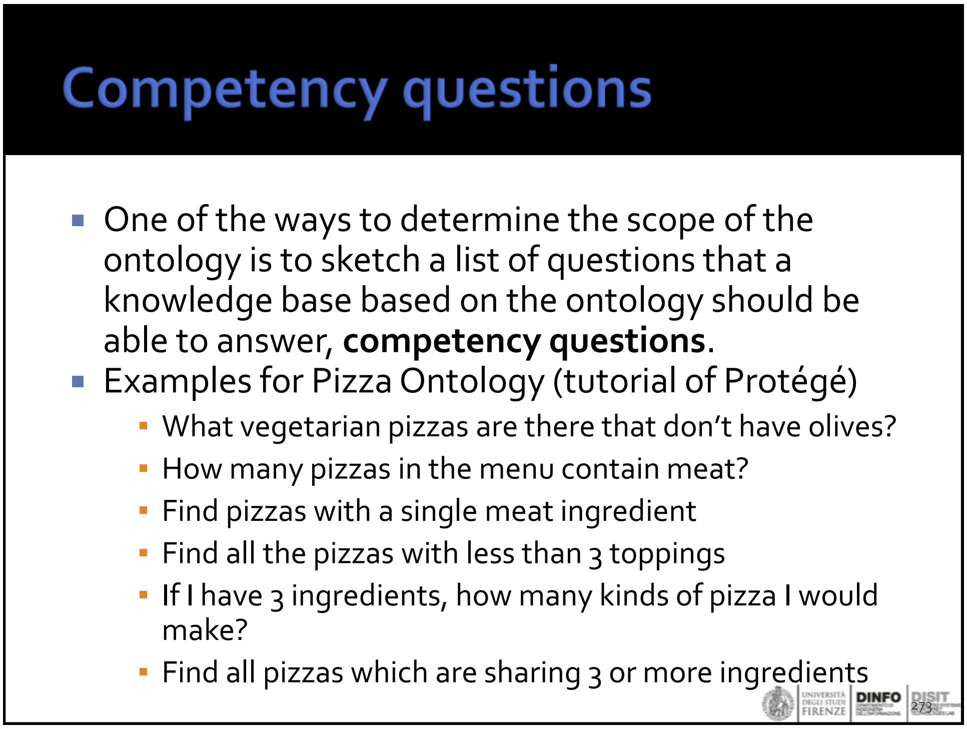
Naming convention

- Remember that an ontology is a white box
- Adopt a naming convention and be consistent with it
  - Camel case or use underscores to separate words
  - Classes are capitalized, properties start with lower case
  - has… prefix for object properties (e.g. hasPart) or is…Of (e.g. isPartOf) for inverse
- Choosing the right name for a class/property is very important
- Use english words
- Check for english grammar errors



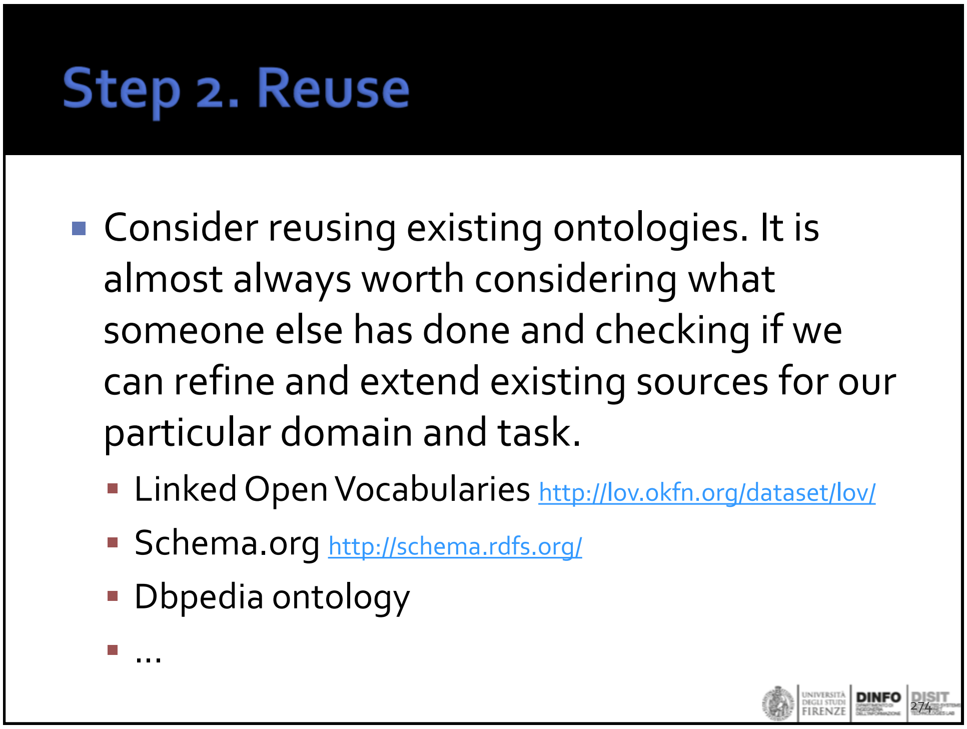
Step 1 – state ontology scope

- What is the domain that the ontology will cover?
- For what we are going to use the ontology?
- For what types of questions the information in the ontology should provide answers?
- Who will use and maintain the ontology?



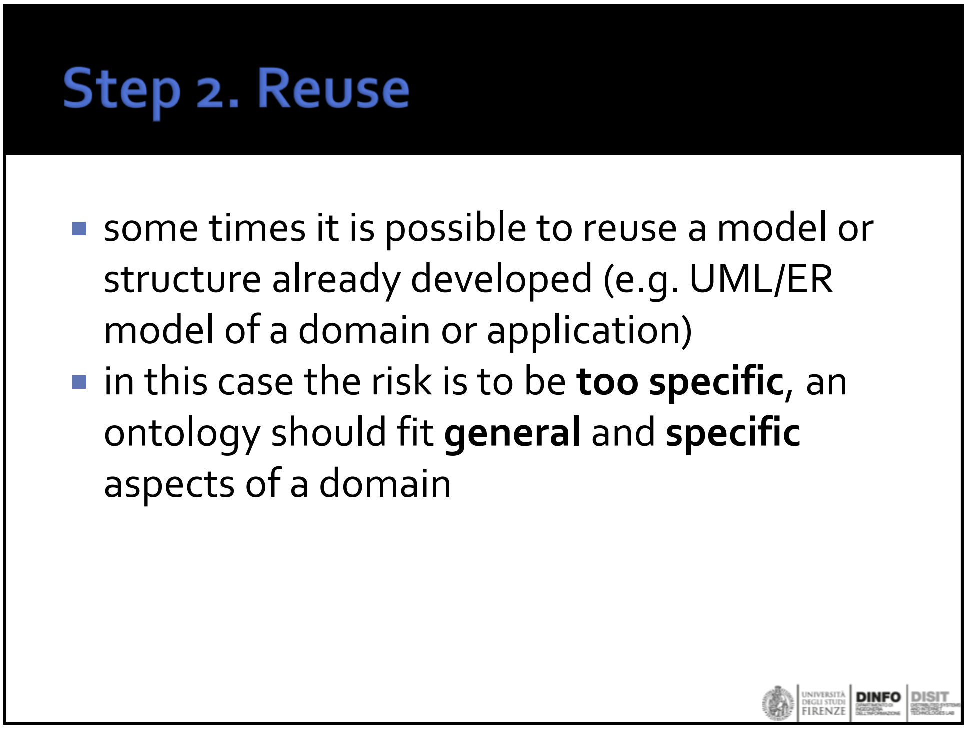
Competency questions

- One of the ways to determine the scope of the ontology is to sketch a list of questions that a knowledge base based on the ontology should be able to answer, **competency questions**.
- Examples for Pizza Ontology (tutorial of Protégé)
  - What vegetarian pizzas are there that don’t have olives?
  - How many pizzas in the menu contain meat?
  - Find pizzas with a single meat ingredient
  - Find all the pizzas with less than 3 toppings
  - If I have 3 ingredients, how many kinds of pizza I would make?
  - Find all pizzas which are sharing 3 or more ingredients



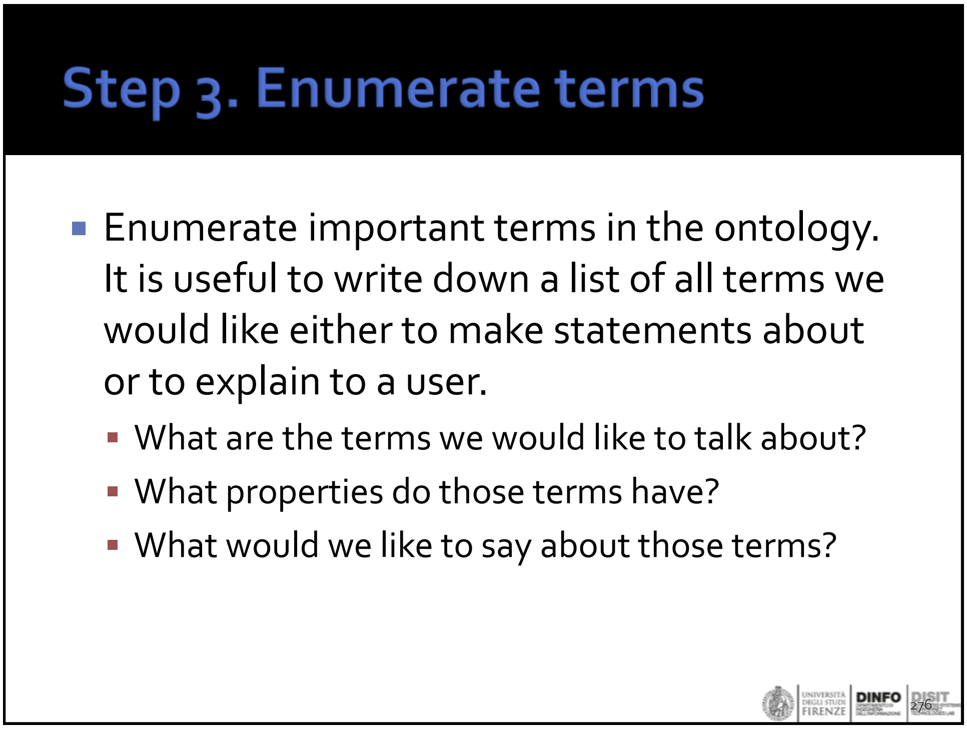
Step 2. Reuse

- Consider reusing existing ontologies. It is almost always worth considering what someone else has done and checking if we can refine and extend existing sources for our particular domain and task.
  - Linked Open Vocabularies http://lov.okfn.org/dataset/lov/
  - Schema.org http://schema.rdfs.org/
  - Dbpedia ontology
  - …



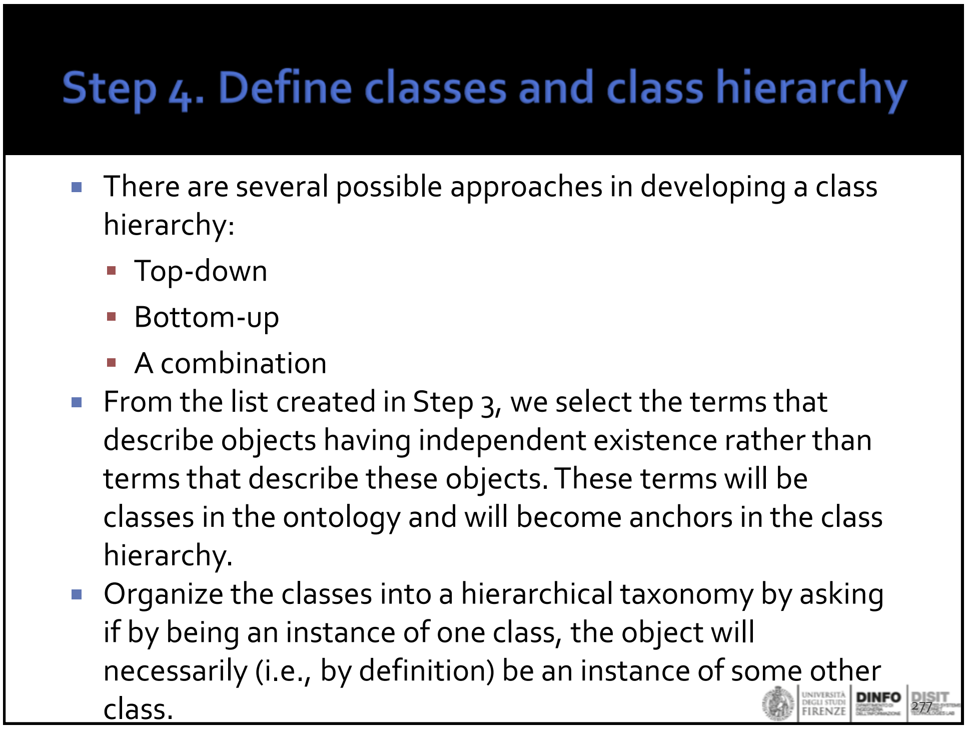
Step 2. Reuse

- some times it is possible to reuse a model or structure already developed (e.g. UML/ER model of a domain or application)
- in this case the risk is to be **too specific**, an ontology should fit **general** and **specific** aspects of a domain



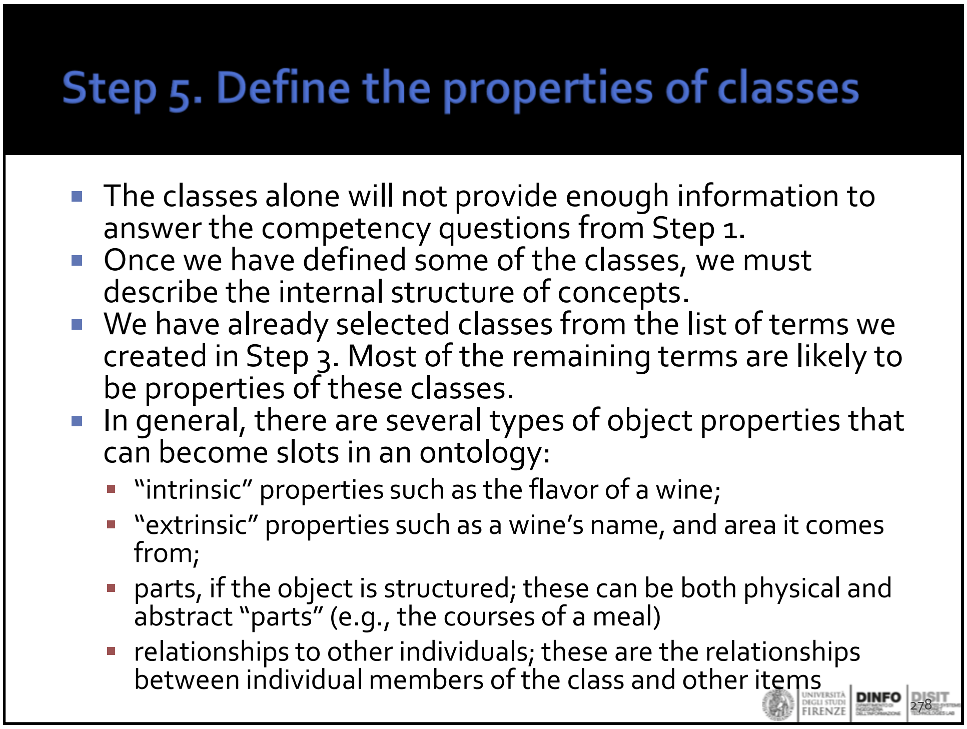
Step 3. Enumerate terms

- Enumerate important terms in the ontology. It is useful to write down a list of all terms we would like either to make statements about or to explain to a user.
  - What are the terms we would like to talk about?
  - What properties do those terms have?
  - What would we like to say about those terms?



Step 4. Define classes and class hierarchy

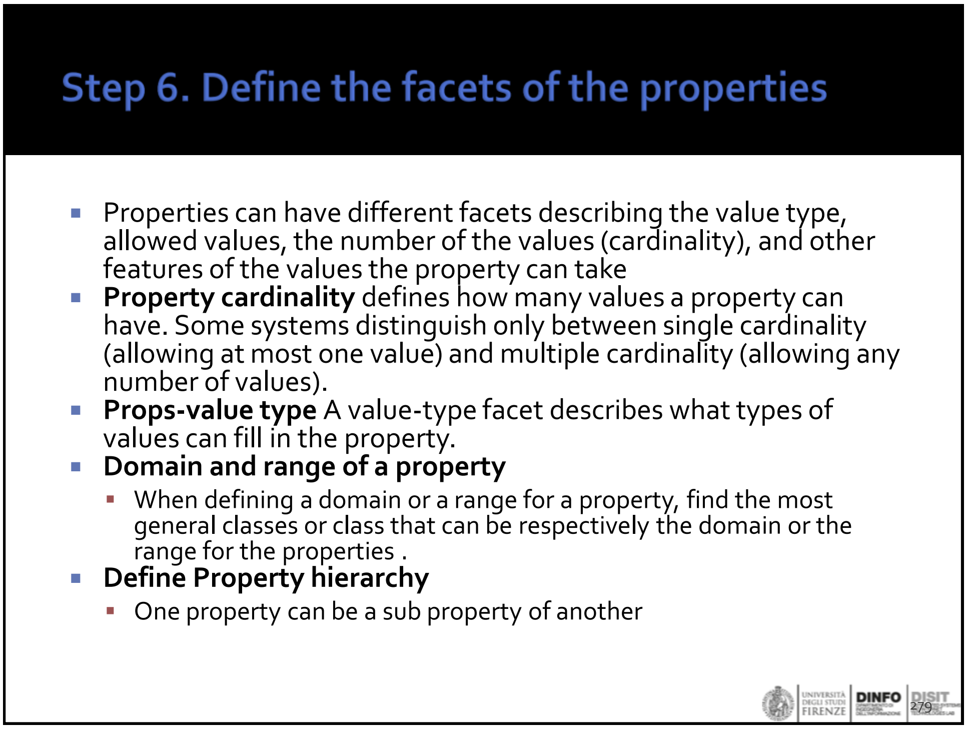
- There are several possible approaches in developing a class hierarchy:
  - Top-down
  - Bottom-up
  - A combination
- From the list created in Step 3, we select the terms that describe objects having independent existence rather than terms that describe these objects. These terms will be classes in the ontology and will become anchors in the class hierarchy.
- Organize the classes into a hierarchical taxonomy by asking if by being an instance of one class, the object will necessarily (i.e., by definition) be an instance of some other class.



