

Part III

Semantic Web Languages

RDF

RDF Data Model

- **Resources**
 - A resource is a thing you talk about (can reference)
 - Resources have URI's
 - RDF definitions are itself Resources (linkage)
- **Properties**
 - slots, defines relationship to other resources or atomic values
- **Statements**
 - “Resource has Property with Value”
 - (Values can be resources or atomic XML data)
- **Similar to Frame Systems**

R
D
F

A simple Example

- **Statement**

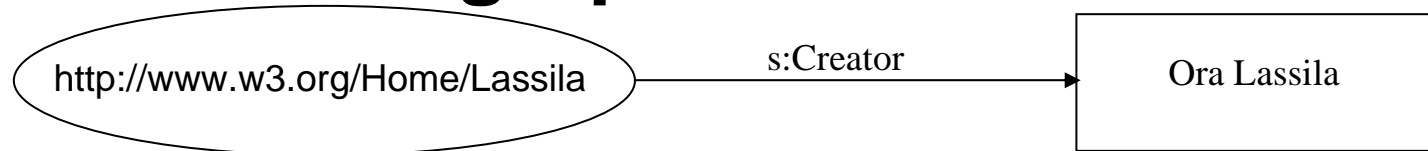
- “Ora Lassila is the creator of the resource
<http://www.w3.org/Home/Lassila>”

R
D
E

- **Structure**

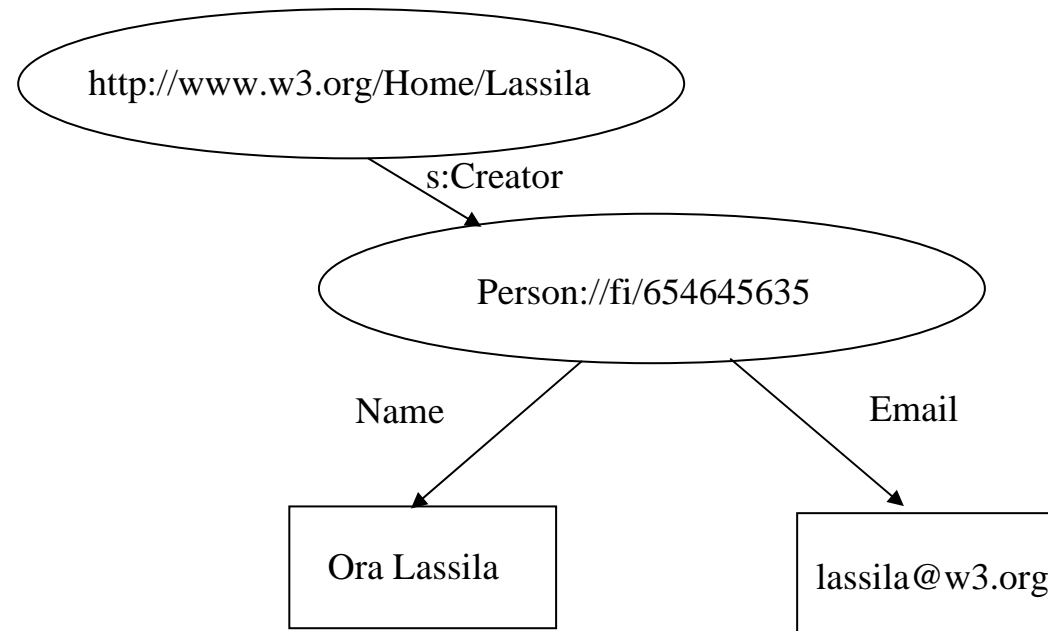
- Resource (subject)
<http://www.w3.org/Home/Lassila>
- Property (predicate) <http://www.schema.org/#Creator>
- Value (object) "Ora Lassila"

- **Directed graph**



Another Example

- To add properties to Creator, point through a intermediate Resource.



Collection Containers

- Multiple occurrences of the same PropertyType doesn't establish a relation between the values
 - The Millers own a boat, a bike, and a TV set
 - The Millers need (a car or a truck)
 - (Sarah and Bob) bought a new car

