OBERÖSTERREICH

University of Applied Sciences

Web-Semantik-Technologien

Vorlesung WS 2019/20

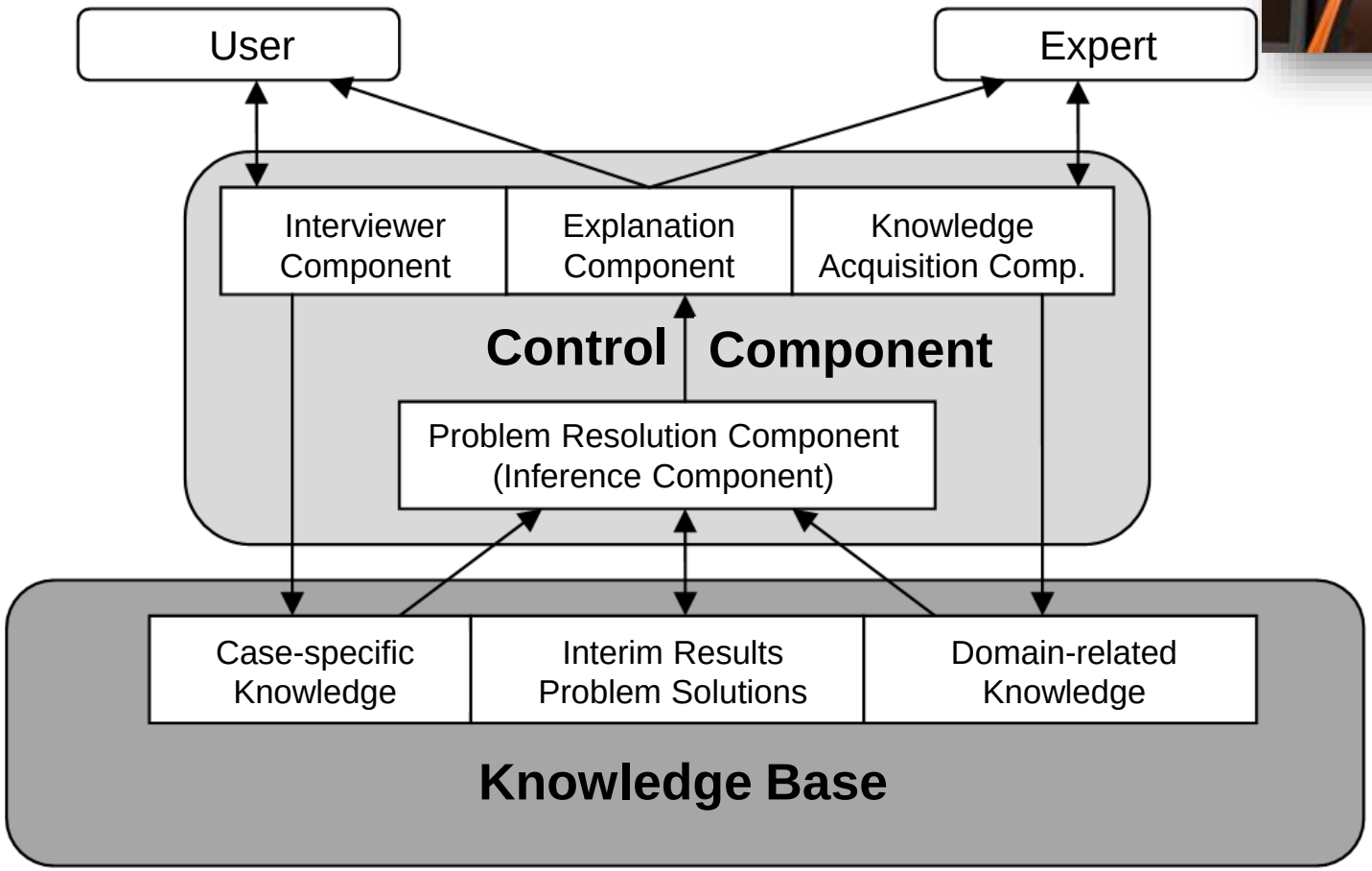
thomas.kern@fh-hagenberg.at

Semantic Technologies

Knowledege Engineering Process

Knoweldge Engineering

Goal: Knowledge-based System (KBS)



Heinsohn, J. & Socher-Ambrosius, R.: Wissensverarbeitung (1999).

KBS Building Blocks

- **Knowledge (Knowledge Base)**
  - Basis of the KBS; Knowledge is collected in the KB and shown as facts and rules.
  - includes additional knowledge to control the inference engine
- **Inference Engine (Knowledge Processing)**
  - "Brain" of the KBS; unlike conventional databases "new" knowledge is generated by inference.
- **User Interface (Dialog Components)**
  - Fill the KB and knowledge query; explanation of inference results; updating and expansion of knowledge.

Knowledge Engineering

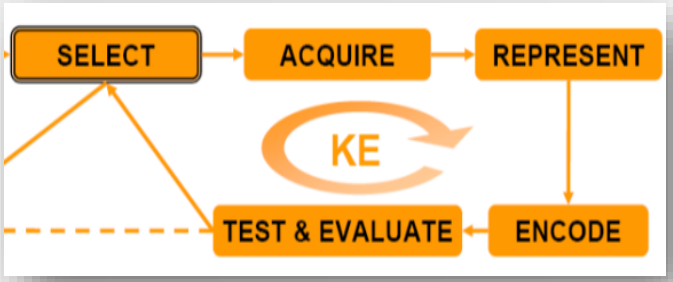
Tasks

- **Knowledge Acquisition**
  - Capturing and organizing of implicit (tacit) and explicit knowledge
  - Acquisition of (expert) knowledge is the bottleneck of KE!
- **Formalization**
  - Computer-processable modeling and representation of knowledge within a knowledgebase
  - Bridging the “AI Gap” aka “Semantic Gap”
- **Knowledge Processing**
  - for problem solving
  - e.g. with inference (reasoners) or expert systems
- **Knowledge Representation (visualization)**

Knowledge Engineering Phases

- Task identification - comparable to SE requirements analysis
- Knowledge acquisition
- Specification of the vocabulary - ontology
- Formalization - axioms and instances
- Inference procedure - Evaluation and test
- Implementation and maintenance

Phase 1: Problem Description



- Task Identification
  - Domain specification, staking the area, electing the experts
  - Problem description and requirement analysis
    requirements engineering (SE) vs. competency questions (KE)
    - Conventional SE requirements engineering is problematic for KE. In KE requirements are initially only difficult to analyze but highly iterative!
    - RE does not provide process models that allow access to the complex and expert domain knowledge
    - Moreover: in knowledge-intensive domains it is not clear to users what they can expect from the software system.