Step 5. Define the properties of classes

- The classes alone will not provide enough information to answer the competency questions from Step 1.
- Once we have defined some of the classes, we must describe the internal structure of concepts.
- We have already selected classes from the list of terms we created in Step 3. Most of the remaining terms are likely to be properties of these classes.
- In general, there are several types of object properties that can become slots in an ontology:
  - "intrinsic" properties such as the flavor of a wine;
  - "extrinsic" properties such as a wine's name, and area it comes from;
  - parts, if the object is structured; these can be both physical and abstract "parts" (e.g., the courses of a meal)
  - relationships to other individuals; these are the relationships between individual members of the class and other items

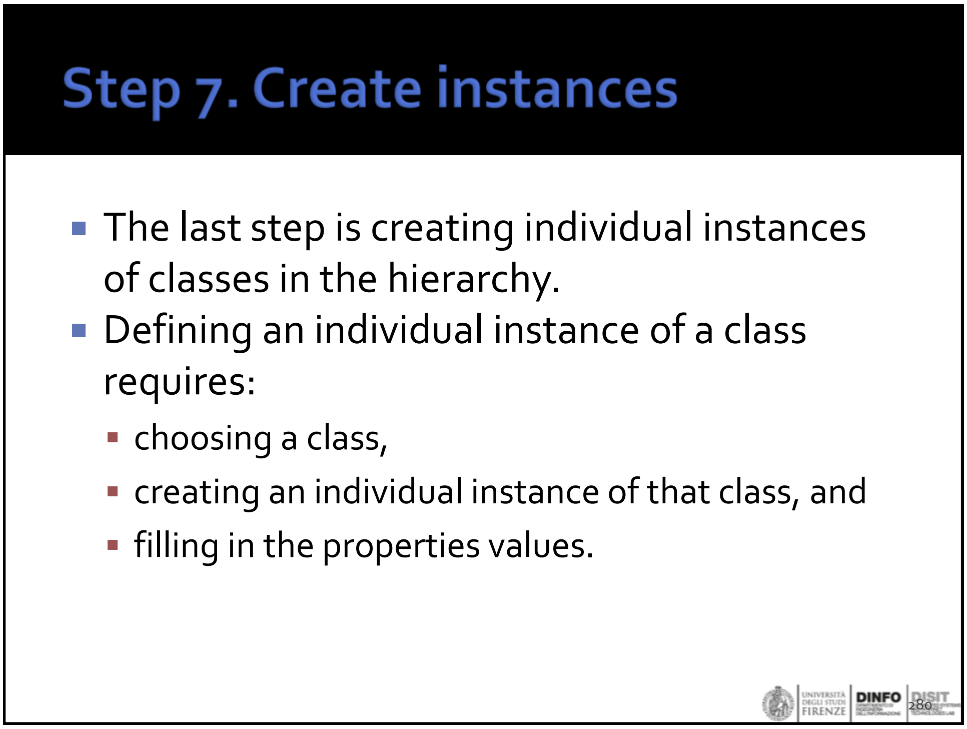
278



Step 6. Define the facets of the properties

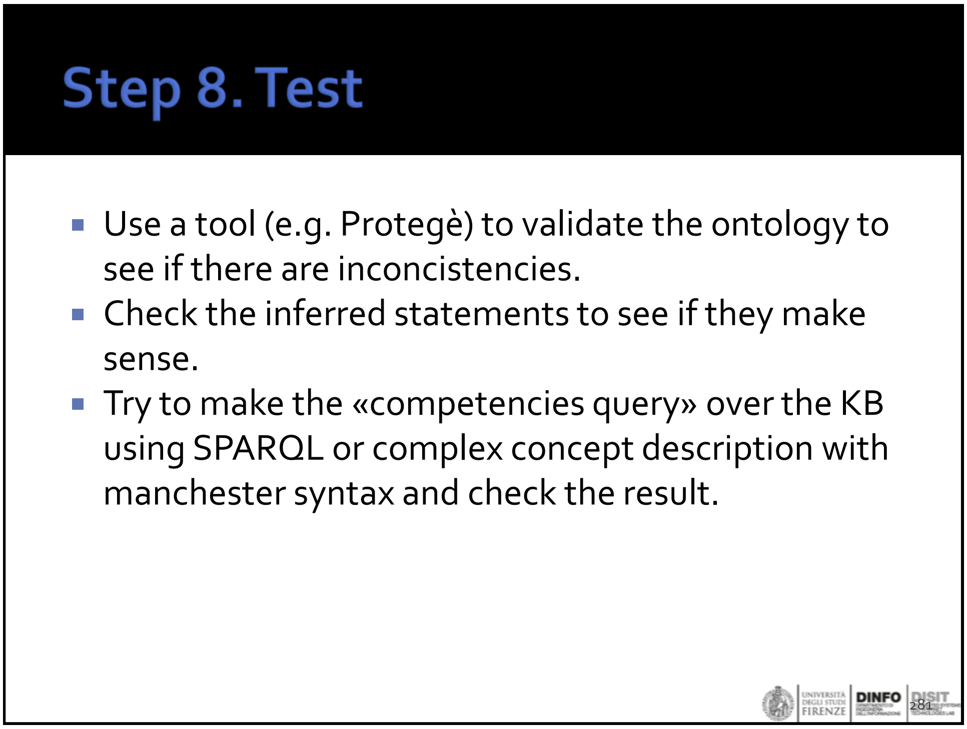
- Properties can have different facets describing the value type, allowed values, the number of the values (cardinality), and other features of the values the property can take
- **Property cardinality** defines how many values a property can have. Some systems distinguish only between single cardinality (allowing at most one value) and multiple cardinality (allowing any number of values).
- **Props-value type** A value-type facet describes what types of values can fill in the property.
- **Domain and range of a property**
  - When defining a domain or a range for a property, find the most general classes or class that can be respectively the domain or the range for the properties .
- **Define Property hierarchy**
  - One property can be a sub property of another

279



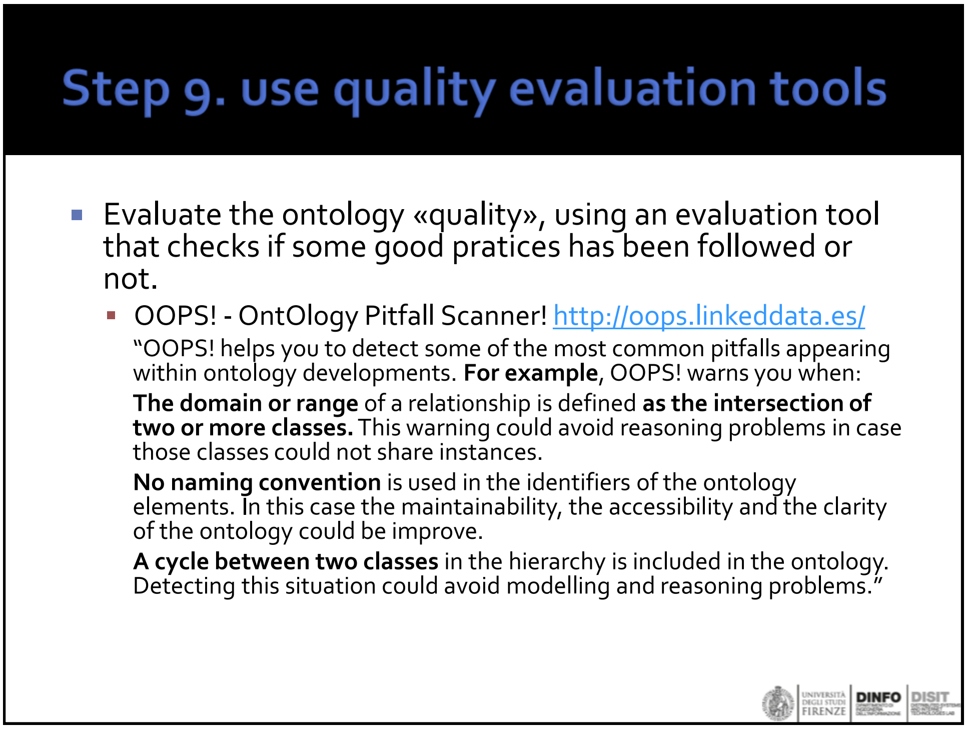
Step 7. Create instances

- The last step is creating individual instances of classes in the hierarchy.
- Defining an individual instance of a class requires:
  - choosing a class,
  - creating an individual instance of that class, and
  - filling in the properties values.



Step 8. Test

- Use a tool (e.g. Protegè) to validate the ontology to see if there are inconcistencies.
- Check the inferred statements to see if they make sense.
- Try to make the «competencies query» over the KB using SPARQL or complex concept description with manchester syntax and check the result.



Step 9. use quality evaluation tools

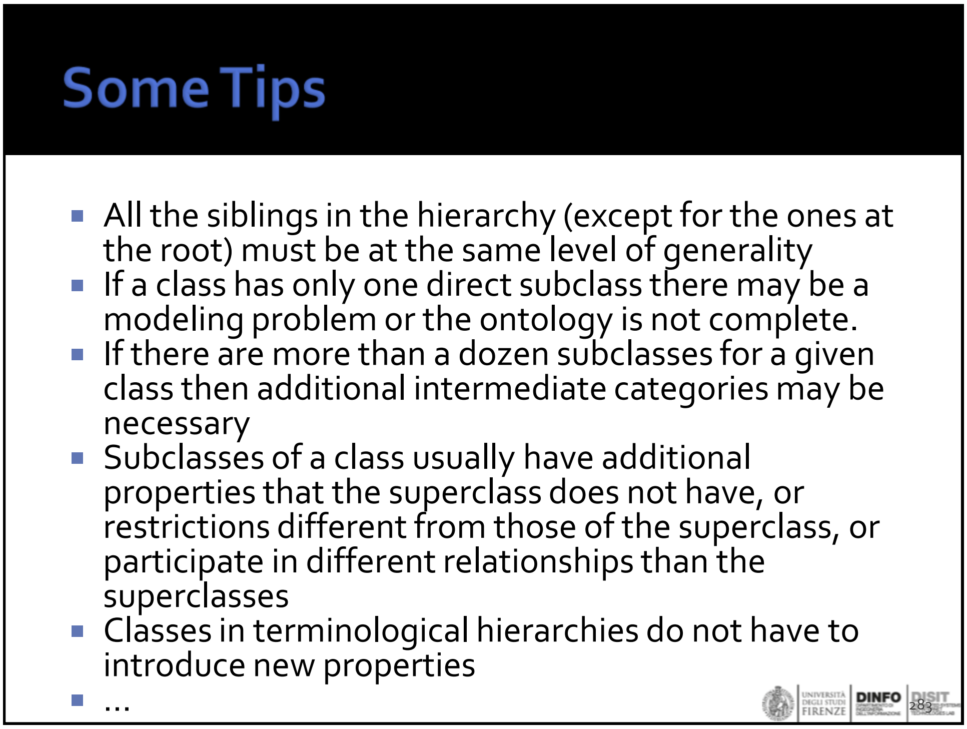
- Evaluate the ontology «quality», using an evaluation tool that checks if some good pratices has been followed or not.
  - OOPS! - OntOlogy Pitfall Scanner! http://oops.linkeddata.es/

    "OOPS! helps you to detect some of the most common pitfalls appearing within ontology developments. **For example**, OOPS! warns you when:

    The domain or range of a relationship is defined **as the intersection of two or more classes.** This warning could avoid reasoning problems in case those classes could not share instances.

    No naming convention is used in the identifiers of the ontology elements. In this case the maintainability, the accessibility and the clarity of the ontology could be improve.

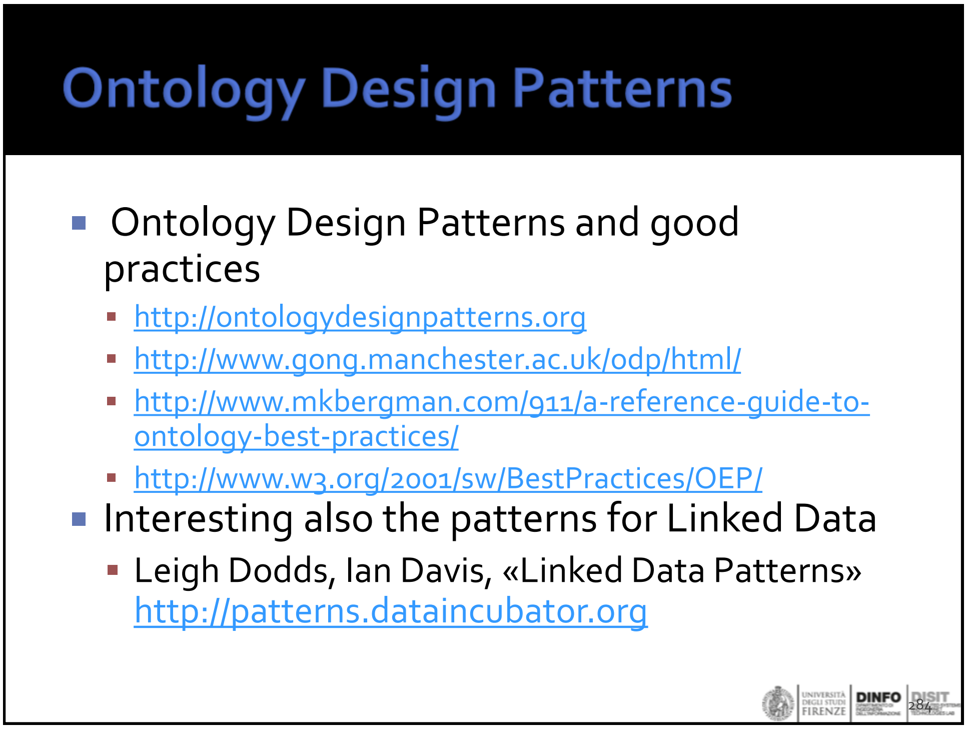
    A cycle between two classes in the hierarchy is included in the ontology. Detecting this situation could avoid modelling and reasoning problems."