R
D
F

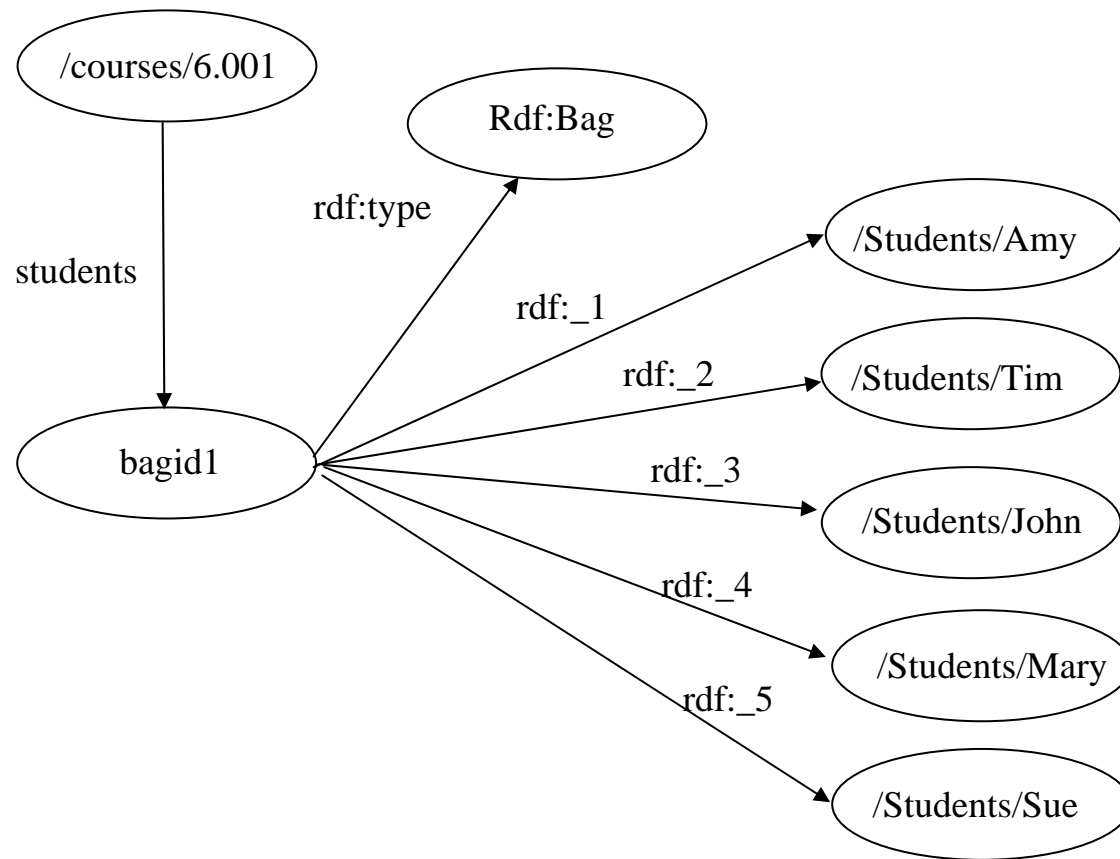
RDF defines three special Resources:

- **Bag** unordered values rdf:Bag
- **Sequence** ordered values rdf:Seq
- **Alternative** single value rdf:Alt
 - Core RDF does not enforce 'set' semantics amongst values

Example: Bag

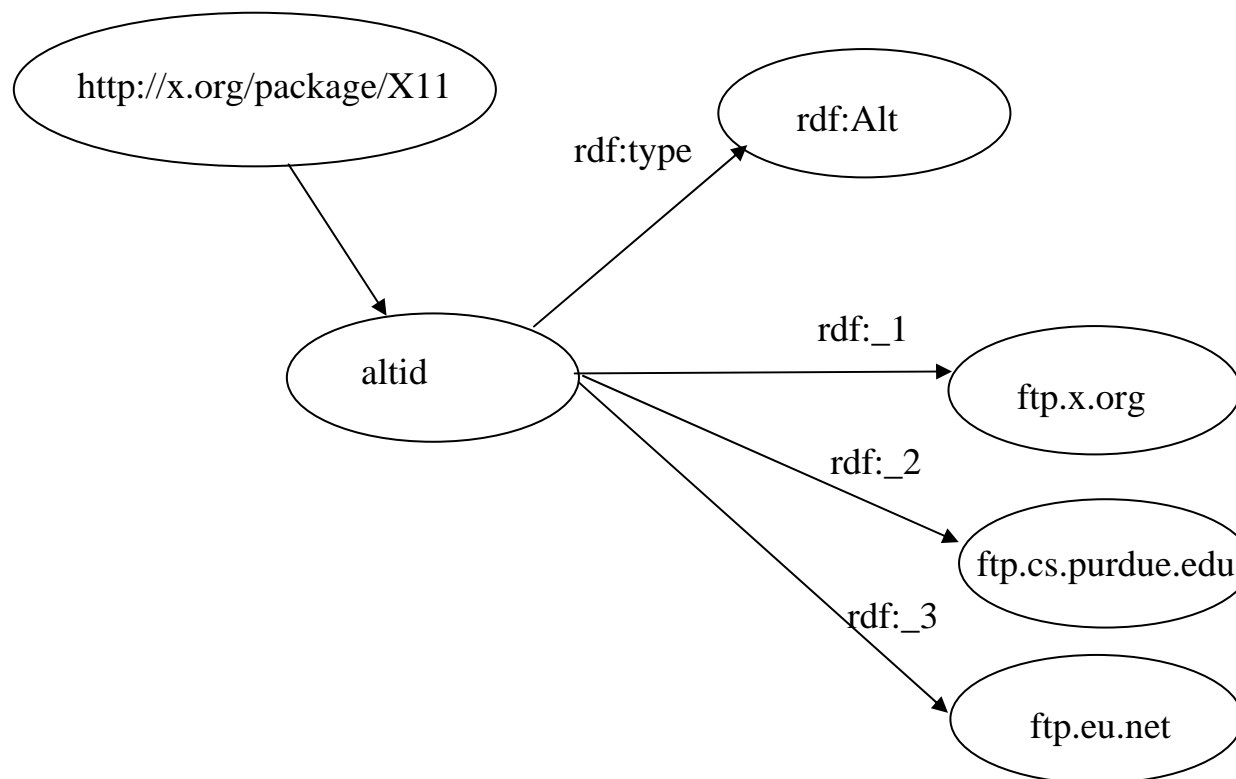
The students in course 6.001 are Amy, Tim, John, Mary, and Sue