KE Application Example Phase 1

- Problem description „Wine & Food“

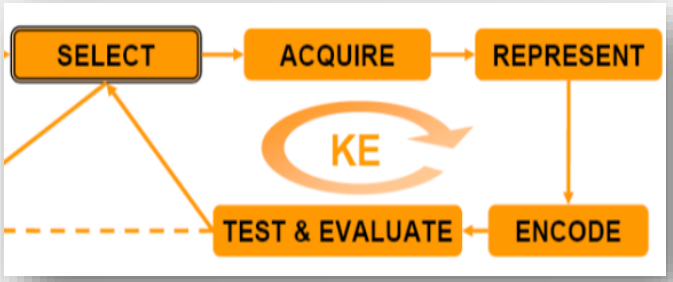
Application for determining appropriate combinations of wine and food.

Which wine goes with a determined food (e.g.: fish)?

KE Application Example Phase 1

- Wine & Food Competency Questions
  - *Which wine characteristics should be considered when choosing a wine?*
  - *Is Bordeaux a red or white wine?*
  - *Fits Cabernet Sauvignon well with seafood?*
  - *What would be the best choice of wine for grilled meat?*
  - *What characteristics of a wine affect its suitability for a specific product?*
  - *Does the bouquet of a wine change with its vintage?*
  - *Which was a good vintage for Napa Zinfandel?*

Phase 2: Knowledge Acquisition



- **Gathering knowledge**
  - primarily on the cognitive level (based on findings of the experts)
- **Direct**
  - Expert operates knowledge acquisition himself
- **Indirect**
  - Knowledge engineer uses several KM methods of knowledge sharing and development together with experts
- **Automated**
  - Machine learning methods are used
- **Model-based**
  - KM models (e.g .: CommonKADS - Knowledge Acquisition and Documentation Structuring; https://commonkads.org/)
- **Documentation!**

Caution:
Knowledge acquisition is the bottleneck of KE.
It is crucial for the success of the system!

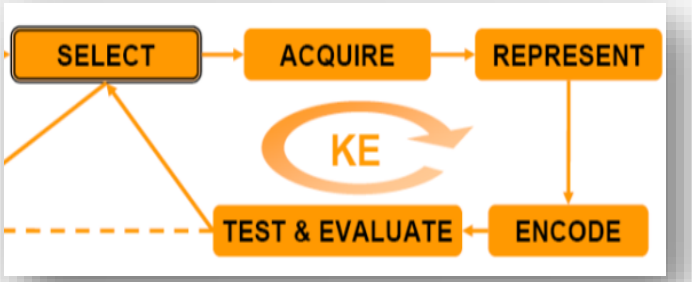
KE Application Example Phase 2

- Knowledge Acquisition "Wine & Food":
- List of important concepts
  - Brainstorming
  - Concepts (terms) and characteristics
  - Simple list without regard to overlaps, relationships, types of properties

KE Application Example Phase 2

- wine
- grape type
- winery
- color
- taste
- sugar content
- meat
- fish
- white wine
- …

Phase 3: Conceptualization

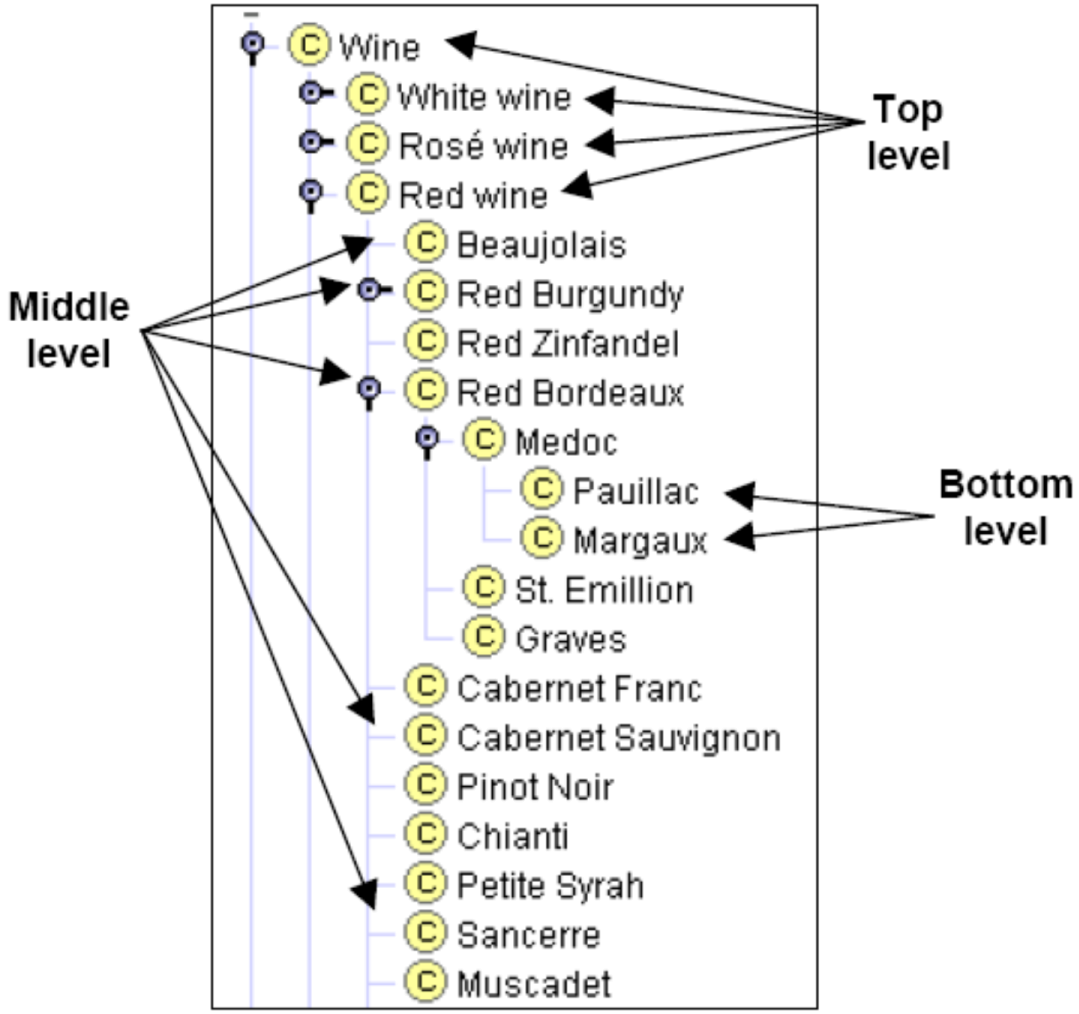


- Specifying the vocabulary (ontology)
  - Transfer of knowledge from the cognitive level to the representation level
  - Consistent representation of knowledge for better manageability
  - Ontology Engineering (initial formalization, conceptualization)