Some Tips

- All the siblings in the hierarchy (except for the ones at the root) must be at the same level of generality
- If a class has only one direct subclass there may be a modeling problem or the ontology is not complete.
- If there are more than a dozen subclasses for a given class then additional intermediate categories may be necessary
- Subclasses of a class usually have additional properties that the superclass does not have, or restrictions different from those of the superclass, or participate in different relationships than the superclasses
- Classes in terminological hierarchies do not have to introduce new properties
- …

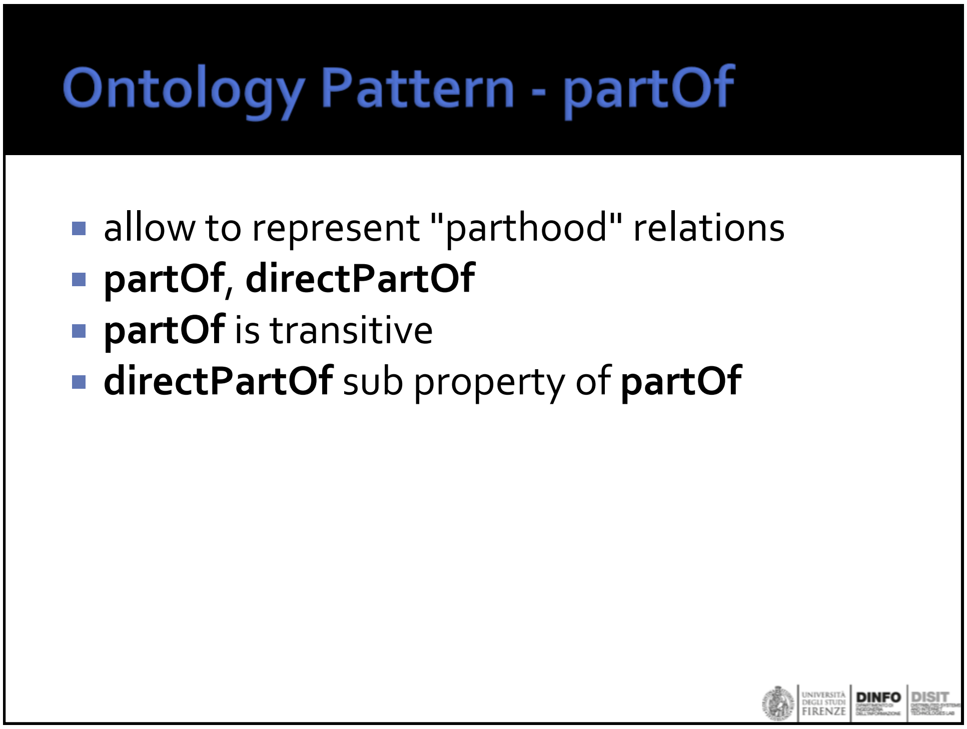
283



Ontology Design Patterns

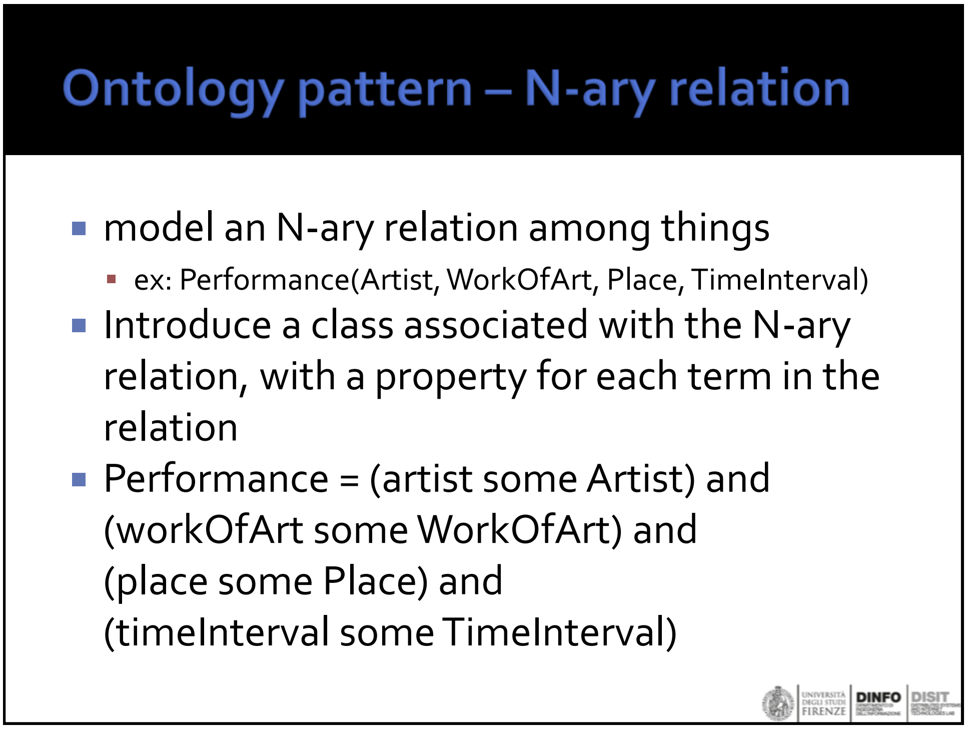
- Ontology Design Patterns and good practices
  - http://ontologydesignpatterns.org
  - http://www.gong.manchester.ac.uk/odp/html/
  - http://www.mkbergman.com/911/a-reference-guide-to-ontology-best-practices/
  - http://www.w3.org/2001/sw/BestPractices/OEP/
- Interesting also the patterns for Linked Data
  - Leigh Dodds, Ian Davis, «Linked Data Patterns» http://patterns.dataincubator.org

284



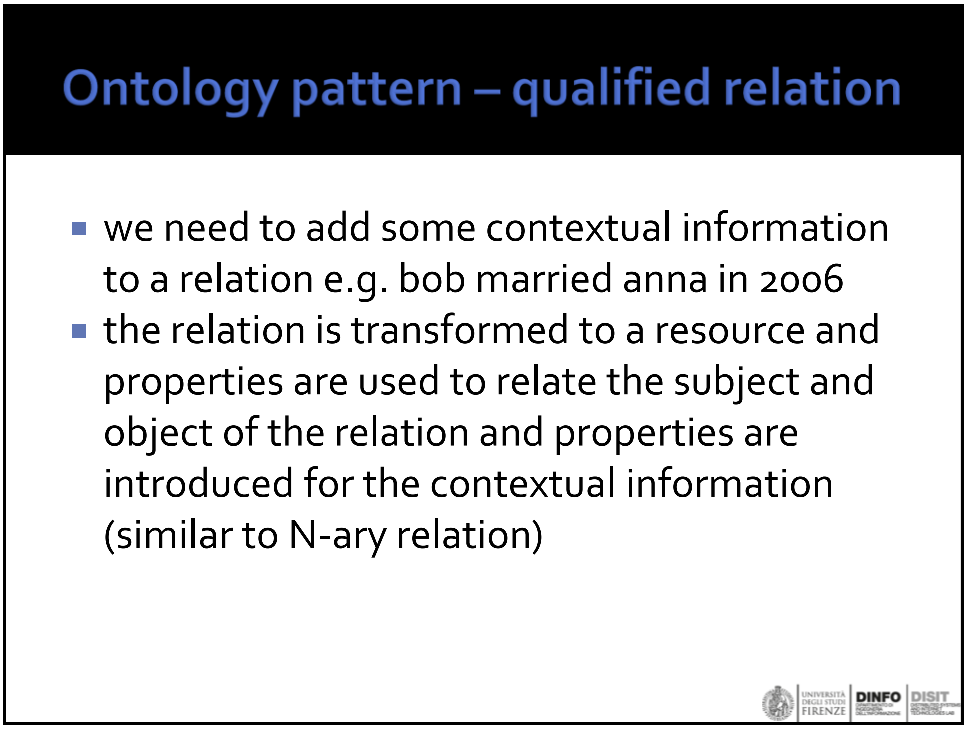
Ontology Pattern - partOf

- allow to represent "parthood" relations
- **partOf**, **directPartOf**
- **partOf** is transitive
- **directPartOf** sub property of **partOf**



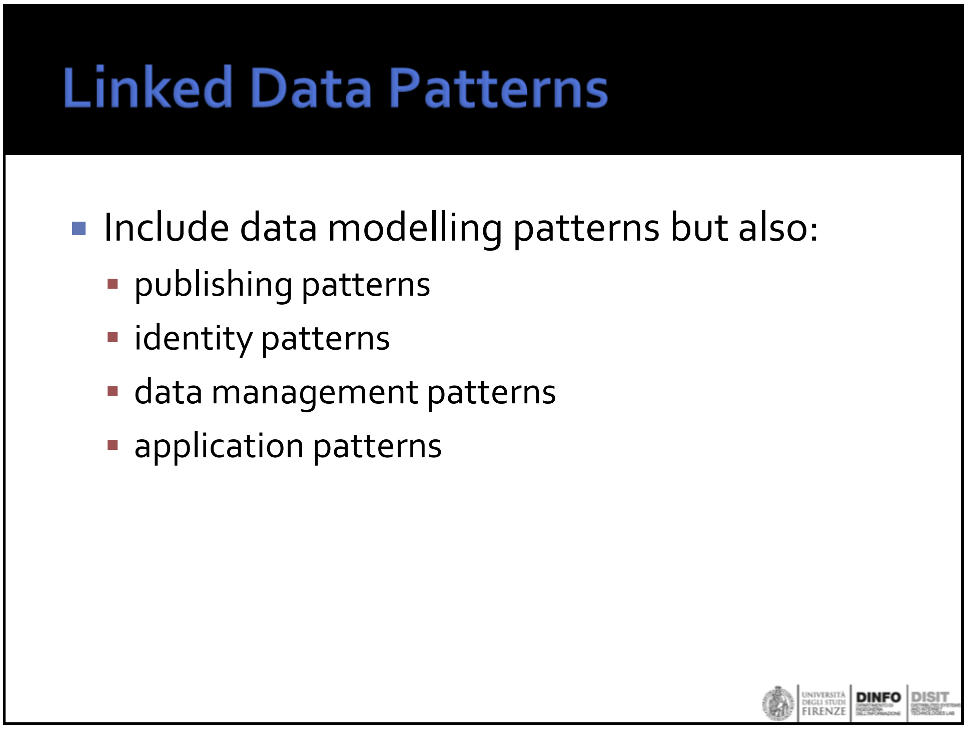
Ontology pattern – N-ary relation

- model an N-ary relation among things
  - ex: Performance(Artist, WorkOfArt, Place, TimeInterval)
- Introduce a class associated with the N-ary relation, with a property for each term in the relation
- Performance = (artist some Artist) and
  (workOfArt some WorkOfArt) and
  (place some Place) and
  (timeInterval some TimeInterval)



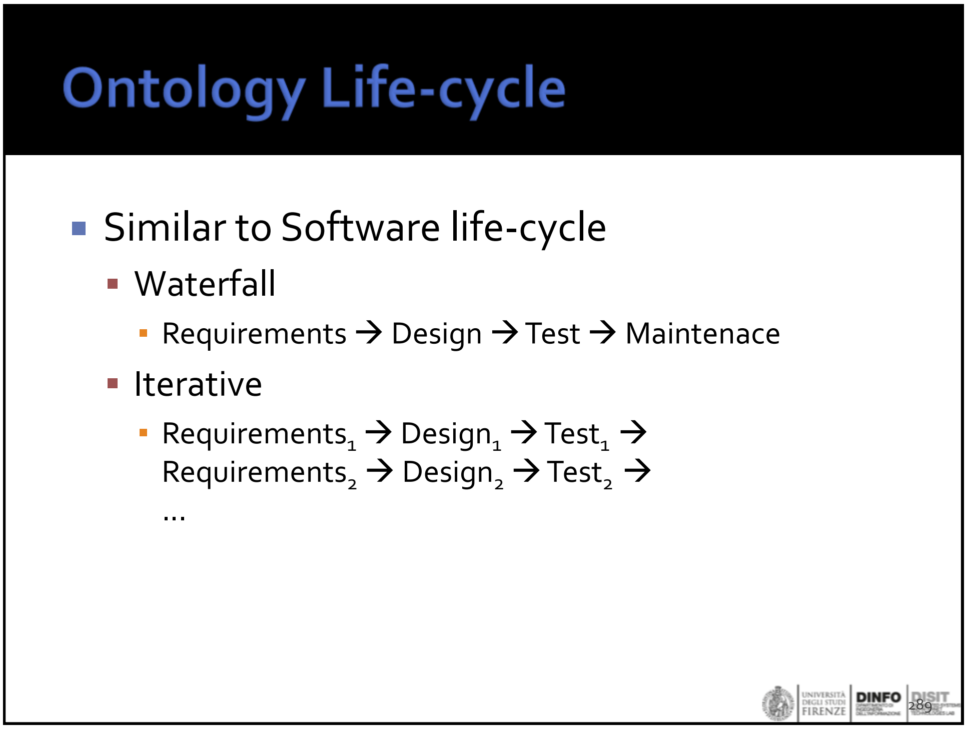
Ontology pattern – qualified relation

- we need to add some contextual information to a relation e.g. bob married anna in 2006
- the relation is transformed to a resource and properties are used to relate the subject and object of the relation and properties are introduced for the contextual information (similar to N-ary relation)



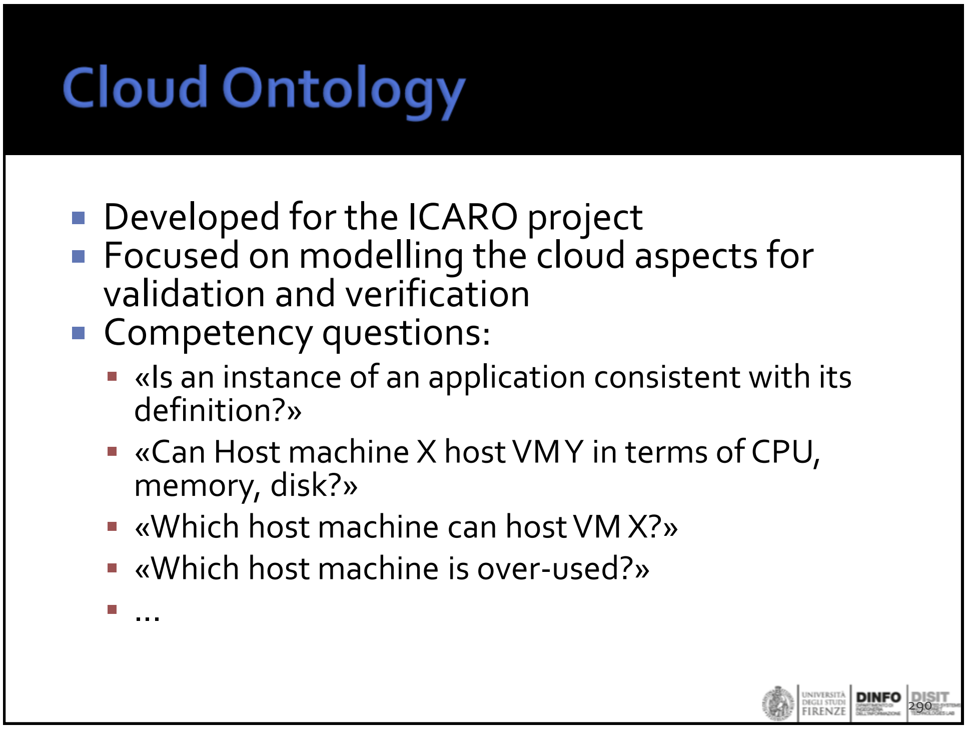
Linked Data Patterns

- Include data modelling patterns but also:
  - publishing patterns
  - identity patterns
  - data management patterns
  - application patterns



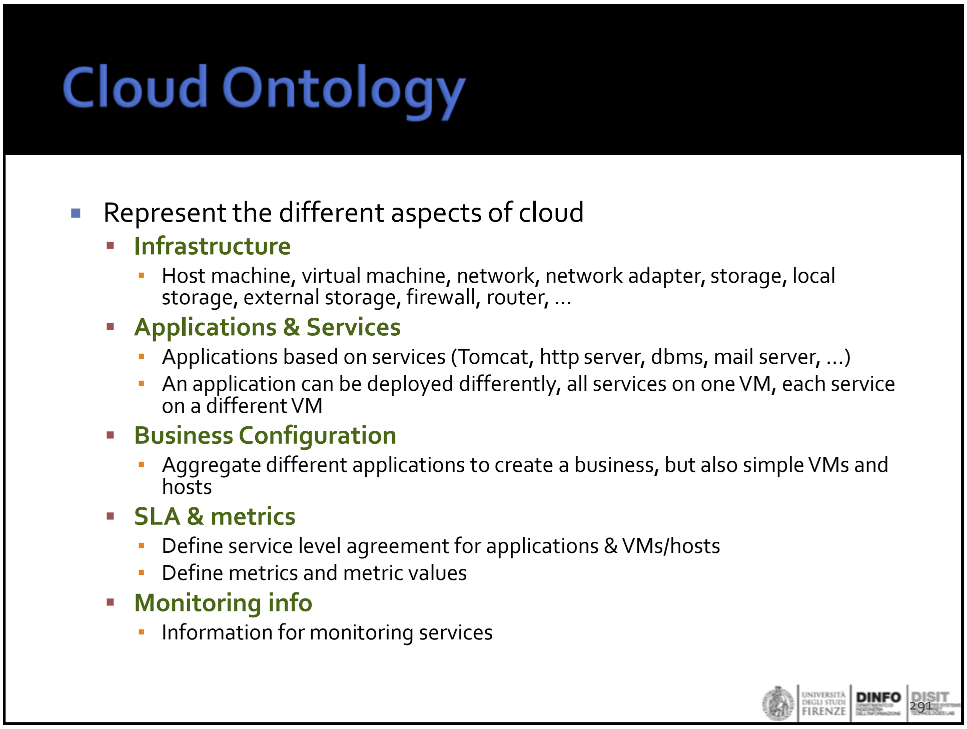
Ontology Life-cycle

- Similar to Software life-cycle
  - Waterfall
    - Requirements → Design → Test → Maintenace
  - Iterative
    - Requirements$_1$ → Design$_1$ → Test$_1$ → Requirements$_2$ → Design$_2$ → Test$_2$ → …



Cloud Ontology

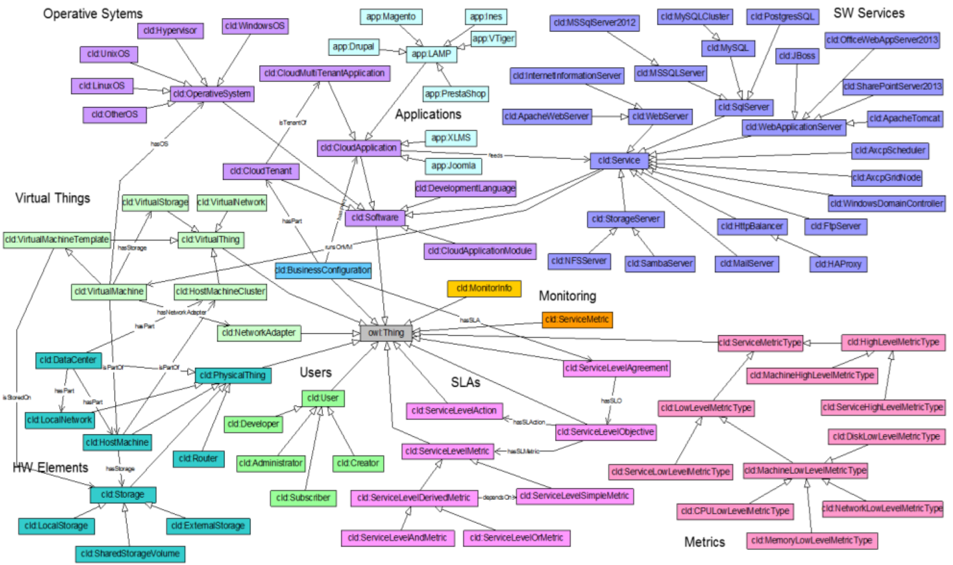
- Developed for the ICARO project
- Focused on modelling the cloud aspects for validation and verification
- Competency questions:
  - «Is an instance of an application consistent with its definition?»
  - «Can Host machine X host VM Y in terms of CPU, memory, disk?»
  - «Which host machine can host VM X?»
  - «Which host machine is over-used?»
  - …