FDR



Example: Alternative

- *The source code for X11 may be found at ftp.x.org, ftp.cs.purdue.edu, or ftp.eu.net*



Statements about Statements (Requirement 2: Dispute Statements)

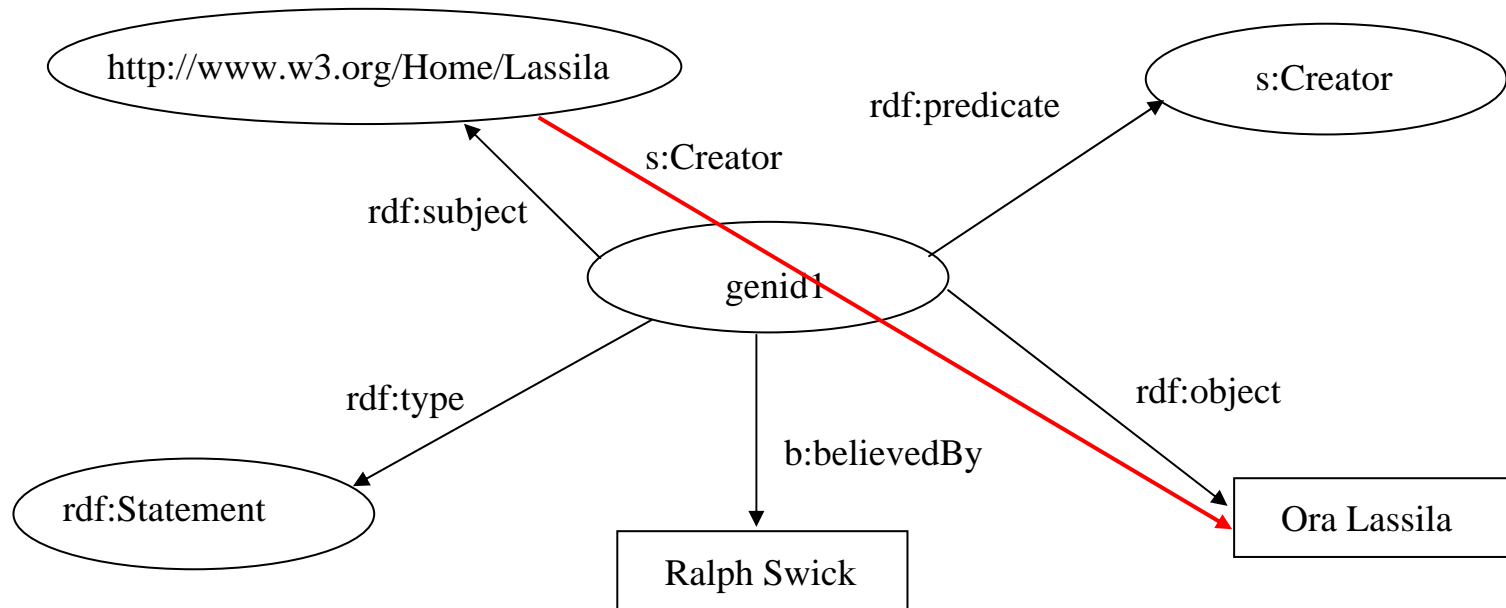
- Making statements about *statements* requires a process for transforming them into Resources

R
D
E

- **subject** the original referent
- **predicate** the original property type
- **object** the original value
- **type** rdf:Statement

Example: Reification

- *Ralph Swick believes that*
 - *the creator of the resource*
http://www.w3.org/Home/Lassila is Ora Lassila