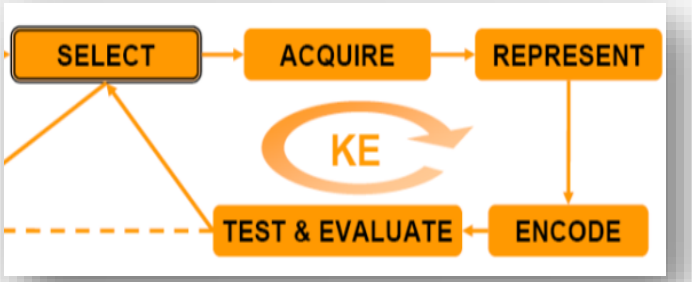
KE Application Example Phase 3

- Vocabulary "Wine & Food"
  - Definition of the classes and their hierarchy (taxonomy)

KE Application Example Phase 3



Phase 4: Formalization



- Axioms and instances
  - Knowledge formalization (second formalization, encoding)
  - Revision of the vocabulary
  - Specification of conditions and constraints
  - Selection of an appropriate representation formalism in terms of syntax, semantics and reasoning possibilities of knowledge

    Axioms: Inheritance, disjunction, etc.?
    Constraints: cardinalities, transitivity, symmetry etc.

KE Application Example Phase 4

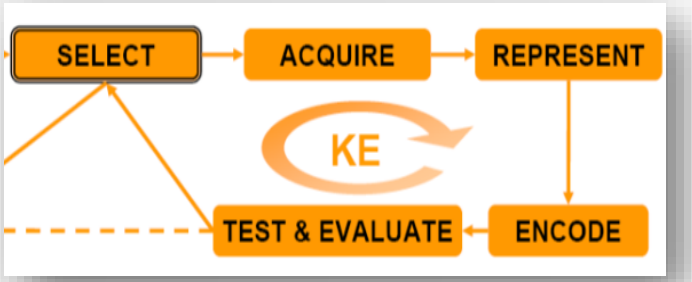
- Formalization of the vocabulary for “Wine & Food”
  - Definition of relationships / properties (slots) and their allowed values (facets)
  - Concretization of concepts

KE Application Example Phase 4

- Drink
  - Wine
    - White wine
    - Red wine
    - Rose wine
    - Dessert wine
  - Soft drink
- Meal course
- Wine grape

Name	Cardinality	Type	Other Facets
body	single	Symbol	allowed-values={FULL,MEDIUM,LIGHT}
color	single	Symbol	allowed-values={RED,ROS□,WHITE}
flavor	single	Symbol	allowed-values={DELICATE,MODERATE,STRONG}
grape	multiple (0:4)	Instance of Wine grape	
maker	single	Instance of Winery	inverse-slot=produces
name	single	String	
sugar	single	Symbol	allowed-values={DRY,SWEET,OFF-DRY}

Phase 5: Evaluation



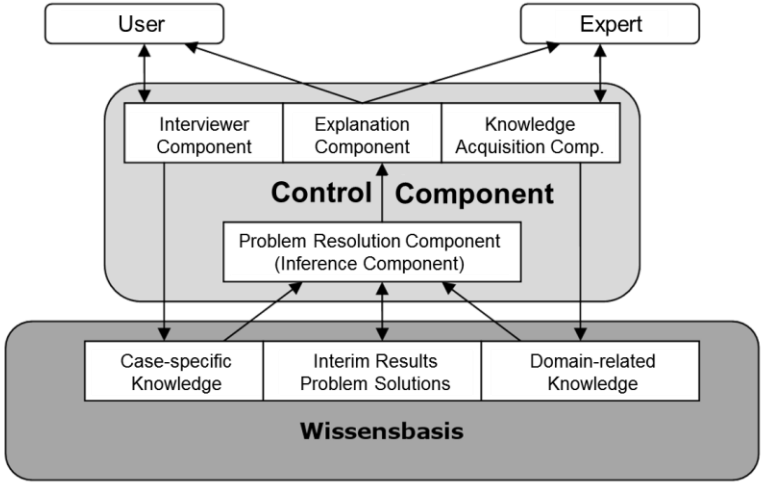
- Inference procedure - Evaluation and Test
  - **Validation of the acquired knowledge**
    - Completeness
    - Reliability
  - **Iteration if necessary**
    - Go back to Step 2) Knowledge Acquisition

KE Application Example Phase 5

- Evaluation of the “Wine & Food” vocabulary
  - Reasoning mechanisms to verify consistency
  - Answering the competency questions

Phase 6: Implementation and Maintenance

- **Implementation of the WB in appropriate formalisms**
- **Implementation of the user interface**
  - Query component
  - Explanation and visualization component
  - Manipulation component



Heinsohn, J. & Socher-Ambrosius, R.: Wissensverarbeitung (1999).

OBERÖSTERREICH

University of Applied Sciences

Web-Semantik-Technologien

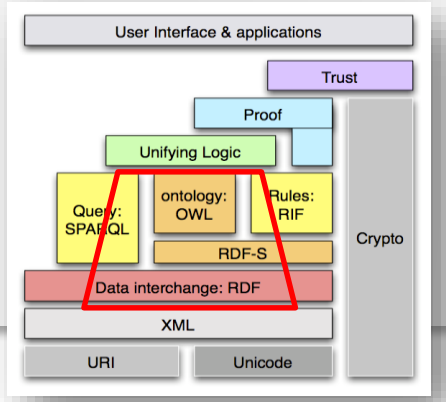
Vorlesung WS 2017/18

thomas.kern@fh-hagenberg.at

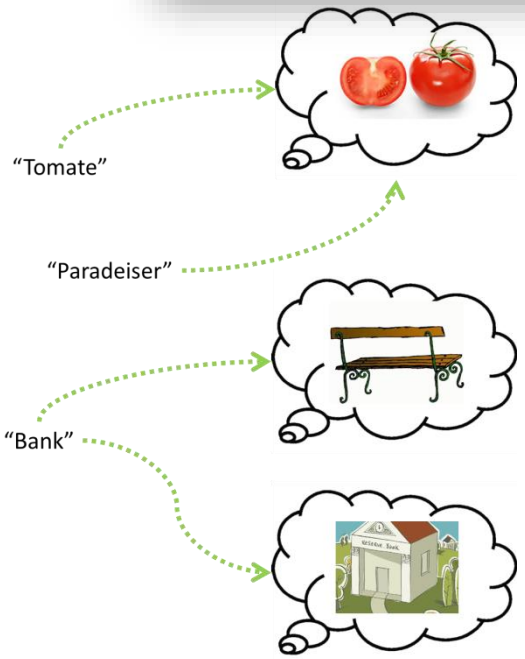
Semantic Technologies

Ontology Discription Languages RDF, RDFS, OWL

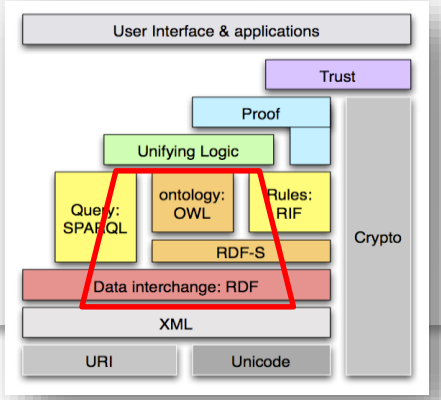
Why Ontologies?