Cloud Ontology

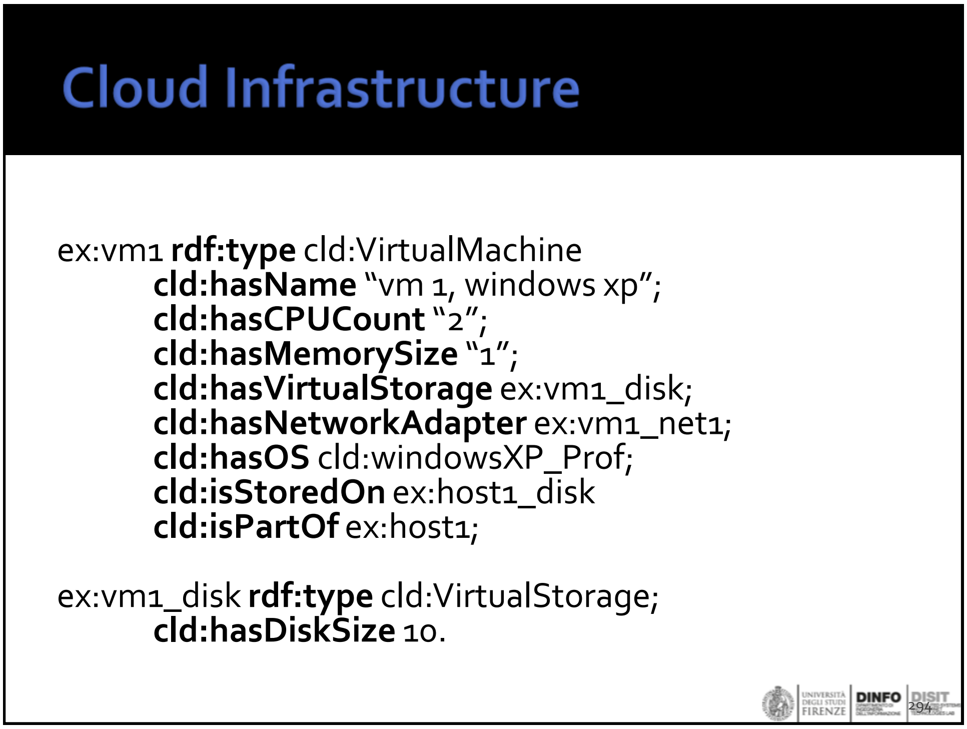
- Represent the different aspects of cloud
  - **Infrastructure**
    - Host machine, virtual machine, network, network adapter, storage, local storage, external storage, firewall, router, …
  - **Applications & Services**
    - Applications based on services (Tomcat, http server, dbms, mail server, …)
    - An application can be deployed differently, all services on one VM, each service on a different VM
  - **Business Configuration**
    - Aggregate different applications to create a business, but also simple VMs and hosts
  - **SLA & metrics**
    - Define service level agreement for applications & VMs/hosts
    - Define metrics and metric values
  - **Monitoring info**
    - Information for monitoring services

Cloud Ontology



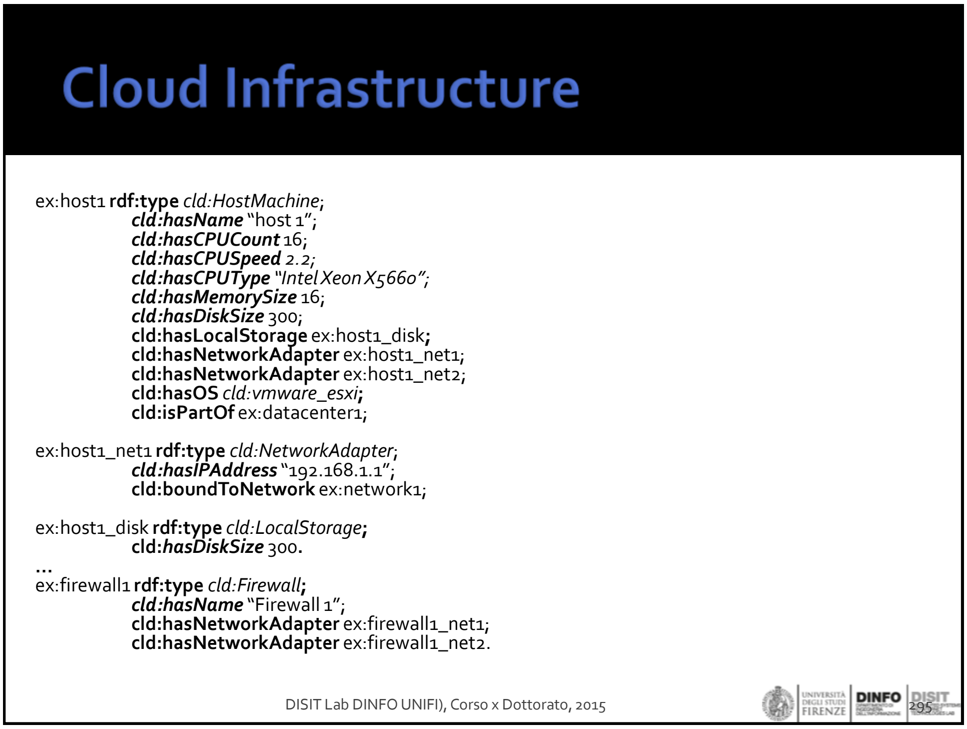
Cloud Infrastructure

```
ex:datacenter1 rdf:type cld:DataCenter;
        cld:hasName "production data center";
        cld:hasPart ex:host1;
        …
        cld:hasPart ex:host100;
        cld:hasPart ex:storage1;
        cld:hasPart ex:firewall1;
        cld:hasPart ex:firewall2;
```



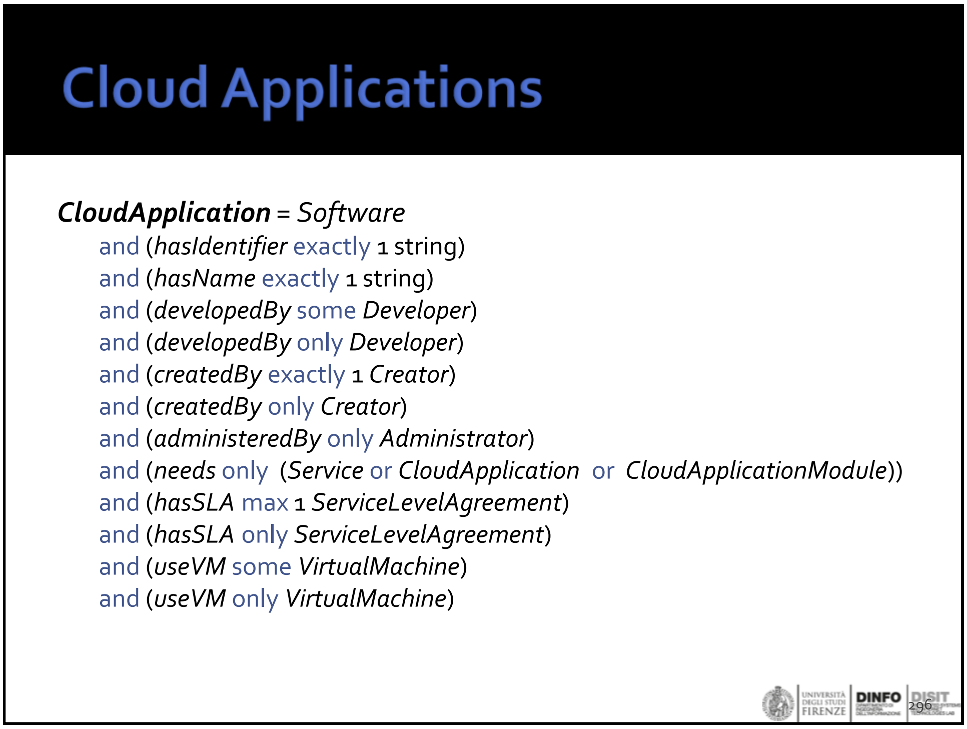
Cloud Infrastructure

```
ex:vm1 rdf:type cld:VirtualMachine
        cld:hasName "vm 1, windows xp";
        cld:hasCPUCount "2";
        cld:hasMemorySize "1";
        cld:hasVirtualStorage ex:vm1_disk;
        cld:hasNetworkAdapter ex:vm1_net1;
        cld:hasOS cld:windowsXP_Prof;
        cld:isStoredOn ex:host1_disk
        cld:isPartOf ex:host1;

ex:vm1_disk rdf:type cld:VirtualStorage;
        cld:hasDiskSize 10.
```



Cloud Infrastructure

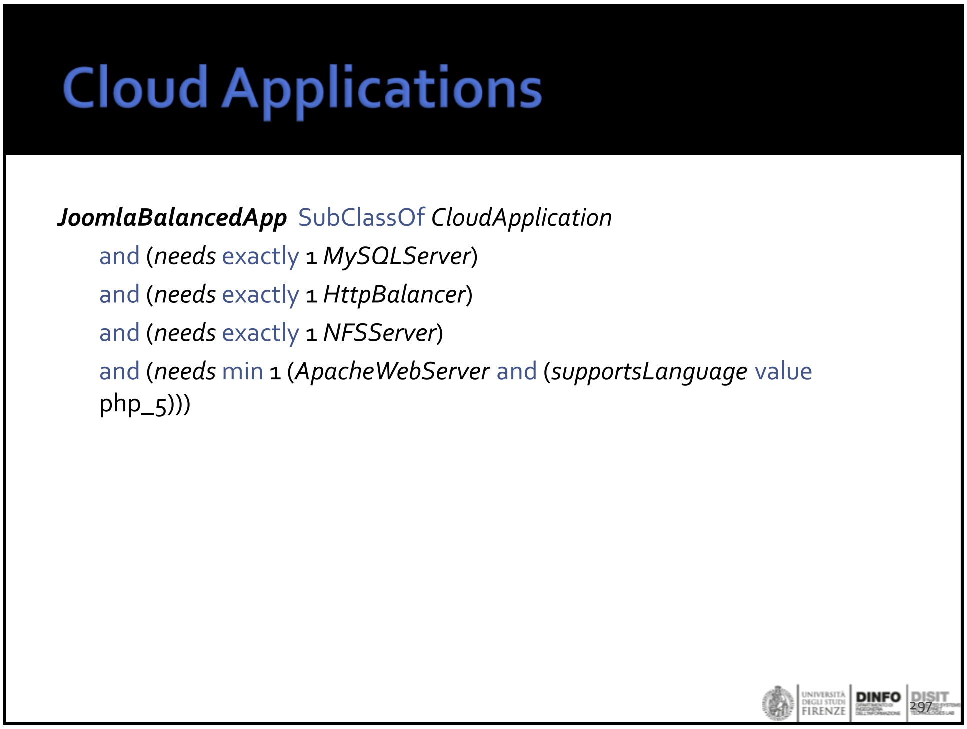
```
ex:host1 rdf:type cld:HostMachine;
        cld:hasName "host 1";
        cld:hasCPUCount 16;
        cld:hasCPUSpeed 2.2;
        cld:hasCPUType "Intel Xeon X5660";
        cld:hasMemorySize 16;
        cld:hasDiskSize 300;
        cld:hasLocalStorage ex:host1_disk;
        cld:hasNetworkAdapter ex:host1_net1;
        cld:hasNetworkAdapter ex:host1_net2;
        cld:hasOS cld:vmware_esxi;
        cld:isPartOf ex:datacenter1;

ex:host1_net1 rdf:type cld:NetworkAdapter;
        cld:hasIPAddress "192.168.1.1";
        cld:boundToNetwork ex:network1;

ex:host1_disk rdf:type cld:LocalStorage;
        cld:hasDiskSize 300.
...
ex:firewall1 rdf:type cld:Firewall;
        cld:hasName "Firewall 1";
        cld:hasNetworkAdapter ex:firewall1_net1;
        cld:hasNetworkAdapter ex:firewall1_net2.
```



Cloud Applications

CloudApplication = *Software*
and (*hasIdentifier* exactly 1 string)
and (*hasName* exactly 1 string)
and (*developedBy* some *Developer*)
and (*developedBy* only *Developer*)
and (*createdBy* exactly 1 *Creator*)
and (*createdBy* only *Creator*)
and (*administeredBy* only *Administrator*)
and (*needs* only (*Service* or *CloudApplication* or *CloudApplicationModule*))
and (*hasSLA* max 1 *ServiceLevelAgreement*)
and (*hasSLA* only *ServiceLevelAgreement*)
and (*useVM* some *VirtualMachine*)
and (*useVM* only *VirtualMachine*)

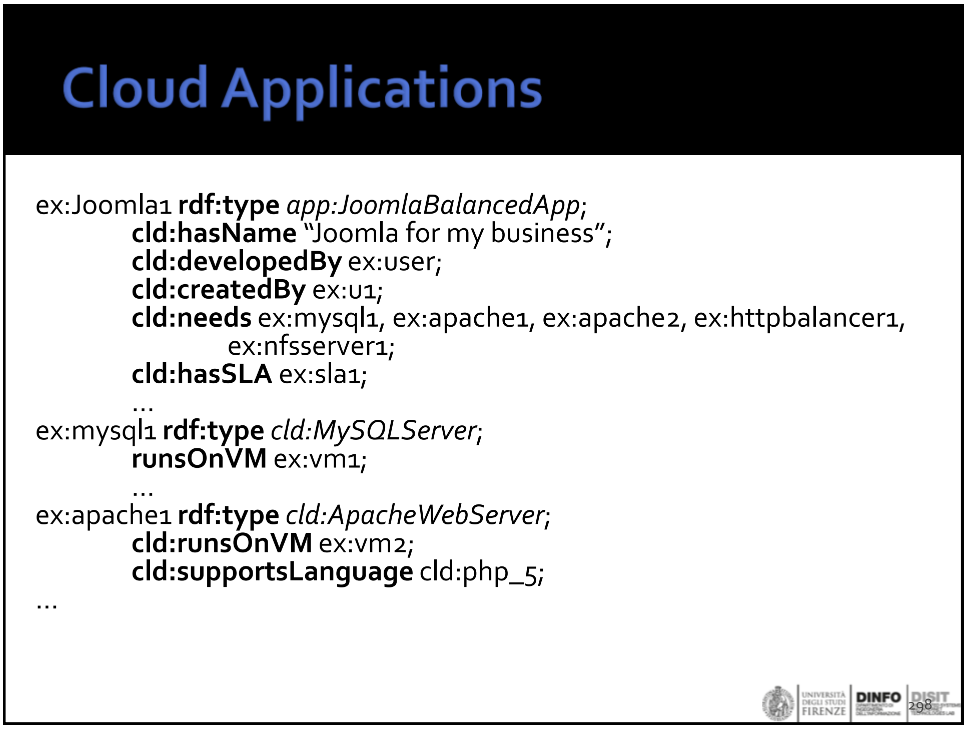
296



Cloud Applications

JoomlaBalancedApp SubClassOf *CloudApplication*
and (*needs* exactly 1 *MySQLServer*)
and (*needs* exactly 1 *HttpBalancer*)
and (*needs* exactly 1 *NFSServer*)
and (*needs* min 1 (*ApacheWebServer* and (*supportsLanguage* value php_5)))