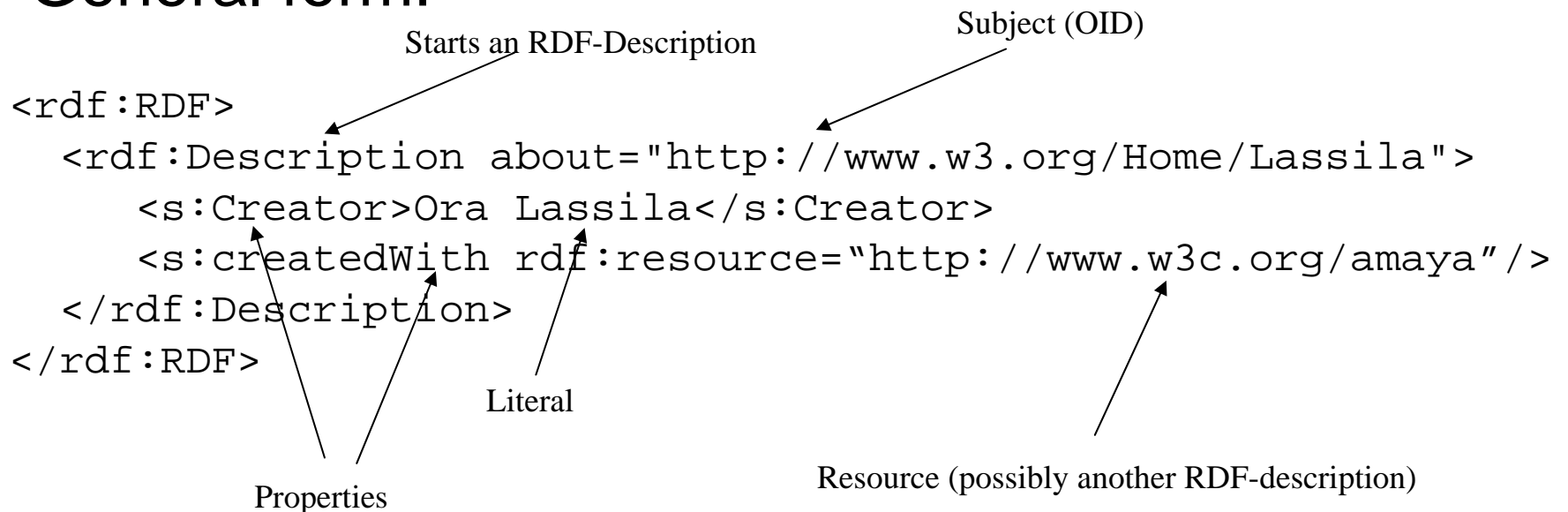
FD R

RDF Syntax I

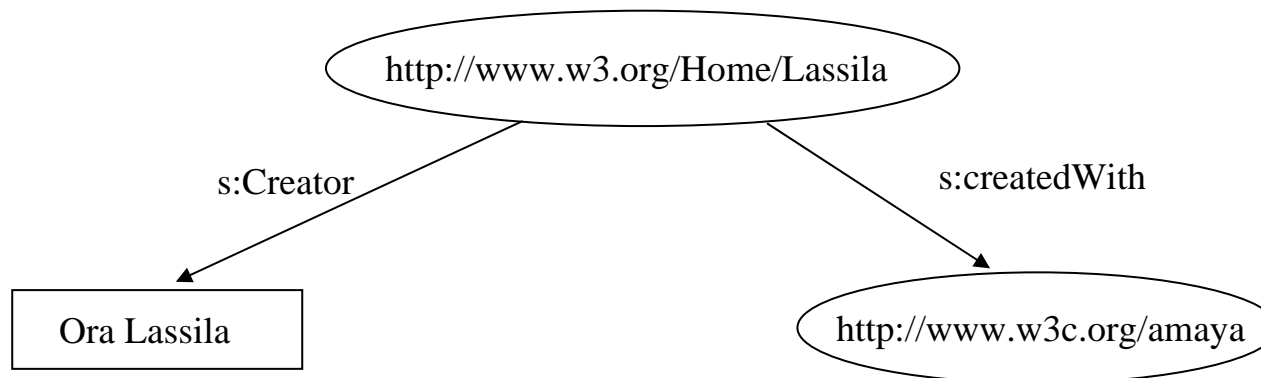
- Datamodel does not enforce particular syntax
- Specification suggests many different syntaxes based on XML



General form:



Resulting Graph



R
D
F

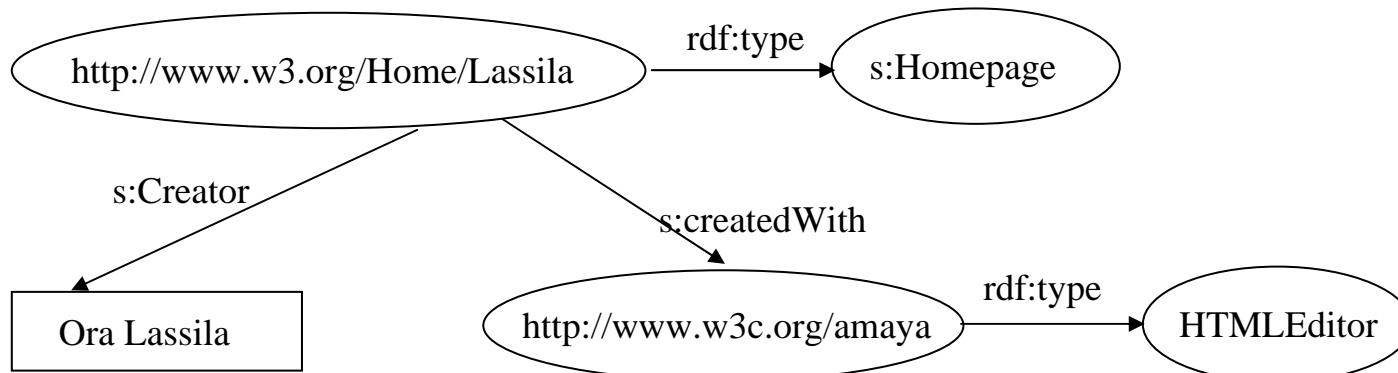
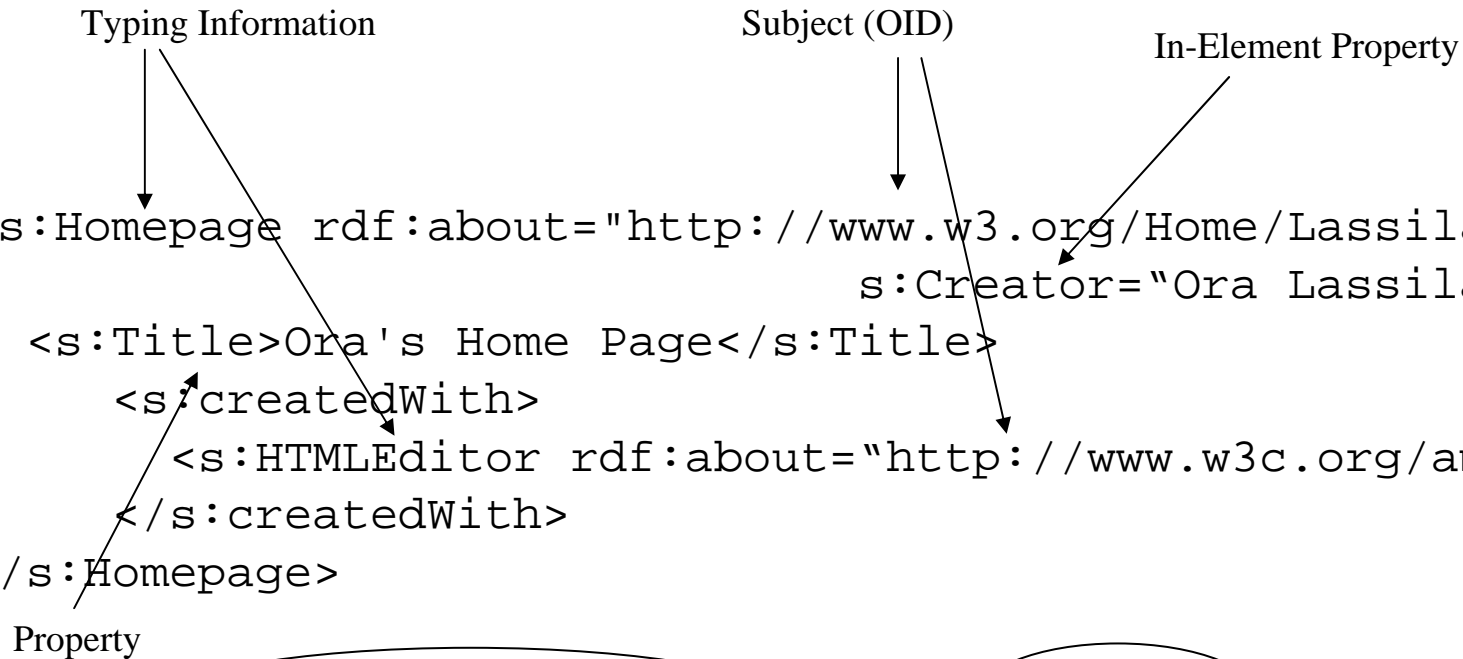
```
<rdf:RDF>  
  <rdf:Description about="http://www.w3.org/Home/Lassila">  
    <s:Creator>Ora Lassila</s:Creator>  
    <s:createdWith rdf:resource="http://www.w3c.org/amaya" />  
  </rdf:Description>  
</rdf:RDF>
```

RDF Syntax II: Syntactic Varieties

R
D
E

```

Typing Information
      |
      v
<s:Homepage rdf:about="http://www.w3.org/Home/Lassila"
           s:Creator="Ora Lassila"/>
<s>Title>Ora's Home Page</s>Title>
<s:createdWith>
  <s:HTMLEditor rdf:about="http://www.w3c.org/amaya"/>
</s:createdWith>
</s:Homepage>
    
```