- Base for data and knowledge sharing
- Ontologies
  - are a common vocabulary
  - to ensure that all have the same understanding of the terms used
- Automation (Reasoning Tasks)
  - Deriving implicit knowledge
  - Identification of inconsistencies
  - Determining concept hierarchies



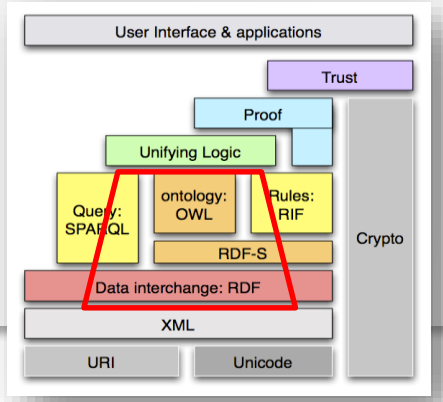
Ontologies

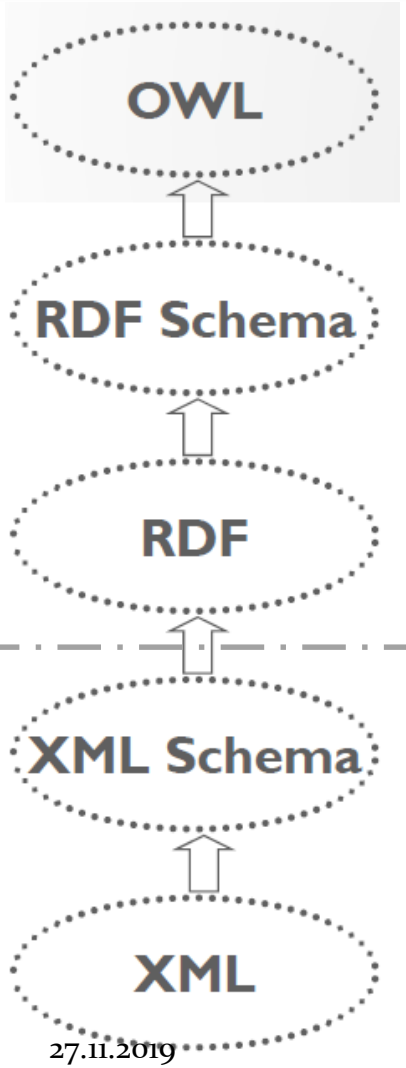


- In computer science ontologies are formally well-ordered representations of **a set of concepts and the relationships between them** in a particular subject area.
  - They are used to interchange "knowledge" digitally and formally between application programs and services.

- Ontologies include …
  - … the **description and definition** of terms (concepts)
  - their **properties**,
  - the **relationships** between them,
  - **limitations** under which the relationships are valid,
  - **inference and integrity rules** for conclusions and to ensure their validity
  - and ultimately concrete **individuals**.

Syntax and Semantics





advanced vocabulary for describing classes and properties

- Based on Description Logic
- Different Dialects

vocabulary for describing classes and properties of RDF ressources;
inheritance semantics for classes and properties

- Just for simple vocabularies
- Not very expressive

data model for resource and relationships between resources (simple semantics);
representation possible in XML

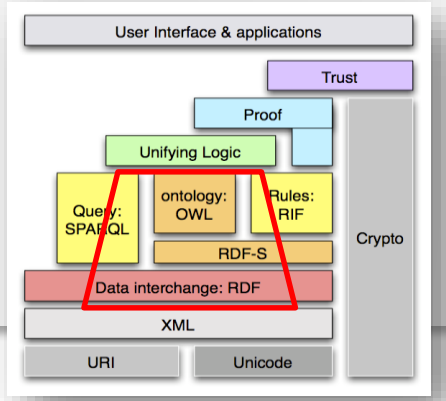
Semantics

Syntax

structural constraints on XML documents;
expansion of XML by data types

syntax for structured documents;
no semantics

RDF (1)

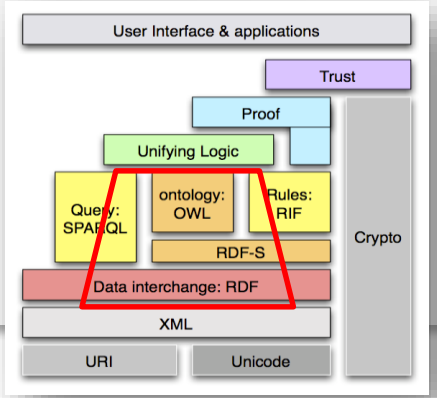


- Resource Description Framework
- (Graphic) language for representing information about resources
- Basic ontology language
- Description of resources in the form of RDF triples (Statements)

Subject – Predicate – Object
(resource – property – property value)

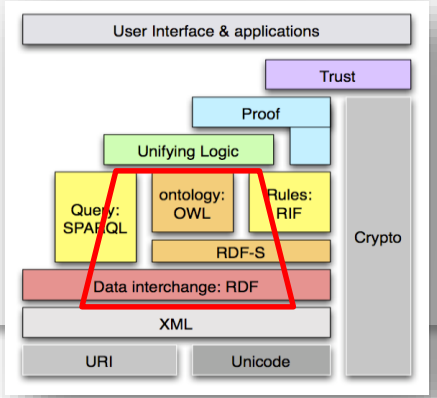
- Set of RDF triples → RDF graph

RDF (2)



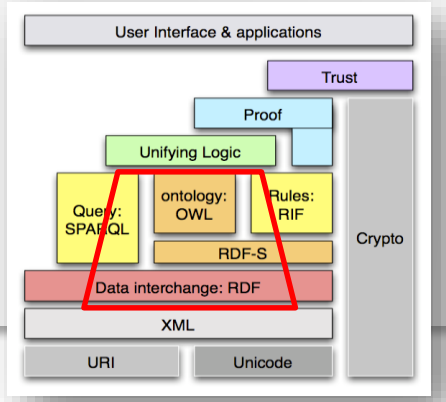
- **Subject:** Kasperl
- **Predicate:** hasFriend
- **Object:** Petzi