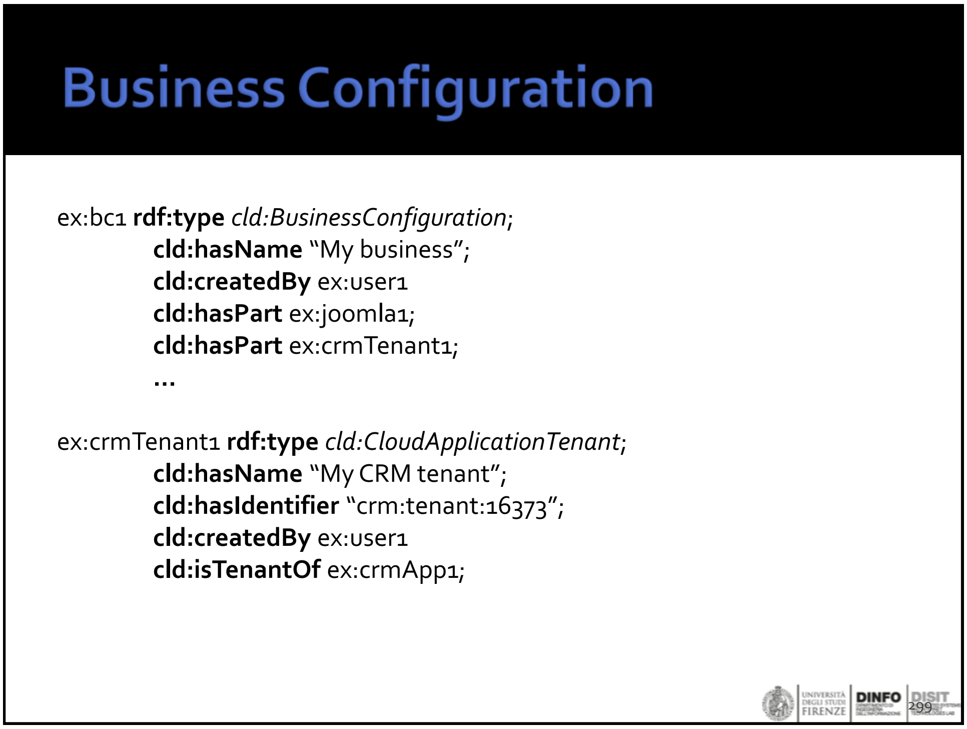
297



Cloud Applications

```
ex:Joomla1 rdf:type app:JoomlaBalancedApp;
        cld:hasName “Joomla for my business”;
        cld:developedBy ex:user;
        cld:createdBy ex:u1;
        cld:needs ex:mysql1, ex:apache1, ex:apache2, ex:httpbalancer1,
                ex:nfsserver1;
        cld:hasSLA ex:sla1;
        …
ex:mysql1 rdf:type cld:MySQLServer;
        runsOnVM ex:vm1;
        …
ex:apache1 rdf:type cld:ApacheWebServer;
        cld:runsOnVM ex:vm2;
        cld:supportsLanguage cld:php_5;
…
```

298

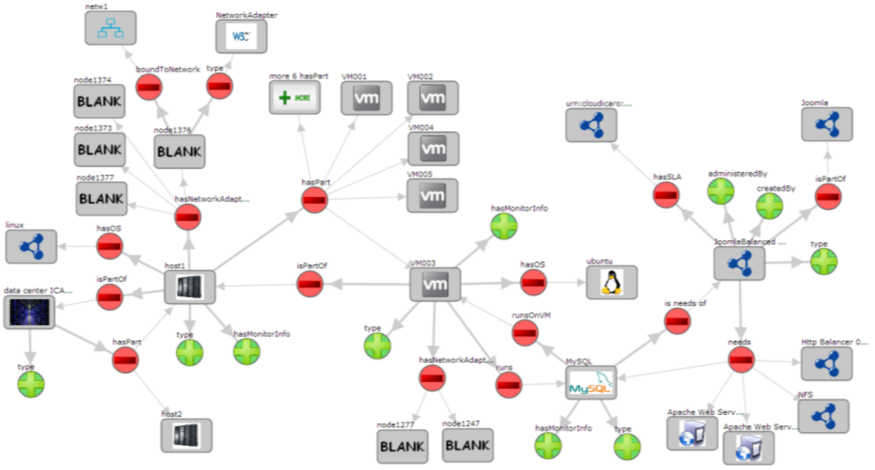


Business Configuration

```
ex:bc1 rdf:type cld:BusinessConfiguration;
        cld:hasName “My business”;
        cld:createdBy ex:user1
        cld:hasPart ex:joomla1;
        cld:hasPart ex:crmTenant1;
        …

ex:crmTenant1 rdf:type cld:CloudApplicationTenant;
        cld:hasName “My CRM tenant”;
        cld:hasIdentifier “crm:tenant:16373”;
        cld:createdBy ex:user1
        cld:isTenantOf ex:crmApp1;
```

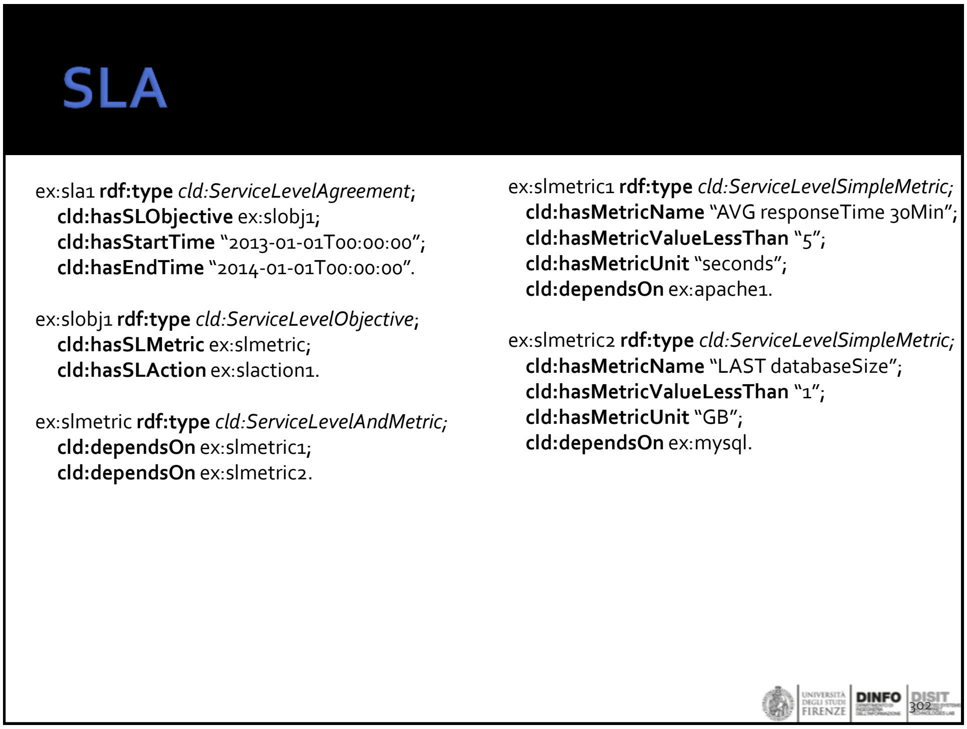
299

log.disit.org



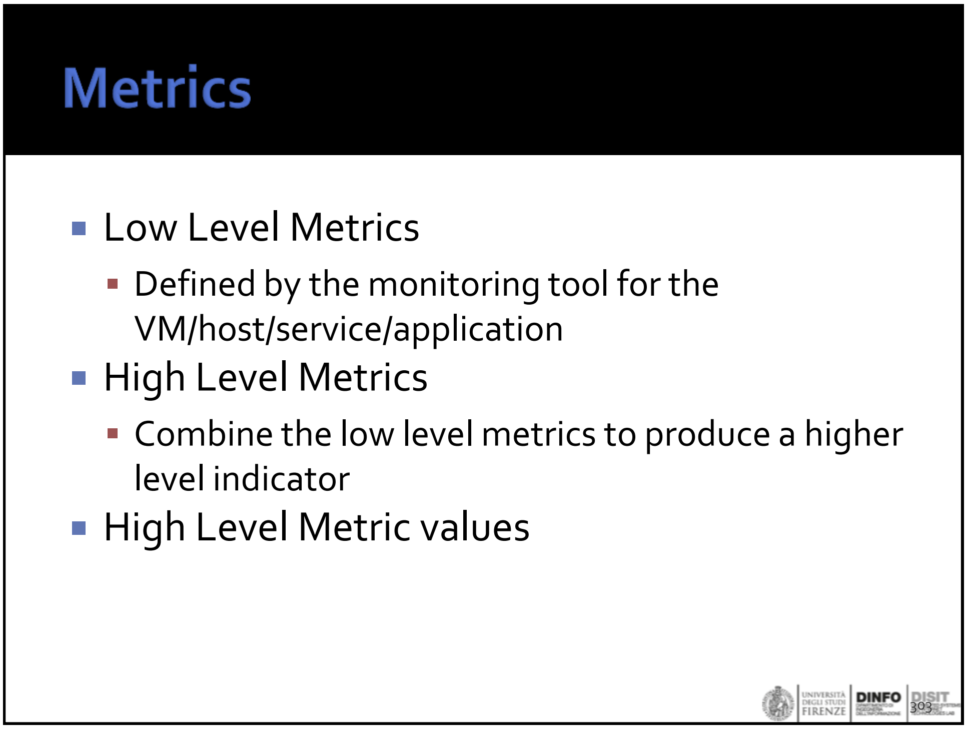
SLA

- Service Level Agreement
- AND/OR Conditions on metric values with reference values
- «AVG responseTime 30Min»(apache1)<5s AND «LAST databaseSize»(mysql)<1GB



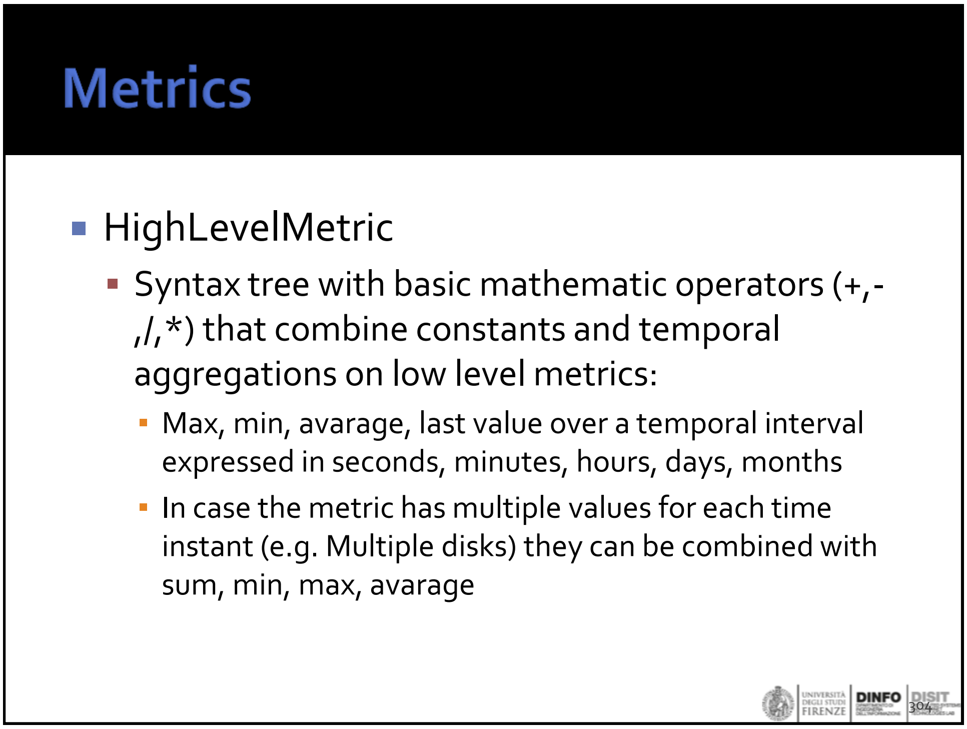
SLA

```
ex:sla1 rdf:type cld:ServiceLevelAgreement;
   cld:hasSLObjective ex:slobj1;
   cld:hasStartTime "2013-01-01T00:00:00";
   cld:hasEndTime "2014-01-01T00:00:00".

ex:slobj1 rdf:type cld:ServiceLevelObjective;
   cld:hasSLMetric ex:slmetric;
   cld:hasSLAction ex:slaction1.

ex:slmetric rdf:type cld:ServiceLevelAndMetric;
   cld:dependsOn ex:slmetric1;
   cld:dependsOn ex:slmetric2.

ex:slmetric1 rdf:type cld:ServiceLevelSimpleMetric;
   cld:hasMetricName "AVG responseTime 30Min";
   cld:hasMetricValueLessThan "5";
   cld:hasMetricUnit "seconds";
   cld:dependsOn ex:apache1.

ex:slmetric2 rdf:type cld:ServiceLevelSimpleMetric;
   cld:hasMetricName "LAST databaseSize";
   cld:hasMetricValueLessThan "1";
   cld:hasMetricUnit "GB";
   cld:dependsOn ex:mysql.
```



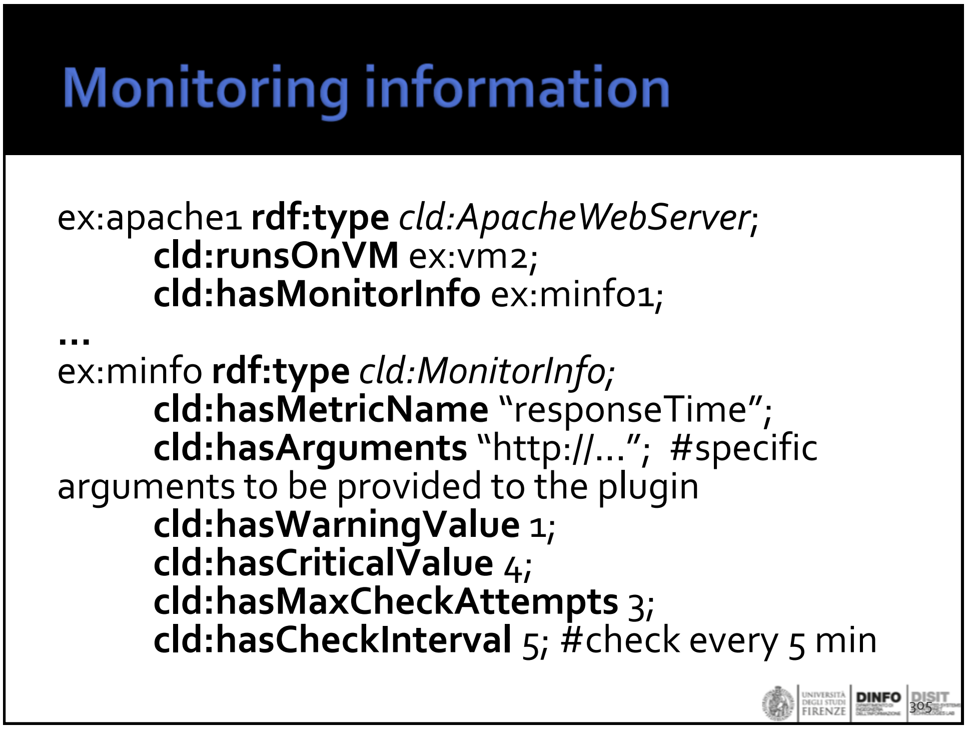
Metrics

- Low Level Metrics
  - Defined by the monitoring tool for the VM/host/service/application
- High Level Metrics
  - Combine the low level metrics to produce a higher level indicator
- High Level Metric values



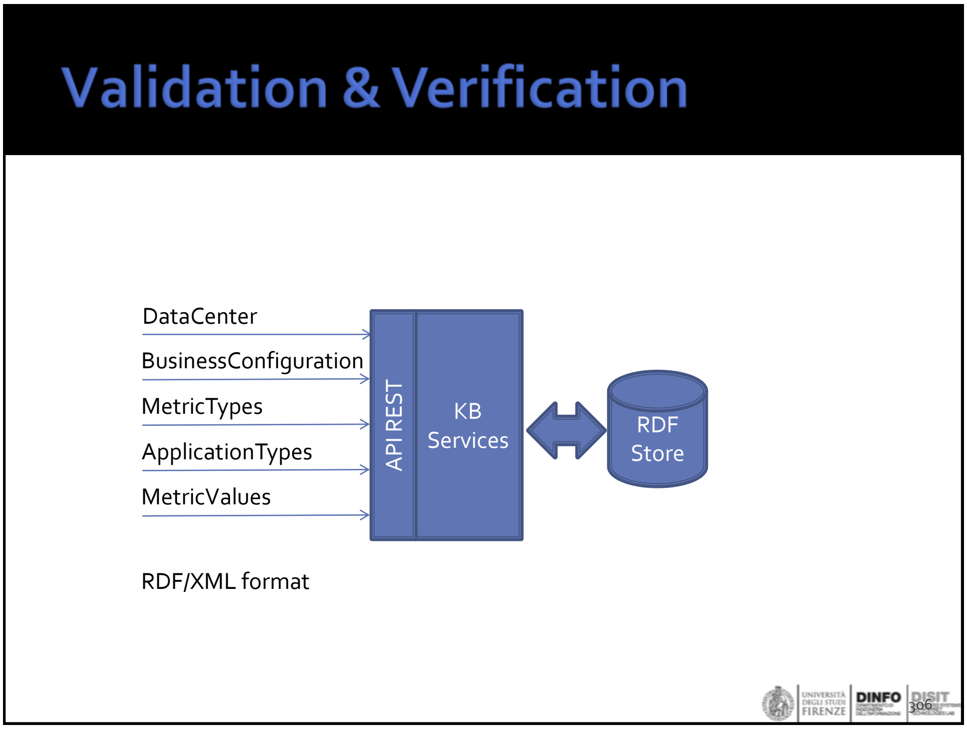
Metrics

- HighLevelMetric
  - Syntax tree with basic mathematic operators (+,-,/,*) that combine constants and temporal aggregations on low level metrics:
    - Max, min, avarage, last value over a temporal interval expressed in seconds, minutes, hours, days, months
    - In case the metric has multiple values for each time instant (e.g. Multiple disks) they can be combined with sum, min, max, avarage