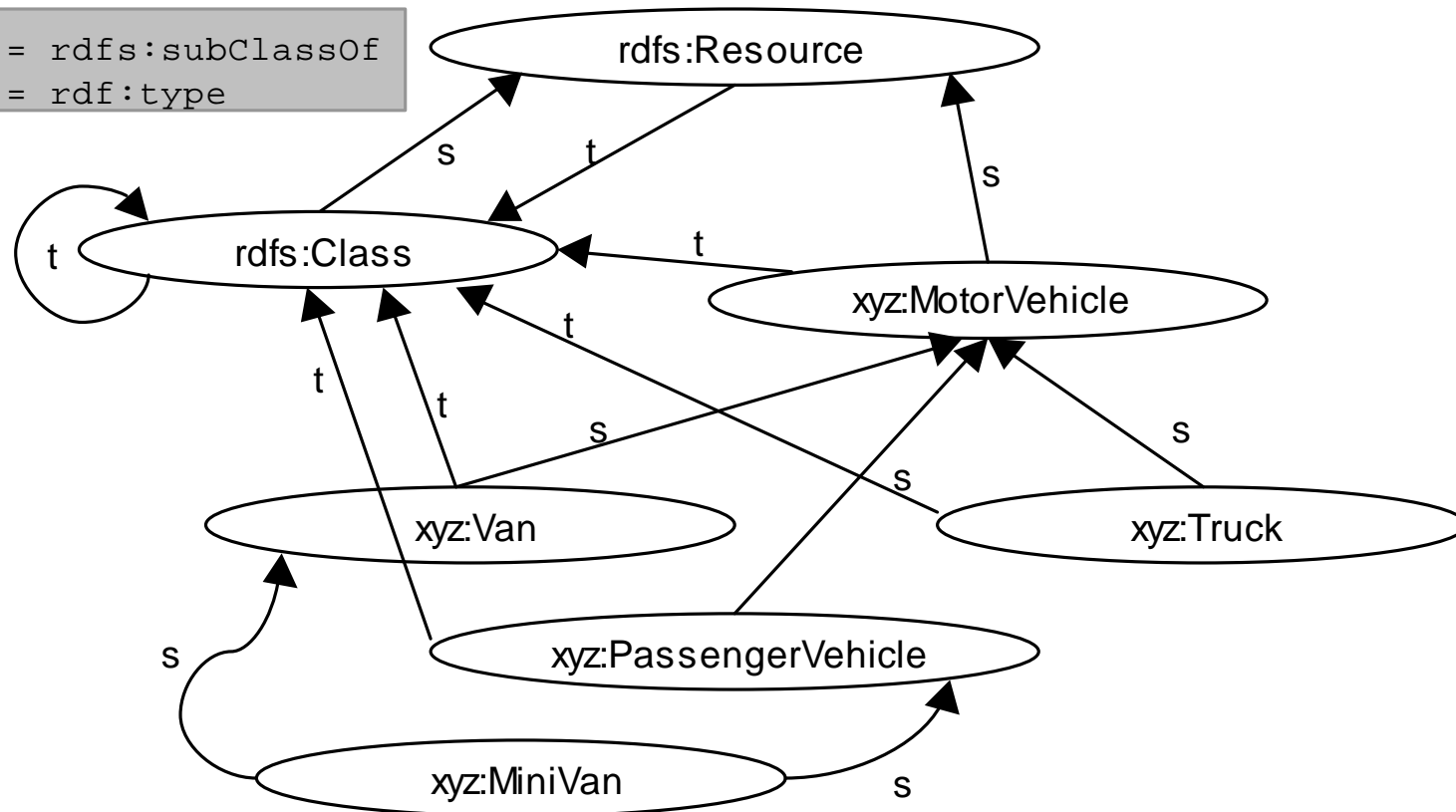
RDF Schema (RDFS)

- RDF just defines the datamodel
- Need for definition of vocabularies for the datamodel - an Ontology Language!
- RDF schemas are Web resources (and have URIs) and can be described using RDF

R
D
F

RDF-Schema: Example

s = rdfs:subClassOf
t = rdf:type



FB4

Rdfs:subclassOf

```
<rdfs:description about=„Xyz:Minivan“>  
  <rdfs:subclassOf about=„xyz:Van“/>  
</rdfs:description>  
<rdfs:description about=„myvan“>  
  <rdf:type about=„xyz:MiniVan“/>  
</rdfs:description>
```

Predicate Logic Consequences:

Forall X: type(X,MiniVan) -> type(X, Van).

Forall X: subclassOf(X,MiniVan) -> subclassOf(X, Van).

Rdf:property

```
<rdf:description about=„possesses“>
  <rdf:type about=„...property“/>
  <rdfs:domain about=„person“/>
  <rdfs:range about=„vehicle“/>
</rdf:description>
<rdf:description about=„peter“>
  <possesses>petersminivan</possesses>
</rdf:description>
```

Predicate Logic Consequences:

Forall X,Y: possesses (X,Y) -> (type(X,person) & type(Y,vehicle)).