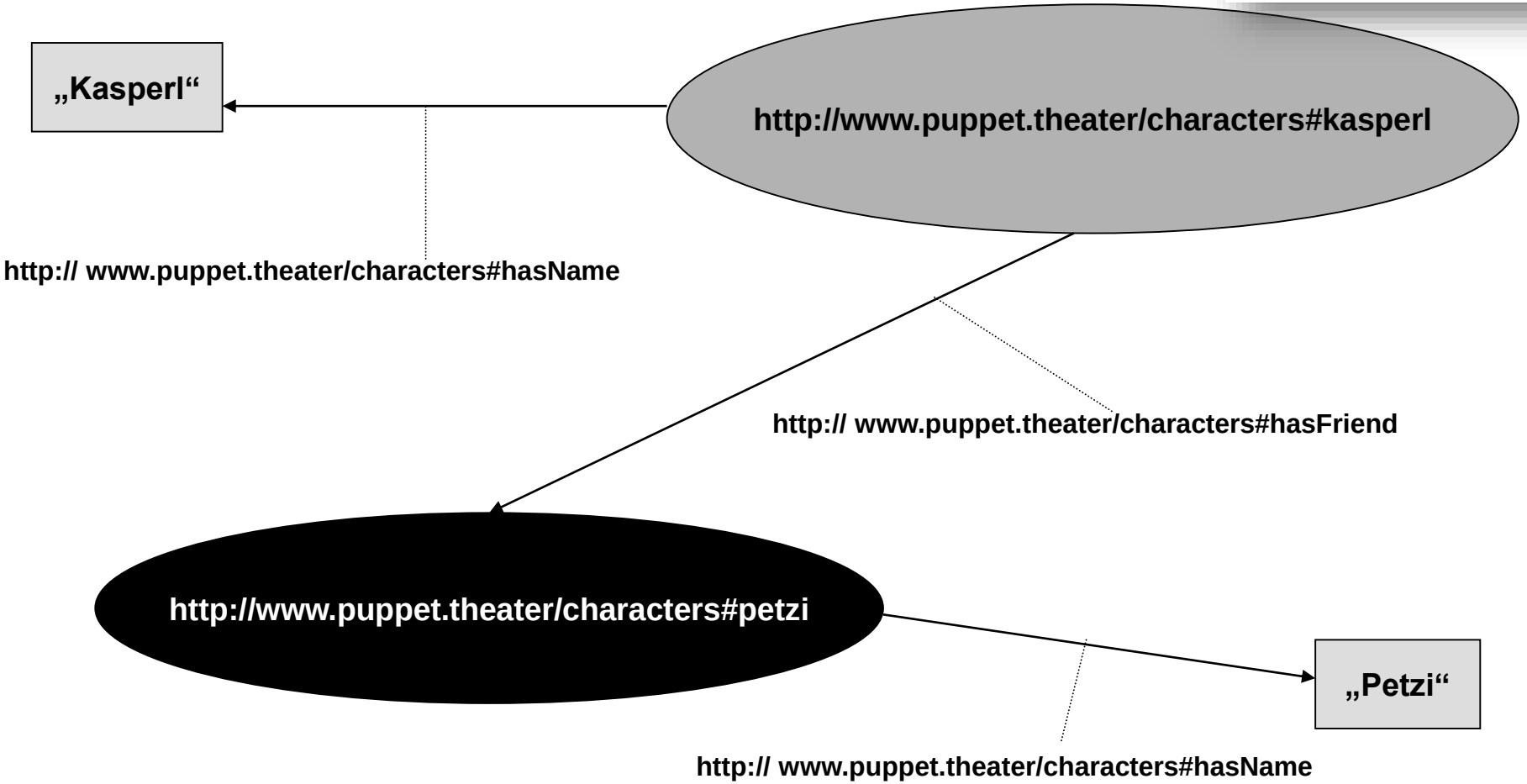
RDF (3)



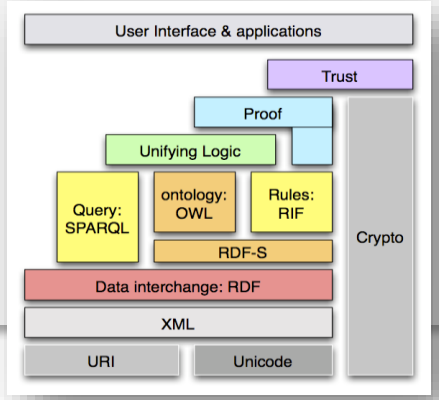
- A **resource** is an object, a statement should be made
- A resource is uniquely identified by a **URI**
  - **U**nique **R**esource **I**dentifiers + # + Fragment Identifier (optional)
  - E.g.: http://www.kasperl.theater/figuren#petzi

RDF (4)


User Interface & applications
Trust
Proof
Unifying Logic
Query: SPARQL
ontology: OWL
Rules: RIF
RDF-S
Data interchange: RDF
XML
URI
Unicode
Crypto

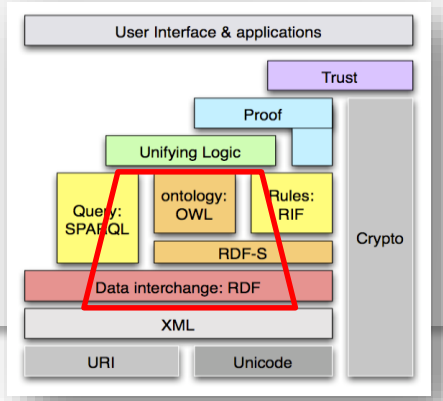

„Kasperl“
http://www.puppet.theater/characters#kasperl
http:// www.puppet.theater/characters#hasName
http:// www.puppet.theater/characters#hasFriend
http://www.puppet.theater/characters#petzi
„Petzi“
http:// www.puppet.theater/characters#hasName

RDF (5)



- **Resources**: everything that can be named and distinctly identified
- **Predicates** (properties, relations):
  - connections between resources
  - are also uniquely identified by URIs
- **rdf: type** is special predicate
  - assignment of "instances" to "classes"
  - but both concepts are resources in RDF
  - uniform representation of data and metadata
    - therefore, no strict separation between schema and instances
- **Allows Reification** (objectification)
  - Statements about statements are possible (by unique URI for statement)
- **Optimized for data exchange**
  - simple structure
  - graph structure - no root node, equivalent nodes
  - global namespace (URI)

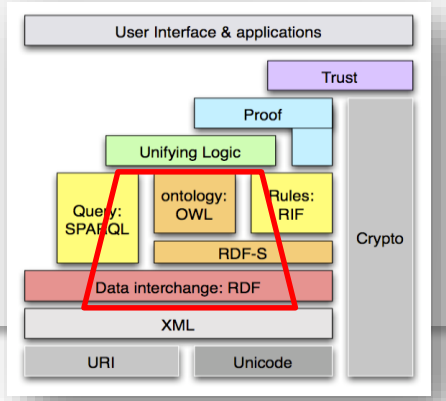
RDF (6)