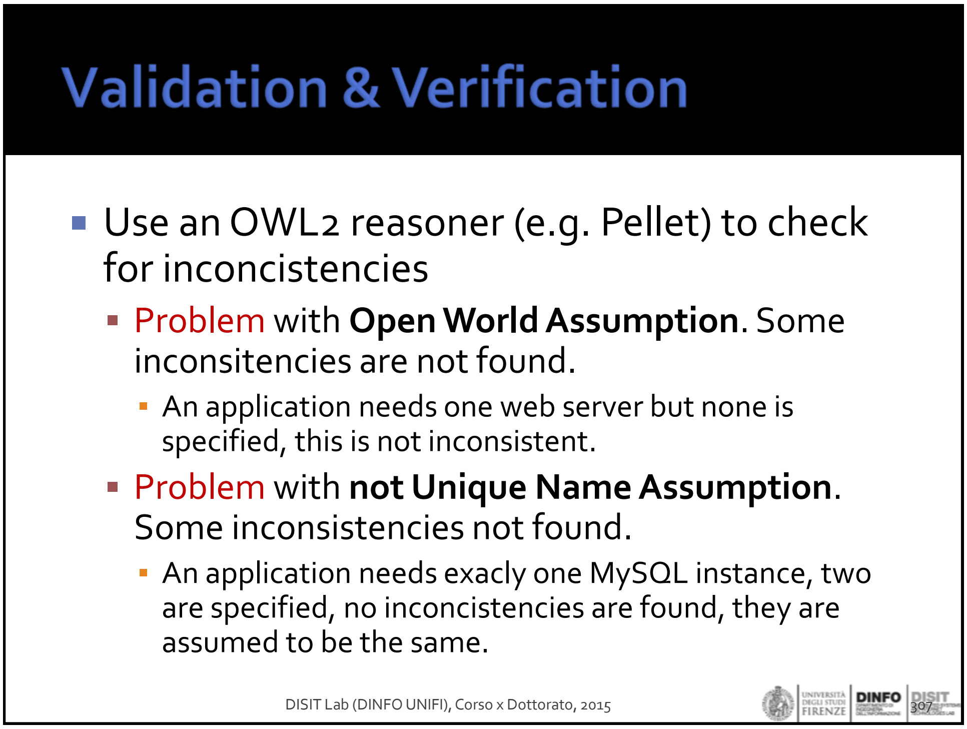
Monitoring information

ex:apache1 **rdf:type** *cld:ApacheWebServer*;
    cld:runsOnVM ex:vm2;
    cld:hasMonitorInfo ex:minfo1;
…
ex:minfo **rdf:type** *cld:MonitorInfo;*
    cld:hasMetricName “responseTime”;
    cld:hasArguments “http://…”; #specific arguments to be provided to the plugin
    cld:hasWarningValue 1;
    cld:hasCriticalValue 4;
    cld:hasMaxCheckAttempts 3;
    cld:hasCheckInterval 5; #check every 5 min



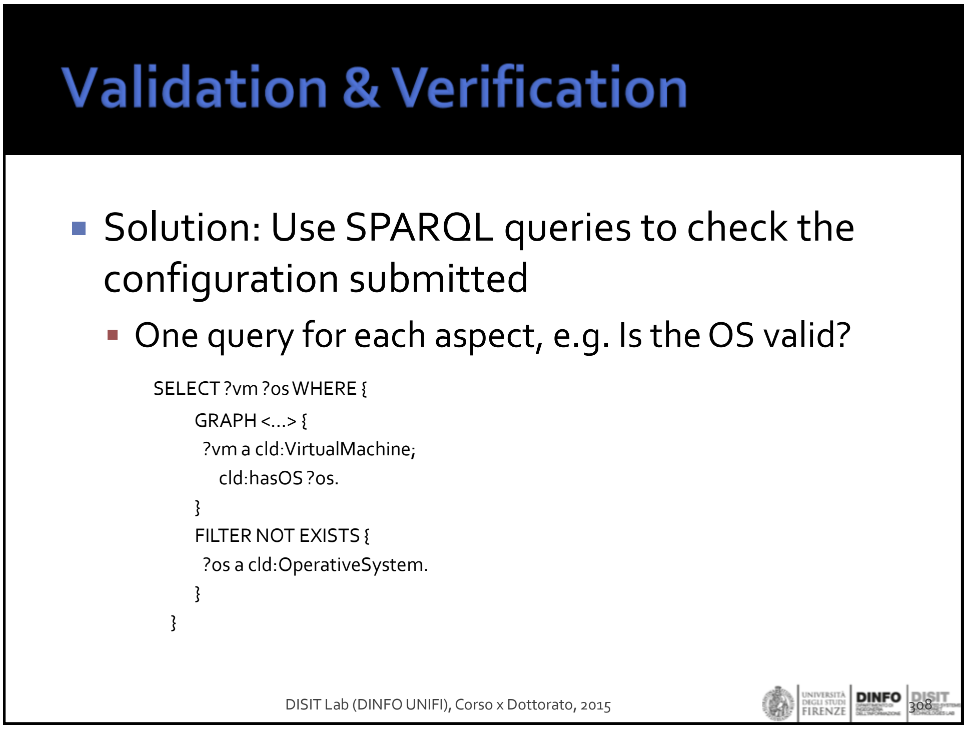
Validation & Verification

RDF/XML format



Validation & Verification

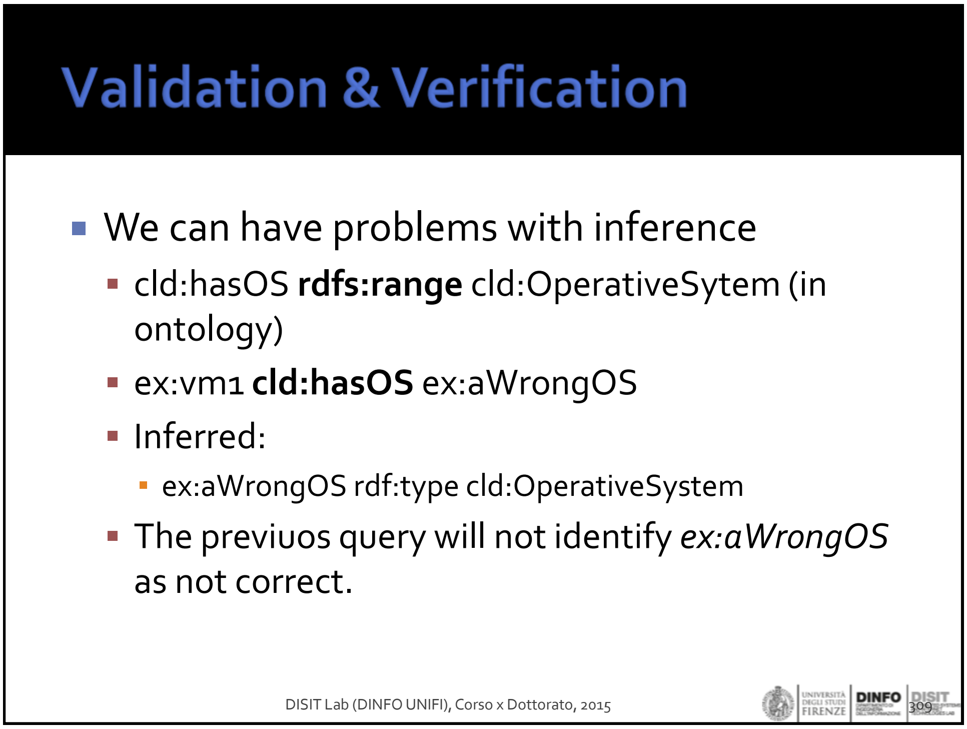
- Use an OWL2 reasoner (e.g. Pellet) to check for inconcistencies
  - Problem with **Open World Assumption**. Some inconsitencies are not found.
    - An application needs one web server but none is specified, this is not inconsistent.
  - Problem with **not Unique Name Assumption**. Some inconsistencies not found.
    - An application needs exacly one MySQL instance, two are specified, no inconcistencies are found, they are assumed to be the same.



Validation & Verification

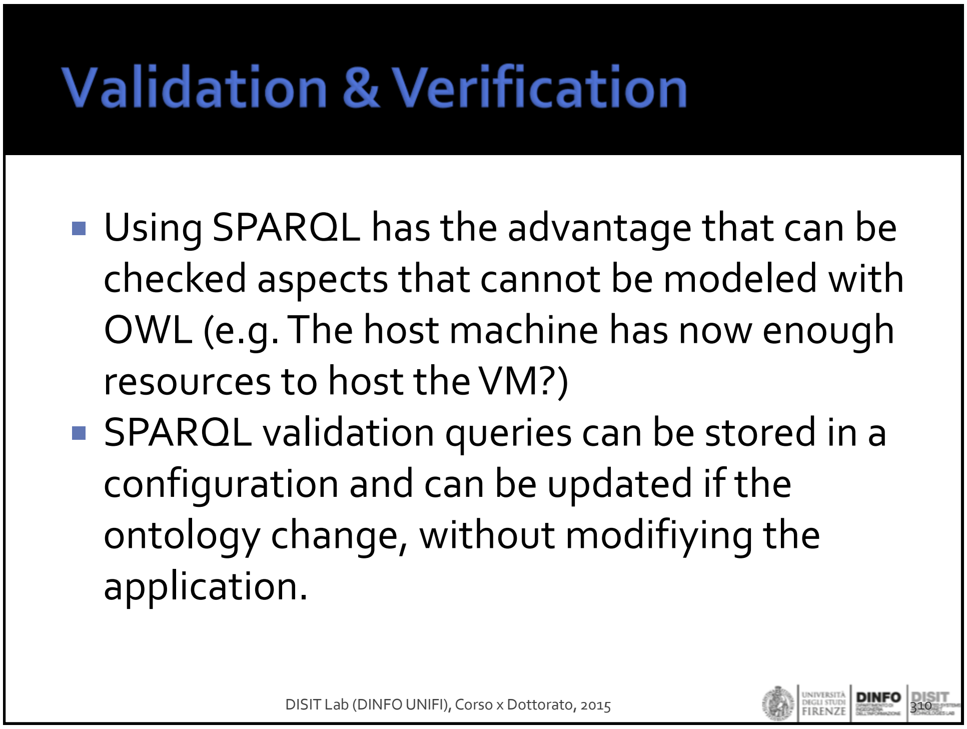
- Solution: Use SPARQL queries to check the configuration submitted
  - One query for each aspect, e.g. Is the OS valid?

```
SELECT ?vm ?os WHERE {
    GRAPH <...> {
     ?vm a cld:VirtualMachine;
       cld:hasOS ?os.
    }
    FILTER NOT EXISTS {
     ?os a cld:OperativeSystem.
    }
  }
```



Validation & Verification

- We can have problems with inference
  - cld:hasOS **rdfs:range** cld:OperativeSytem (in ontology)
  - ex:vm1 **cld:hasOS** ex:aWrongOS
  - Inferred:
    - ex:aWrongOS rdf:type cld:OperativeSystem
  - The previuos query will not identify *ex:aWrongOS* as not correct.

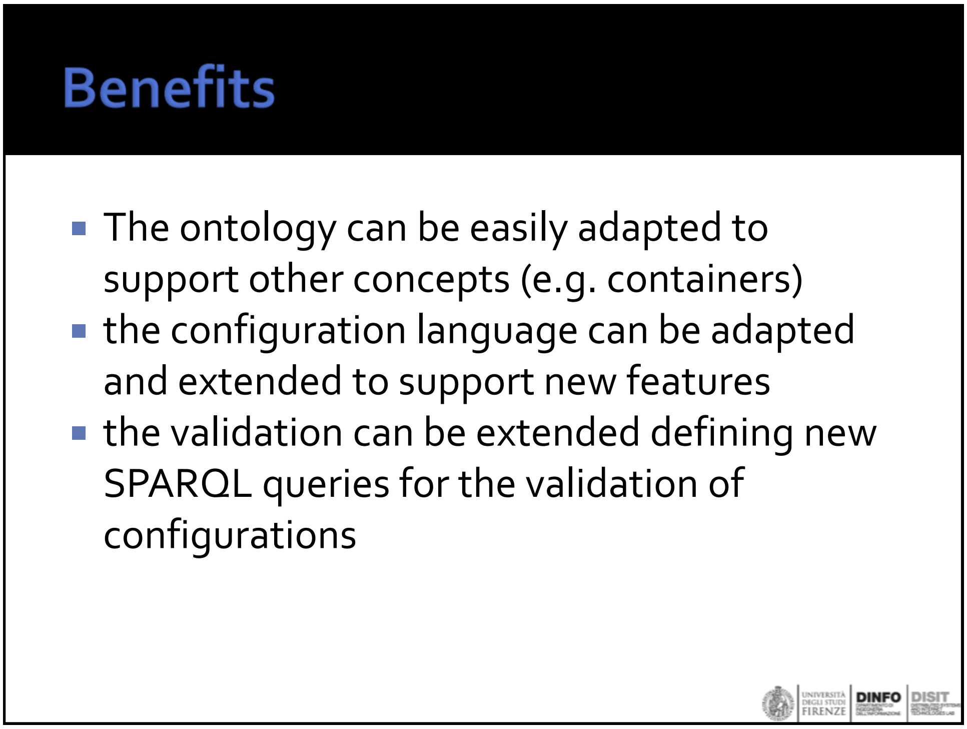


Validation & Verification

- Using SPARQL has the advantage that can be checked aspects that cannot be modeled with OWL (e.g. The host machine has now enough resources to host the VM?)
- SPARQL validation queries can be stored in a configuration and can be updated if the ontology change, without modifiying the application.

DISIT Lab (DINFO UNIFI), Corso x Dottorato, 2015

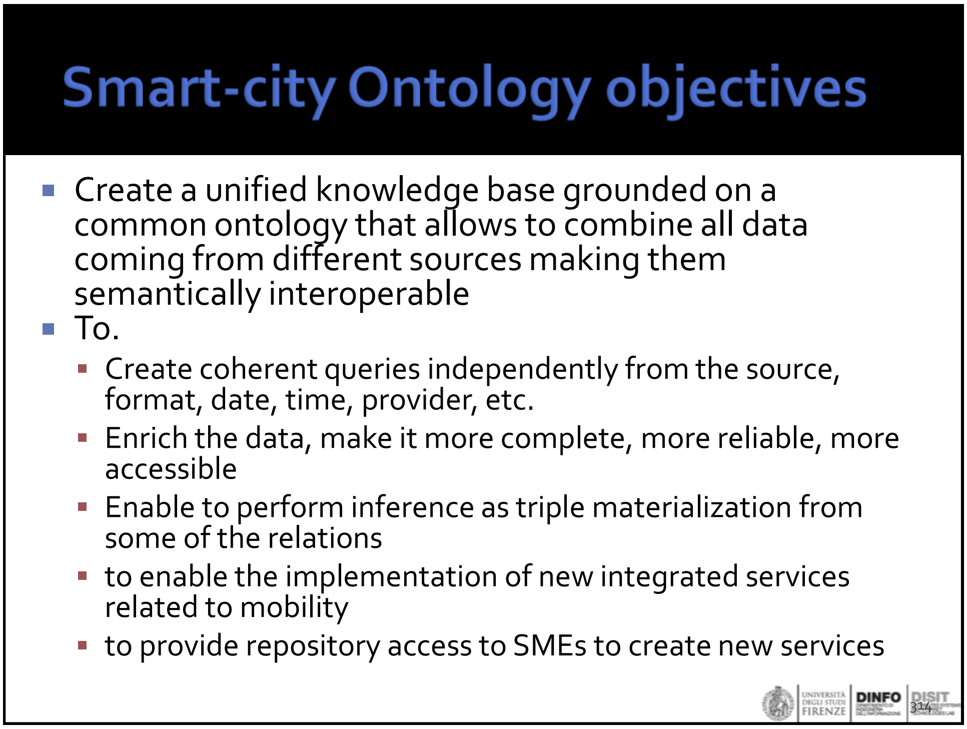
310



Benefits

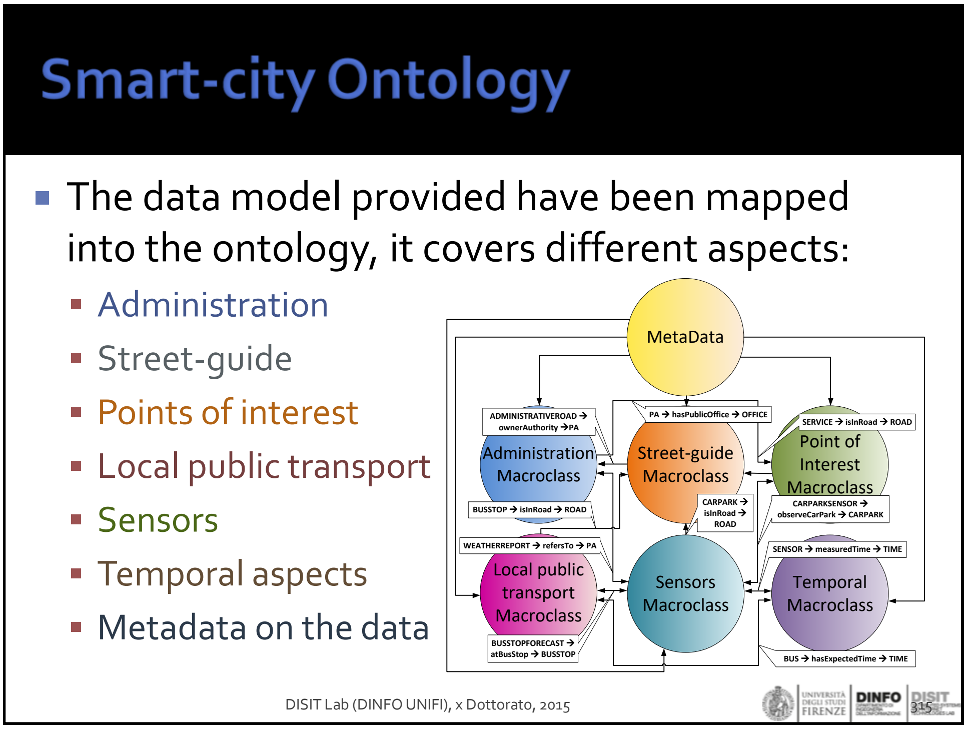
- The ontology can be easily adapted to support other concepts (e.g. containers)
- the configuration language can be adapted and extended to support new features
- the validation can be extended defining new SPARQL queries for the validation of configurations