OWL

OWL

- OWL1.0 (acronym for Web Ontology Language) is a W3C recommendation
- OWL stems from a family of logics, called „description logics“

Description Logics (Terminological Logics, DLs)

- Fragments of FOL
- Most often decidable
- Moderately expressive
- Stem from semantic networks
- W3C Standard OWL DL corresponds to SHOIN(D)

DLs – general structure

- DLs are a **Family** of logic-based formalism for knowledge representation
- Special language characterized by:
 - Constructors to define complex concepts and roles based on simpler ones.
 - Set of axiom to express facts using concepts, roles and individuals.
- ALC is the smallest DL, which is propositionally closed:
 - \wedge, \vee, \neg are constructors, noted by \sqcap, \sqcup, \neg .
 - Quantors define how roles are to be interpreted:

Man $\sqcap \exists \text{hasChild.Female} \sqcap \exists \text{hasChild.Male}$
 $\sqcap \forall \text{hasChild.}(\text{Rich} \sqcup \text{Happy})$

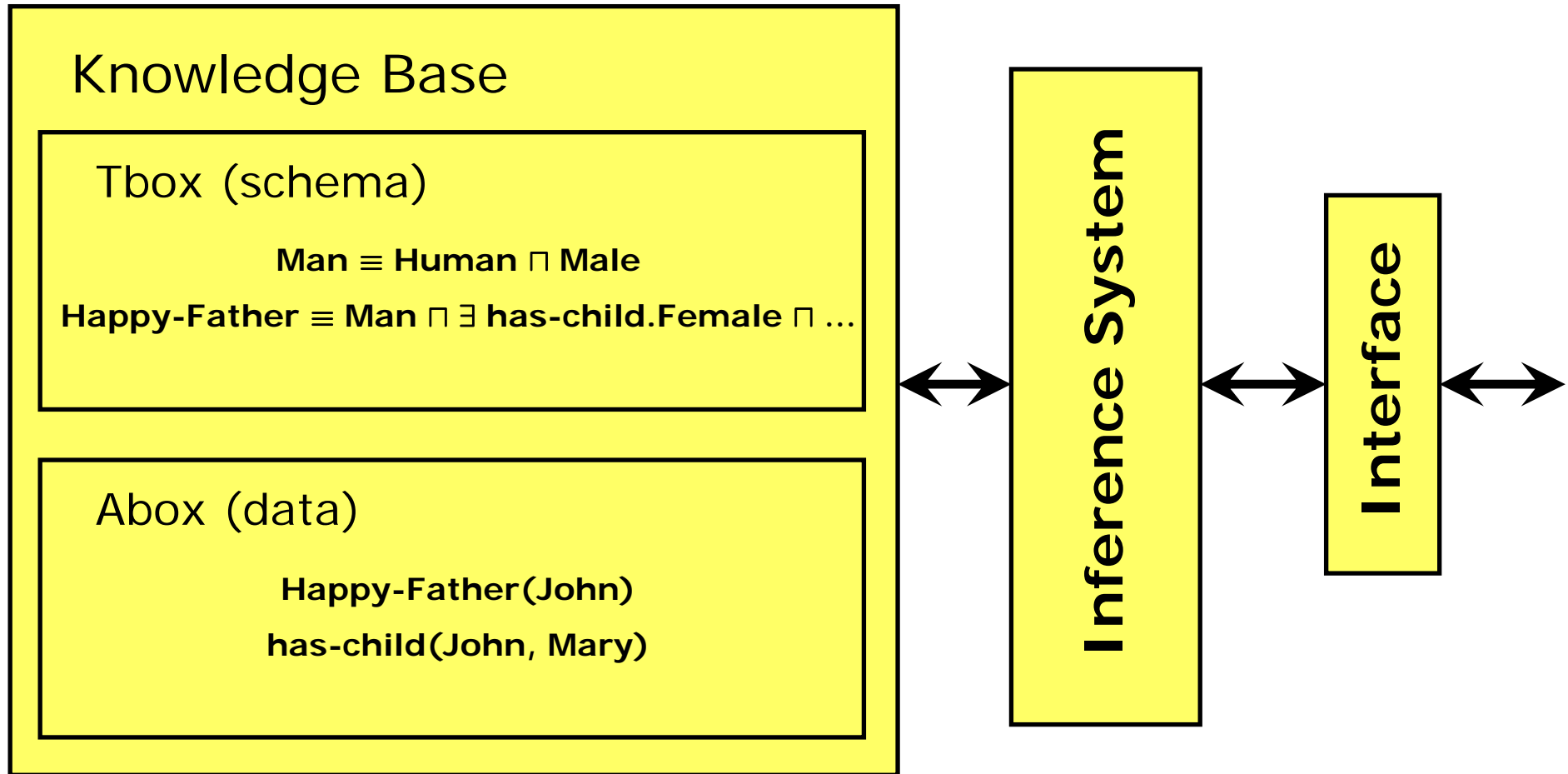
Further DL concepts and role constructors

- Number restrictions (cardinality constraints) for roles:
 ≥ 3 hasChild, ≤ 1 hasMother
- Qualified number restrictions:
 ≥ 2 hasChild.Female, ≤ 1 hasParent.Male
- Nominals (definition by extension):
{Italy, France, Spain}
- Concrete domains (datatypes): hasAge.(≥ 21)
- Inverse roles: hasChild⁻¹ \equiv hasParent
- Transitive roles: hasAncestor* (descendant)
- Role composition: hasParent.hasBrother (uncle)

DL Knowledge Bases

- DL Knowledge Bases consist of two parts (in general):
 - TBox: Axioms, describing the structure of a modelled domain (conceptual schema):
 - $\text{HappyFather} \equiv \text{Man} \sqcap \exists \text{hasChild.Female} \sqcap \dots$
 - $\text{Elephant} \sqsubseteq \text{Animal} \sqcap \text{Large} \sqcap \text{Grey}$
 - $\text{transitive}(\text{hasAncestor})$
 - ABox: Axiome describing concrete situations (data, facts):
 - $\text{HappyFather}(\text{John})$
 - $\text{hasChild}(\text{John}, \text{Mary})$
- The distinction between TBox/ABox does not have a deep logical distinction
... but it is common useful modelling practice.