- Different formats for serialization
- De facto standard: RDF/XML

```
<?xml version=„1.0“?>
<rdf:RDF
        xmlns:rdf=„http://www.w3.org/1999/02/22-rdf-syntax-ns#“
        xmlns:rdfs=„http://www.w3.org/2000/01-rdf-schema#“
        xmlns:pt=„http://www.puppet.theater/figures#“
        xml:base=„ http://www.puppet.theater/figures“>
    <rdf:Description rdf:ID=„petzi“>
        <pt:hasName>Petzi</pt:hasName>
    </rdf:Description>
    <rdf:Description rdf:ID=„kasperl“>
        <pt:hasName>Kasperl</pt:hasName>
        <pt:hasFriend rdf:resource=„#petzi“ />
    </rdf:Description>
</rdf:RDF>
```

RDF (7)

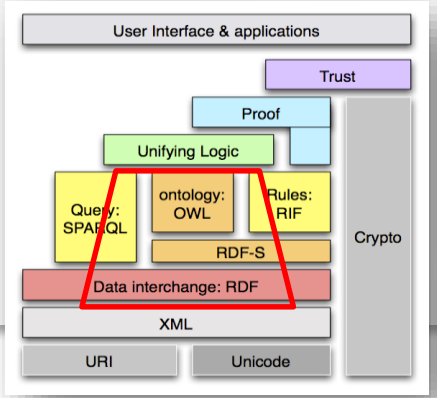


- **Terse RDF Triple Language (Turtle)**
  - better readable
  - one triple in a row

```
@prefix foaf: <http://xmlns.com/foaf/0.1> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#/>
@prefix people: http://semwebprogramming.net/people/
@prefix people: <http://semwebprogramming.net/2008/06/ont/foaf-extension# />

people:Ryan ext:worksWith people:John .
people:Matt foaf:knows people:John .
people:Andrew
        foaf:knows people:Matt ;
        foaf:surname „Perez-Lopez“ .
```

- **N-Triples**
  - simplified version of Turtle

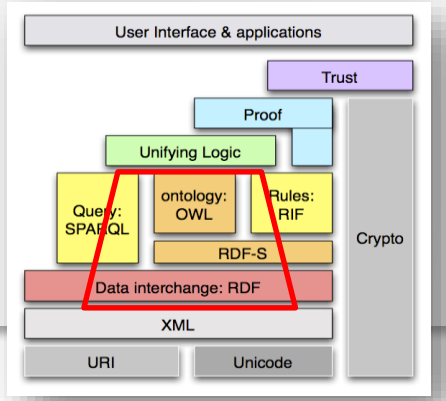
RDF (8)



- RDF Concepts
  - `rdf:Resource`
  - `rdf:Property`
  - `rdf:Statement`
  - `rdf:Type`
  - `rdf:Subject`
  - `rdf:Predicate`
  - `rdf:Object`

- Require defined vocabulary for exchange!

RDF (9)



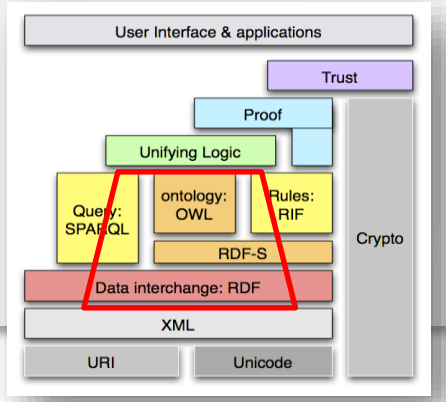
- Advanced Concepts for RDF statement:
  - **Blank nodes**
    - For nodes without globally unique ID (useful when only "intermediate nodes" are used)
    - Easy exchange between documents

```
sw:Bob sw:hasResidence [
                sw:isInCity „Arlington“ ;
                sw:isinState „VA“
                        ] .
```

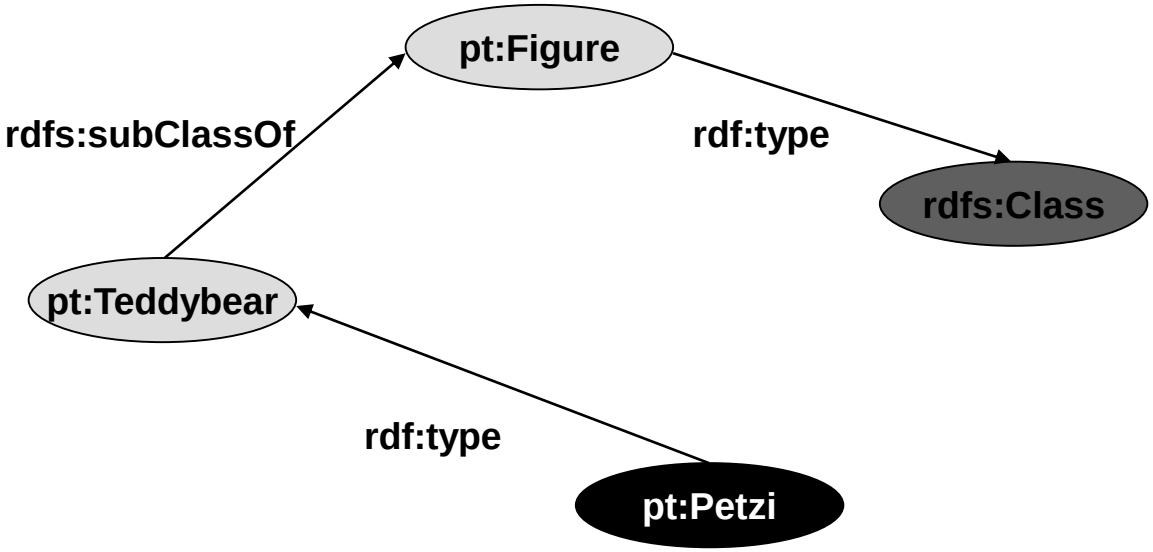
  - **Containers (rdf:Bag, rdf:Seq)**

```
ex:Authors rdf:type rdf:Bag ;
                rdf:_1 people:Ryan;
                rdf:_2 people:Matt;
                rdf:_3 people:John .

ex:Book ex:writtenBy ex:Authors .
```

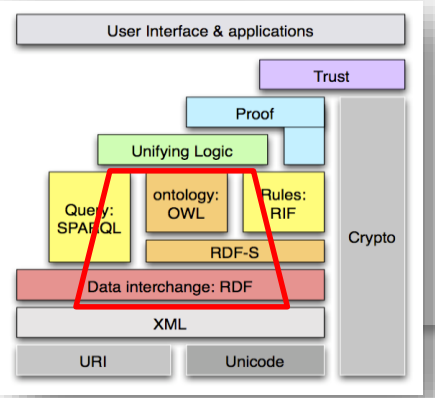
RDFS (1)