km4city ontology



Smart-city Ontology objectives

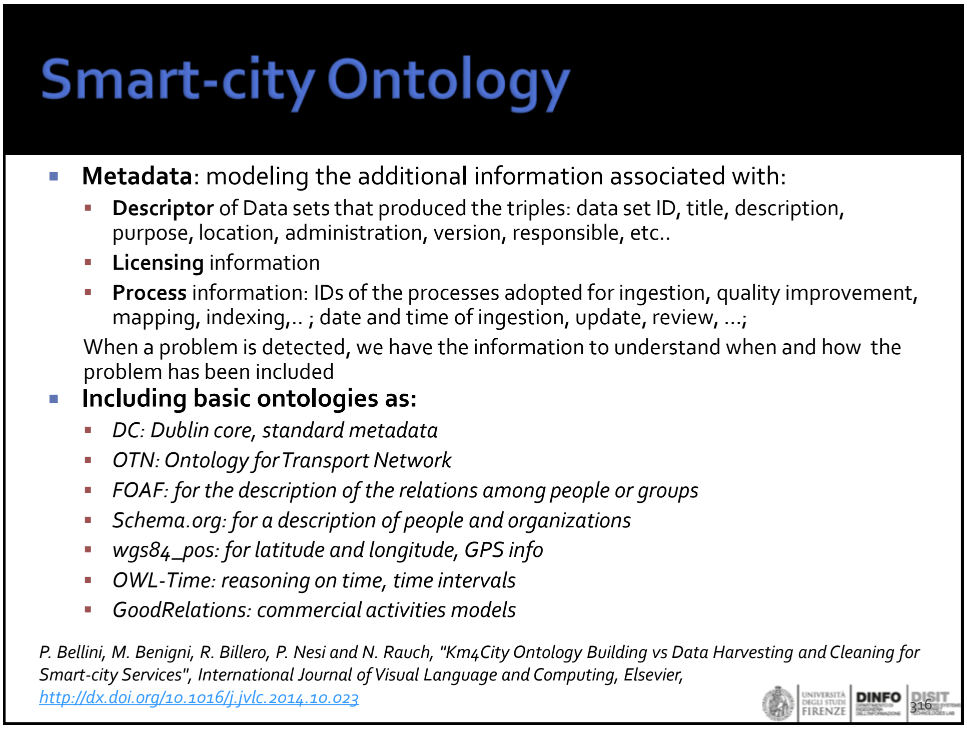
- Create a unified knowledge base grounded on a common ontology that allows to combine all data coming from different sources making them semantically interoperable
- To.
  - Create coherent queries independently from the source, format, date, time, provider, etc.
  - Enrich the data, make it more complete, more reliable, more accessible
  - Enable to perform inference as triple materialization from some of the relations
  - to enable the implementation of new integrated services related to mobility
  - to provide repository access to SMEs to create new services



Smart-city Ontology

- The data model provided have been mapped into the ontology, it covers different aspects:
  - Administration
  - Street-guide
  - Points of interest
  - Local public transport
  - Sensors
  - Temporal aspects
  - Metadata on the data

MetaData
ADMINISTRATIVEROAD → ownerAuthority →PA
PA → hasPublicOffice → OFFICE
SERVICE → isInRoad → ROAD
Administration Macroclass
Street-guide Macroclass
Point of Interest Macroclass
BUSSTOP → isInRoad → ROAD
CARPARK → isInRoad → ROAD
CARPARKSENSOR → observeCarPark → CARPARK
WEATHERREPORT → refersTo → PA
SENSOR → measuredTime → TIME
Local public transport Macroclass
Sensors Macroclass
Temporal Macroclass
BUSSTOPFORECAST → atBusStop → BUSSTOP
BUS → hasExpectedTime → TIME



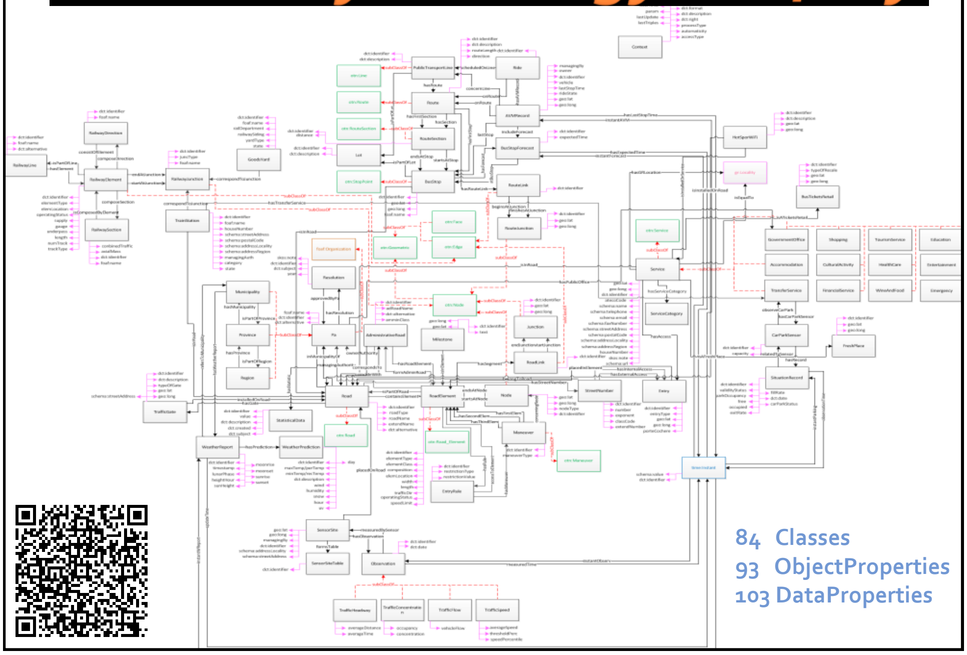
Smart-city Ontology

- **Metadata**: modeling the additional information associated with:
  - **Descriptor** of Data sets that produced the triples: data set ID, title, description, purpose, location, administration, version, responsible, etc..
  - **Licensing** information
  - **Process** information: IDs of the processes adopted for ingestion, quality improvement, mapping, indexing,.. ; date and time of ingestion, update, review, ...;

  When a problem is detected, we have the information to understand when and how the problem has been included
- **Including basic ontologies as:**
  - *DC: Dublin core, standard metadata*
  - *OTN: Ontology for Transport Network*
  - *FOAF: for the description of the relations among people or groups*
  - *Schema.org: for a description of people and organizations*
  - *wgs84_pos: for latitude and longitude, GPS info*
  - *OWL-Time: reasoning on time, time intervals*
  - *GoodRelations: commercial activities models*

P. Bellini, M. Benigni, R. Billero, P. Nesi and N. Rauch, "Km4City Ontology Building vs Data Harvesting and Cleaning for Smart-city Services", International Journal of Visual Language and Computing, Elsevier, http://dx.doi.org/10.1016/j.jvlc.2014.10.023

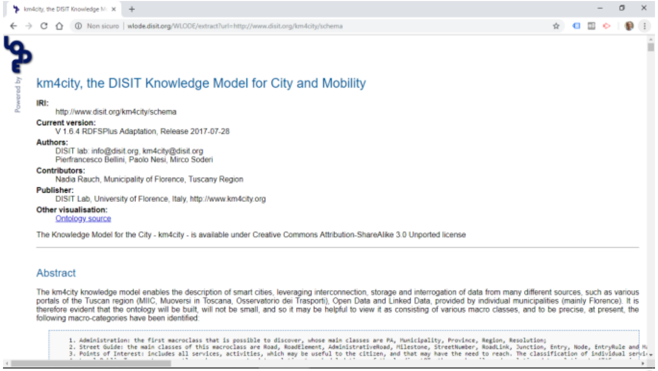
Smart-city Ontology km4city



84 Classes
93 ObjectProperties
103 DataProperties

LOV

- km4city ontology is a LOV
- http://www.disit.org/km4city/schema



km4city, the DISIT Knowledge M ×
Non sicuro | wlode.disit.org/WLODE/extract?url=http://www.disit.org/km4city/schema
Powered by

km4city, the DISIT Knowledge Model for City and Mobility

IRI:
http://www.disit.org/km4city/schema

Current version:
V 1.6.4 RDFSPlus Adaptation, Release 2017-07-28

Authors:
DISIT lab: info@disit.org, km4city@disit.org
Pierfrancesco Bellini, Paolo Nesi, Mirco Soderi

Contributors:
Nadia Rauch, Municipality of Florence, Tuscany Region

Publisher:
DISIT Lab, University of Florence, Italy, http://www.km4city.org

Other visualisation:
Ontology source

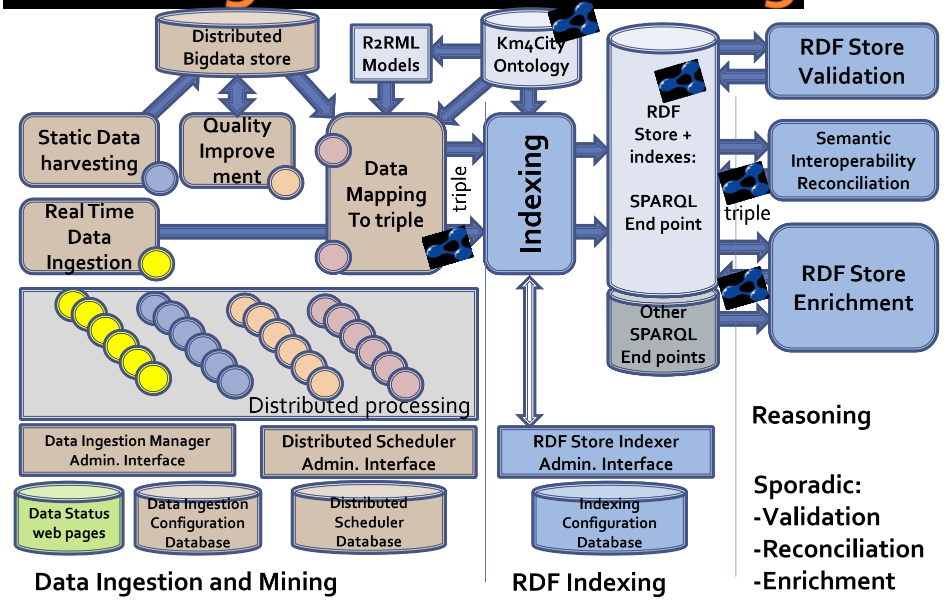
The Knowledge Model for the City - km4city - is available under Creative Commons Attribution-ShareAlike 3.0 Unported license

Abstract

The km4city knowledge model enables the description of smart cities, leveraging interconnection, storage and interrogation of data from many different sources, such as various portals of the Tuscan region (MIIC, Muoversi in Toscana, Osservatorio dei Trasporti), Open Data and Linked Data, provided by individual municipalities (mainly Florence). It is therefore evident that the ontology will be built, will not be small, and so it may be helpful to view it as consisting of various macro classes, and to be precise, at present, the following macro-categories have been identified:

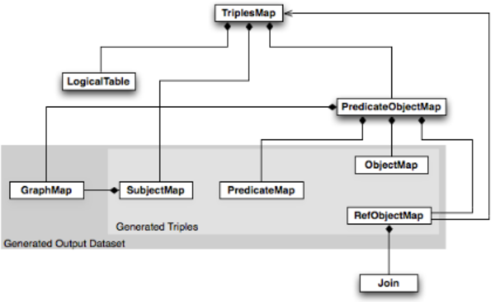
```
1. Administration: the first macroclass that is possible to discover, whose main classes are PA, Municipality, Province, Region, Resolution;
2. Street Guide: the main classes of this macroclass are Road, RoadElement, AdministrativeRoad, Milestone, StreetNumber, RoadLink, Junction, Entry, Node, EntryRule and Ma
3. Points of Interest: includes all services, activities, which may be useful to the citizen, and that may have the need to reach. The classification of individual servi
```

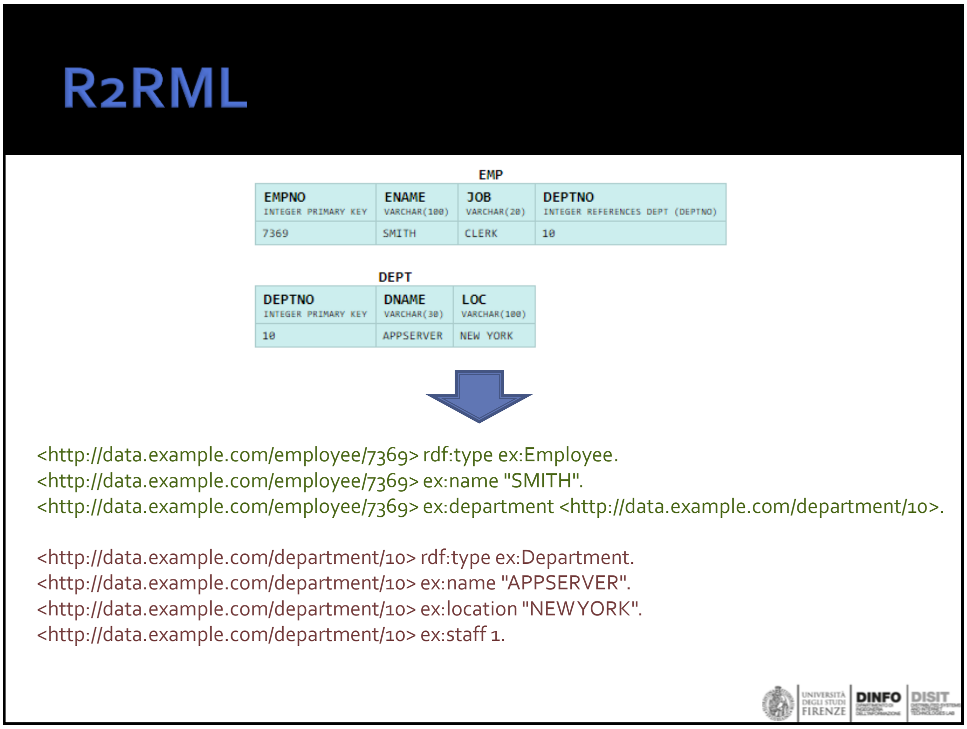
Data Ingestion and Mining



W3C R2RML

- **W3C R2RML: RDB to RDF Mapping Language**
- is a standard for the mapping of relational tables to RDF





R2RML

EMP

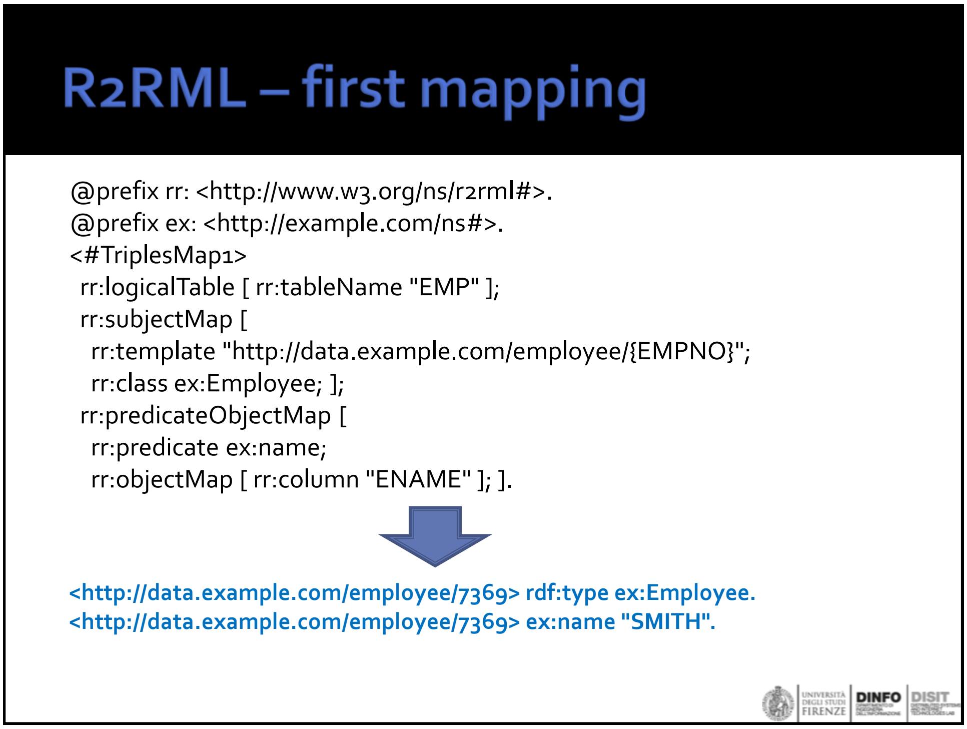
EMPNO INTEGER PRIMARY KEY	ENAME VARCHAR(100)	JOB VARCHAR(20)	DEPTNO INTEGER REFERENCES DEPT (DEPTNO)
7369	SMITH	CLERK	10

DEPT

DEPTNO INTEGER PRIMARY KEY	DNAME VARCHAR(30)	LOC VARCHAR(100)
10	APPSERVER	NEW YORK

<http://data.example.com/employee/7369> rdf:type ex:Employee.
<http://data.example.com/employee/7369> ex:name "SMITH".
<http://data.example.com/employee/7369> ex:department <http://data.example.com/department/10>.