General DL Architecture



Knowledge modelling in OWL

Example ontology and conclusion from
<http://owl.man.ac.uk/2003/why/latest/#2>

- Also an example for OWL Abstract Syntax.

Namespace(a = <<http://cohse.semanticweb.org/ontologies/people#>>)

Ontology(
 ObjectProperty(a:drives)
 ObjectProperty(a:eaten_by)
 ObjectProperty(a:eats inverseOf(a:eaten_by) domain(a:animal))
 ...
 Class(a:adult partial annotation(rdfs:comment "Things that are adult."))
 Class(a:animal partial restriction(a:eats someValuesFrom (owl:Thing)))
 Class(a:animal_lover complete intersectionOf(restriction(a:has_pet
 minCardinality(3)) a:person))
 ...))

...)

Knowledge modelling: examples

Class(a:bus_driver complete intersectionOf(a:person
restriction(a:drives someValuesFrom (a:bus))))

bus_driver \equiv person \sqcap \exists drives.bus

Class(a:driver complete intersectionOf(a:person
restriction(a:drives someValuesFrom (a:vehicle))))

driver \equiv person \sqcap \exists drives.vehicle

Class(a:bus partial a:vehicle)

bus \sqsubseteq vehicle

- A bus driver is a person that drives a bus.
- A bus is a vehicle.
- A bus driver drives a vehicle, so must be a driver.

The subclass is inferred due to subclasses being used in existential quantification.

Knowledge modelling: examples

Class(a:driver complete intersectionOf(a:person restriction(a:drives someValuesFrom (a:vehicle)))) **driver \equiv person \sqcap \exists drives.vehicle**

Class(a:driver partial a:adult) **driver \sqsubseteq adult**

Class(a:grownup complete intersectionOf(a:adult a:person)) **grownup \equiv adult \sqcap person**

- Drivers are defined as persons that drive cars (complete definition)
- We also know that drivers are adults (partial definition)
- So all drivers must be adult persons (e.g. grownups)

An example of axioms being used to assert additional necessary information about a class. We do not need to know that a driver is an adult in order to recognize one, but once we have recognized a driver, we know that they must be adult.

$\exists \text{partof. animal} \sqcup \text{animal} \not\equiv \text{plant} \sqcup \exists \text{partof. plant}$

Knowledge modelling: Examples

Class(a:cow partial a:vegetarian)

DisjointClasses(unionOf(restriction(a:part_of someValuesFrom (a:animal)) a:animal) unionOf(a:plant restriction(a:part_of someValuesFrom (a:plant))))

Class(a:vegetarian complete intersectionOf(restriction(a:eats allValuesFrom (complementOf(restriction(a:part_of someValuesFrom (a:animal)))))) restriction(a:eats allValuesFrom (complementOf(a:animal))) a:animal))

Class(a:mad_cow complete intersectionOf(a:cow restriction(a:eats someValuesFrom (intersectionOf(restriction(a:part_of someValuesFrom (a:sheep)) a:brain))))))

Class(a:sheep partial a:animal restriction(a:eats allValuesFrom (a:grass)))

- Cows are naturally vegetarians
- A mad cow is one that has been eating sheeps brains
- Sheep are animals

Thus a mad cow has been eating part of an animal, which is inconsistent with the definition of a vegetarian

Knowledge modelling: Example

```
Individual(a:Walt type(a:person) value(a:has_pet a:Huey)  
value(a:has_pet a:Louie) value(a:has_pet a:Dewey))
```

```
Individual(a:Huey type(a:duck))
```

```
Individual(a:Dewey type(a:duck))
```

```
Individual(a:Louie type(a:duck))
```

```
DifferentIndividuals(a:Huey a:Dewey a:Louie)
```

```
Class(a:animal_lover complete intersectionOf(a:person  
restriction(a:has_pet minCardinality(3))))
```

```
ObjectProperty(a:has_pet domain(a:person) range(a:animal))
```

- Walt has pets Huey, Dewey and Louie.
- Huey, Dewey and Louie are all distinct individuals.
- Walt has at least three pets and is thus an animal lover.

Note that in this case, we don't actually need to include person in the definition of animal lover (as the domain restriction will allow us to draw this inference).

Knowledge modelling: Some Research Challenges

- Concluding with
 - uncertainty (fuzzy, probabilistic)
 - Inkonsistencies (paraconsistent)
 - Rules
 - Further AI-Paradigms (nonmonotonic reasoning, preferences ...)
- Maintenance (updates, infrastructure, etc)
- Scalability of reasoning
- ...