- **RDF** is a language to describe resources
  - it's a simple ontology language
- **RDF Schema** is a semantic extension of RDF
  - introduces **classes**, **properties**, and **hierarchies** (of classes and properties)
- Description of RDF resources, e.g.: `rdfs: Class`, `rdfs: subClassOf`
  - use with `rdf: type` (class - instance relationship)

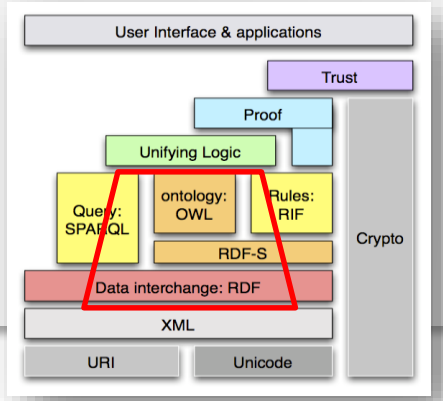


RDFS (3)



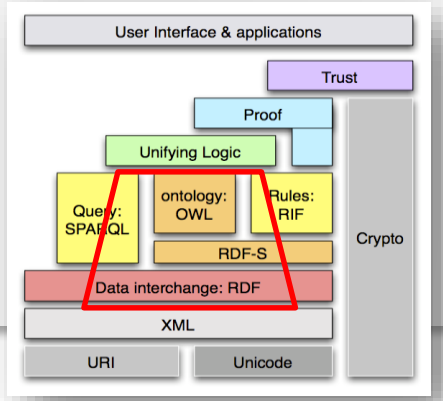
- **`rdfs:Resource`** for all ressources
- **`rdfs:Class`** class definition
- **`rdfs:Literal`** literals (strings)
- **`rdf:Property`** properties
- **`rdf:Statement`** reified statements
- **`rdf:type`** relationship between instance (resource) and class
- **`rdfs:subClassOf`** to define hierarchies
- **`rdfs:subPropertyOf`** definition of hierarchies between properties
- **`rdfs:domain:`** a domain is source (subject) of a relationship
- **`rdfs:range:`** target (object, or data type) of a relation (predicate)

RDFS (4)



Literal
range
range
DVD
UID
Phone
subClassOf
domain
domain
Work
domain
isRepresentedBy
range
Publisher
subClassOf
FilmPublisher
subClassOf
subPropertyOf
subClassOf
Book
domain
isDistributedBy
range
BookPublisher
type
type
RDF-S
RDF
A Semantic Web Primer
isDistributedBy
MIT Press

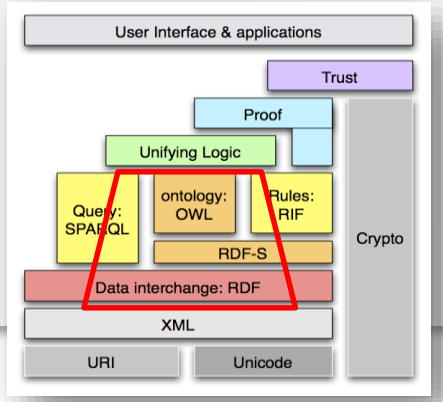
RDFS (5)



- Excerpt in XML syntax

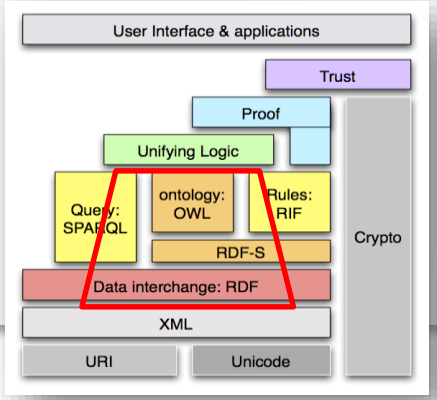
```
<rdfs:Class rdf:about="#Work" />
<rdfs:Class rdf:about="#Publisher" />

<rdfs:Class rdf:about="#Book">
  <rdfs:subClassOf rdfs:Class="#Work"/>
</rdfs:Class>

<rdfs:Class rdf:about="#BookPublisher">
  <rdfs:subClassOf rdfs:Class="#Publisher"/>
</rdfs:Class>


<rdf:Property rdf:about="#isDistributedBy">
  <rdfs:domain rdf:resource="#Book"/>
  <rdfs:range rdf:resource="#BookPublisher"/>
</rdf:Property>
```

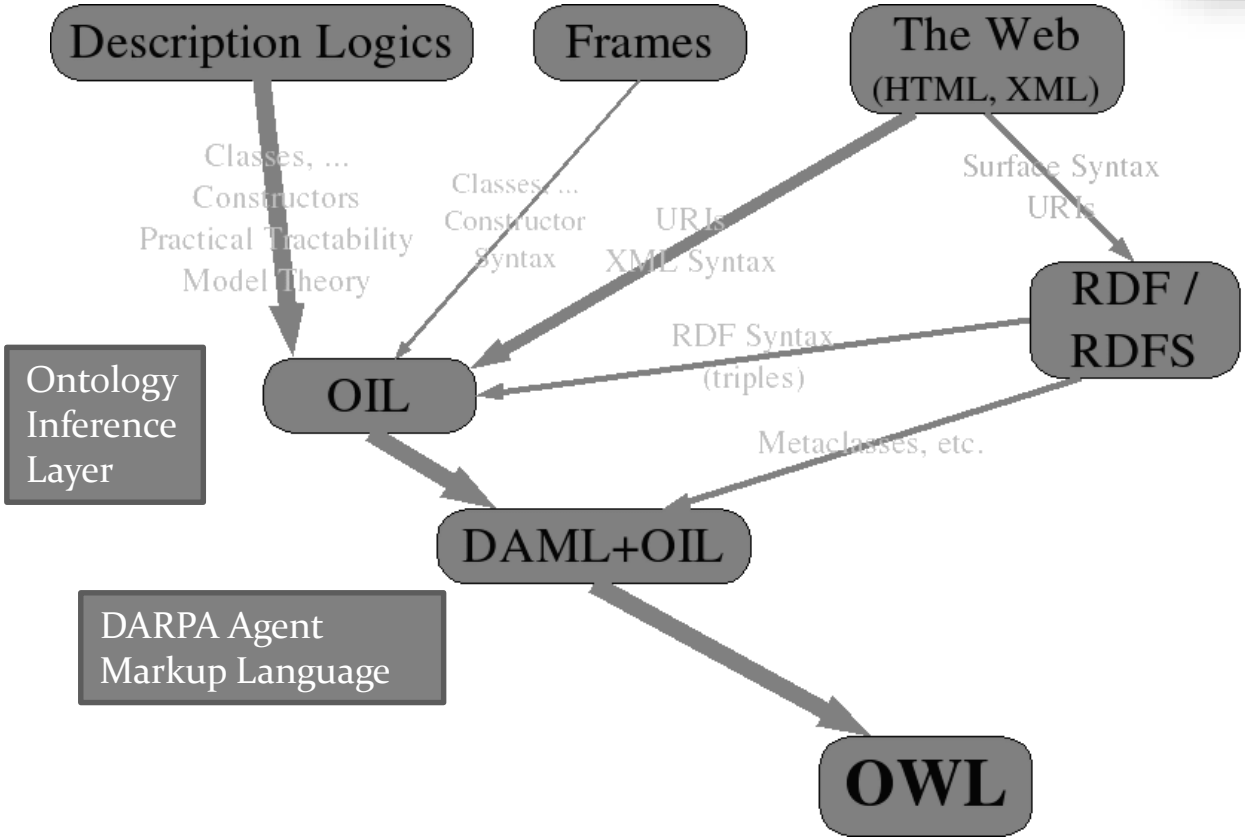
OWL (1)