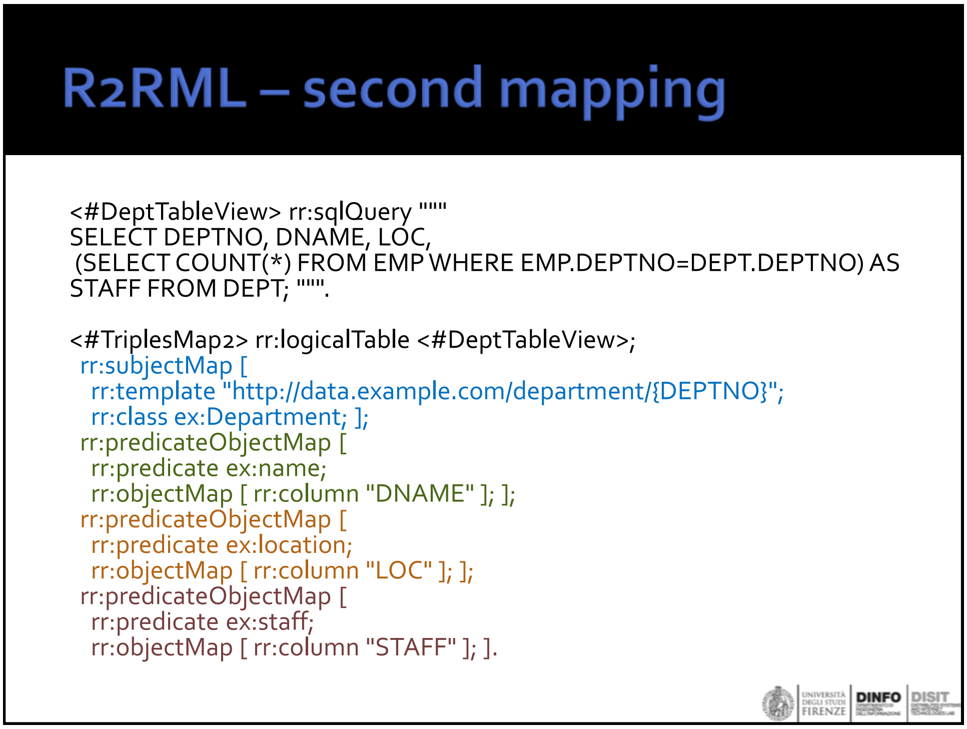
<http://data.example.com/department/10> rdf:type ex:Department.
<http://data.example.com/department/10> ex:name "APPSERVER".
<http://data.example.com/department/10> ex:location "NEWYORK".
<http://data.example.com/department/10> ex:staff 1.



R2RML – first mapping

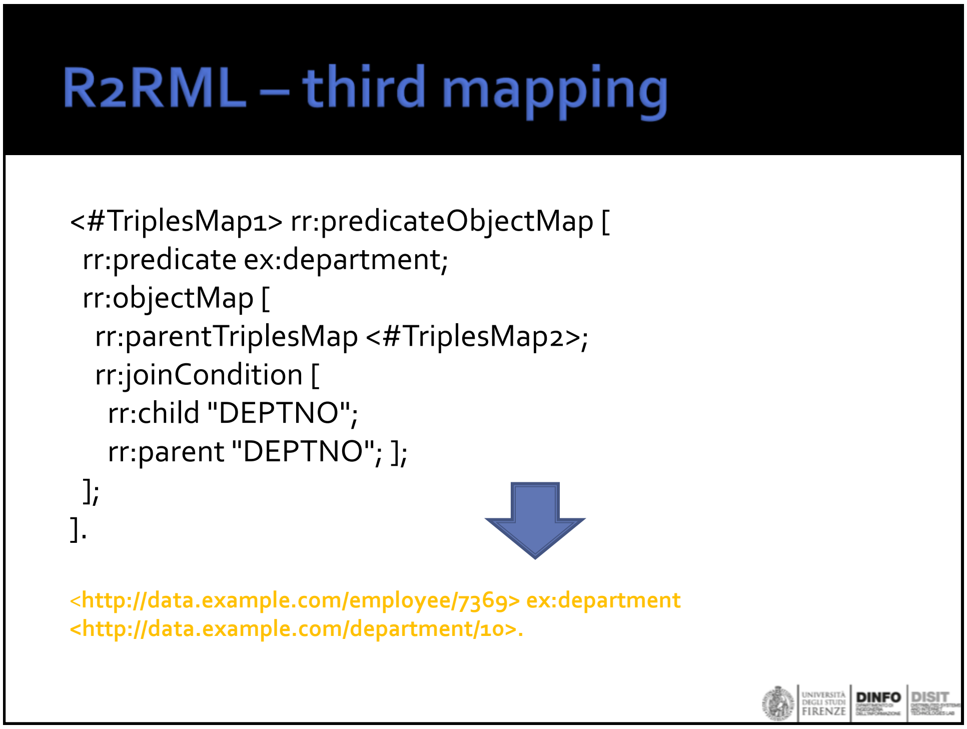
```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://example.com/ns#>.
<#TriplesMap1>
 rr:logicalTable [ rr:tableName "EMP" ];
 rr:subjectMap [
  rr:template "http://data.example.com/employee/{EMPNO}";
  rr:class ex:Employee; ];
 rr:predicateObjectMap [
  rr:predicate ex:name;
  rr:objectMap [ rr:column "ENAME" ]; ].
```

<http://data.example.com/employee/7369> rdf:type ex:Employee.
<http://data.example.com/employee/7369> ex:name "SMITH".



R2RML – second mapping

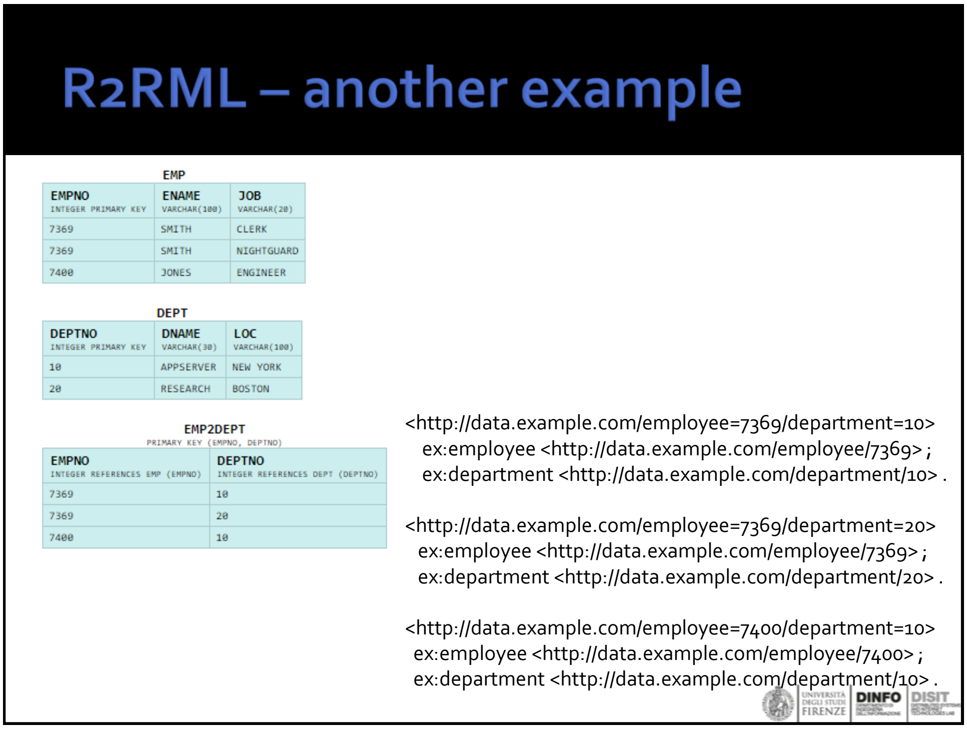
```
<#DeptTableView> rr:sqlQuery """
SELECT DEPTNO, DNAME, LOC,
 (SELECT COUNT(*) FROM EMP WHERE EMP.DEPTNO=DEPT.DEPTNO) AS
STAFF FROM DEPT; """.

<#TriplesMap2> rr:logicalTable <#DeptTableView>;
 rr:subjectMap [
  rr:template "http://data.example.com/department/{DEPTNO}";
  rr:class ex:Department; ];
 rr:predicateObjectMap [
  rr:predicate ex:name;
  rr:objectMap [ rr:column "DNAME" ]; ];
 rr:predicateObjectMap [
  rr:predicate ex:location;
  rr:objectMap [ rr:column "LOC" ]; ];
 rr:predicateObjectMap [
  rr:predicate ex:staff;
  rr:objectMap [ rr:column "STAFF" ]; ].
```



R2RML – third mapping

```
<#TriplesMap1> rr:predicateObjectMap [
 rr:predicate ex:department;
 rr:objectMap [
  rr:parentTriplesMap <#TriplesMap2>;
  rr:joinCondition [
   rr:child "DEPTNO";
   rr:parent "DEPTNO"; ];
 ];
].
```

```
<http://data.example.com/employee/7369> ex:department
<http://data.example.com/department/10>.
```



R2RML – another example

EMP

EMPNO	ENAME	JOB
INTEGER PRIMARY KEY	VARCHAR(100)	VARCHAR(20)
7369	SMITH	CLERK
7369	SMITH	NIGHTGUARD
7400	JONES	ENGINEER

DEPT

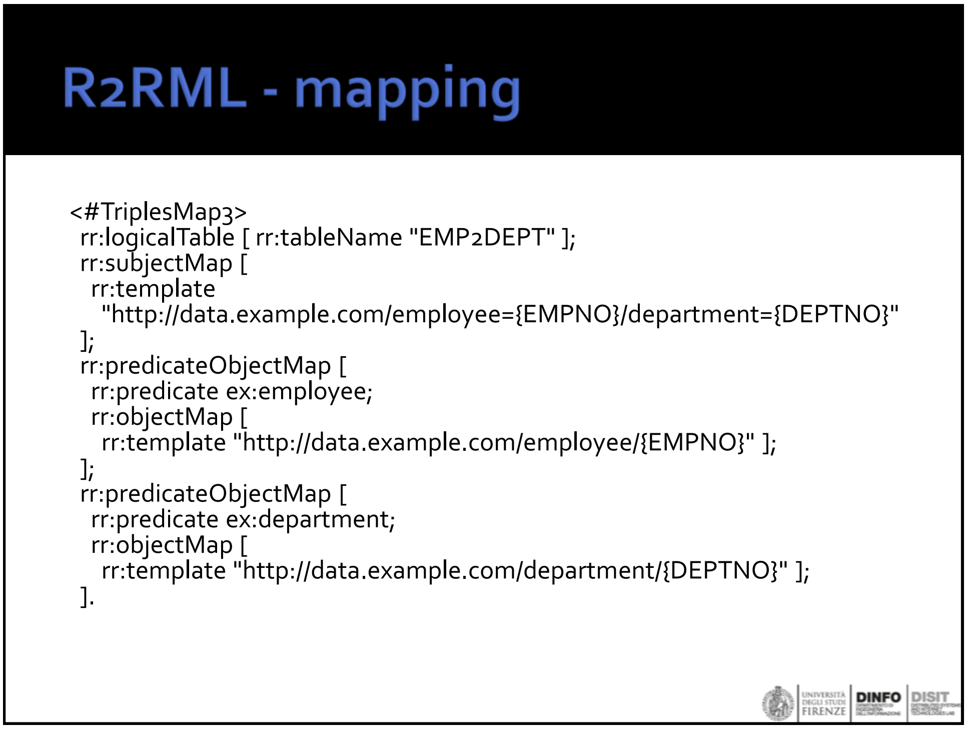
DEPTNO	DNAME	LOC
INTEGER PRIMARY KEY	VARCHAR(30)	VARCHAR(100)
10	APPSERVER	NEW YORK
20	RESEARCH	BOSTON

EMP2DEPT

PRIMARY KEY (EMPNO, DEPTNO)

EMPNO	DEPTNO
INTEGER REFERENCES EMP (EMPNO)	INTEGER REFERENCES DEPT (DEPTNO)
7369	10
7369	20
7400	10

```
<http://data.example.com/employee=7369/department=10>
  ex:employee <http://data.example.com/employee/7369> ;
  ex:department <http://data.example.com/department/10> .

<http://data.example.com/employee=7369/department=20>
  ex:employee <http://data.example.com/employee/7369> ;
  ex:department <http://data.example.com/department/20> .

<http://data.example.com/employee=7400/department=10>
  ex:employee <http://data.example.com/employee/7400> ;
  ex:department <http://data.example.com/department/10> .
```



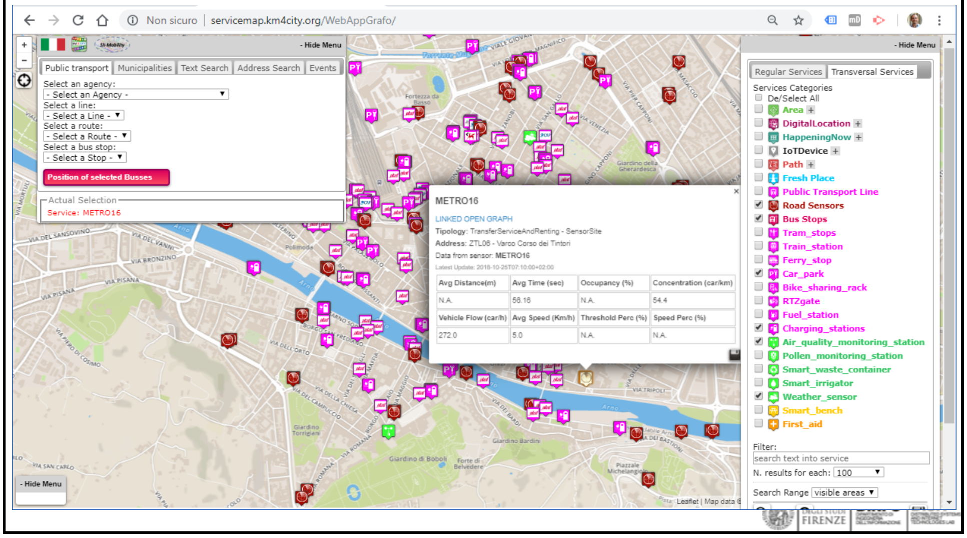
R2RML - mapping

```
<#TriplesMap3>
  rr:logicalTable [ rr:tableName "EMP2DEPT" ];
  rr:subjectMap [
    rr:template
      "http://data.example.com/employee={EMPNO}/department={DEPTNO}"
  ];
  rr:predicateObjectMap [
    rr:predicate ex:employee;
    rr:objectMap [
      rr:template "http://data.example.com/employee/{EMPNO}" ];
  ];
  rr:predicateObjectMap [
    rr:predicate ex:department;
    rr:objectMap [
      rr:template "http://data.example.com/department/{DEPTNO}" ];
  ].
```

R2RML – another mapping

```
<#TriplesMap3>
 rr:logicalTable [ rr:tableName "EMP2DEPT" ];
 rr:subjectMap [
  rr:template "http://data.example.com/employee/{EMPNO}";
 ];
 rr:predicateObjectMap [
  rr:predicate ex:department;
  rr:objectMap [
   rr:template
     "http://data.example.com/department/{DEPTNO}"
  ];
 ].
```

servicemap.km4city.org



Non sicuro | servicemap.km4city.org/WebAppGrafo/

Select an agency: - Select an Agency -
Select a line: - Select a Line -
Select a route: - Select a Route -
Select a bus stop: - Select a Stop -

Actual Selection
Service: METRO16

METRO16

LINKED OPEN GRAPH

Tipology: TransferServiceAndRenting - SensorSite

Address: ZTL06 - Varco Corso dei Tintori

Data from sensor: **METRO16**

Latest Update: 2018-10-25T07:10:00+02:00

Avg Distance(m)	Avg Time (sec)	Occupancy (%)	Concentration (car/km)
N.A.	56.16	N.A.	54.4
Vehicle Flow (car/h)	Avg Speed (Km/h)	Threshold Perc (%)	Speed Perc (%)
272.0	5.0	N.A.	N.A.

Services Categories
□ De/Select All
□ Area +
□ DigitalLocation +
□ HappeningNow +
□ IoTDevice +
□ Path +
□ Fresh Place
□ Public Transport Line
☑ Road Sensors
☑ Bus Stops
□ Tram_stops
□ Train_station
□ Ferry_stop
☑ Car_park
□ Bike_sharing_rack
□ RTZgate
□ Fuel_station
☑ Charging_stations
☑ Air_quality_monitoring_station
□ Pollen_monitoring_station
□ Smart_waste_container
□ Smart_irrigator
☑ Weather_sensor
□ Smart_bench
□ First_aid

Filter: search text into service
N. results for each: 100
Search Range visible areas