- **OWL - Web Ontology Language**
  - Ontology Language (for the web)
  - Development by W3C (Web Ontology Working Group)
  - Formalism for design and dissemination of ontologies (semantic web)
  - Influenced by RDF(S), Description Logic (DL), and Frame Concepts
  - Language for ontology description and reasoning in ontologies → structuring and linking information
  - Determination of the meaningfulness of ontologies
  - Representation in RDF / XML syntax possible
  - Representation in Description Logic possible

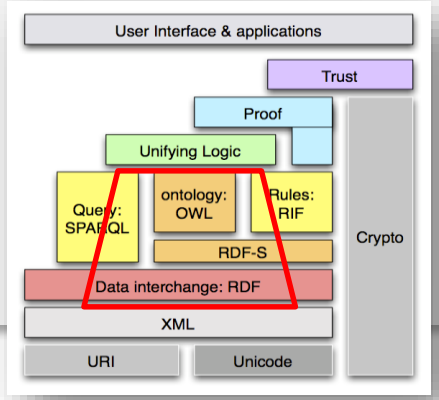
OWL (2)





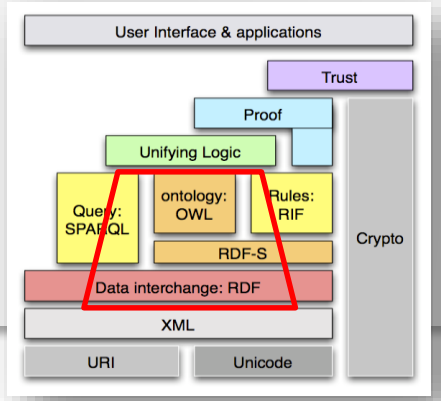
http://ect.bell-labs.com/who/pfps/talks/owl/#%2810%29

OWL (3)

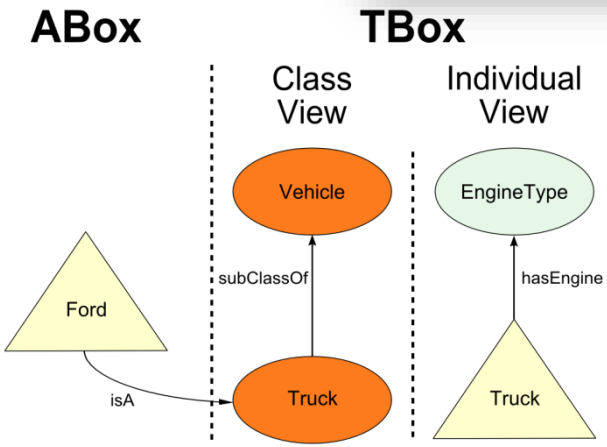


- Ontologies as distributed, machine-processable vocabularies for a specific domain
  - Describe a detail of the world:
    - What things are there?
    - What is the relation of these things?
    - What characteristics do these things and the relationships with each other have?
  - Ontology description = Syntax
  - Ontology rules = Semantics

OWL (5)

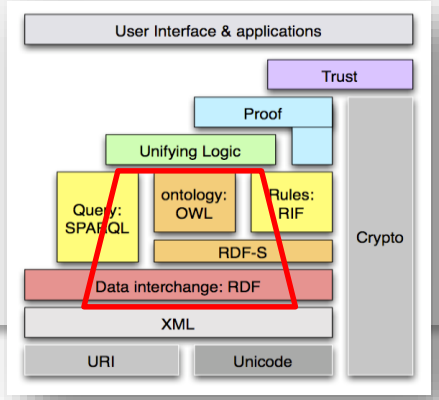


- **Language for structuring information**
  - What things are there?
    - **Individuals (ABox)**
  - What is the relation of these things?
    - **Schema definition level (TBox)**



- **Open-World vs. Closed World Assumption**
  - **Open World Assumption:** A reasoner does not assume that something does not exist, as long as it is not explicitly defined that it does not exist.
    - It is assumed that the knowledge has not been added to the knowledge base yet.
    - This allows uncertainty and ambiguity.
    - Not to know that a statement is true, does not mean that this is wrong!
    - Be careful with restrictive conditions when reasoning (e.g.: max, exactly, only).
  - In databases typically the **Closed World Assumption** is applied.

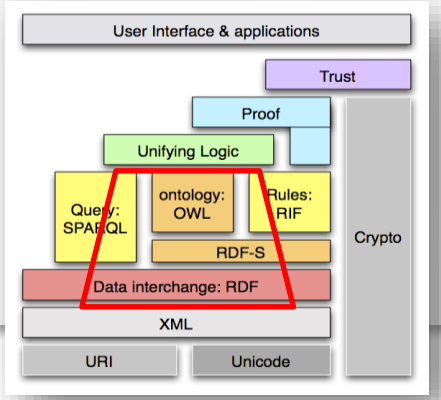
OWL (6)



- **Example in RDF/OWL**

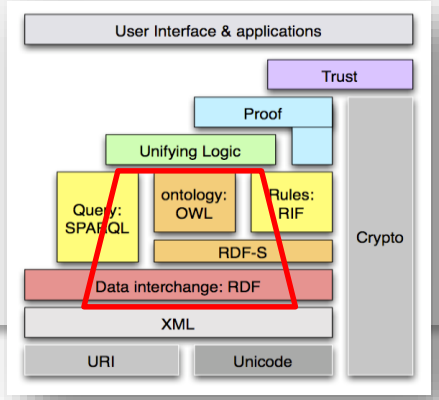
```
<owl:Class rdf:ID="WhiteWine">
  <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Wine" />
      <owl:Restriction>
          <owl:onProperty rdf:resource="#hasColor"
  />
          <owl:hasValue rdf:resource="#White" />
      </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

OWL (7)



- Example in Description Logic

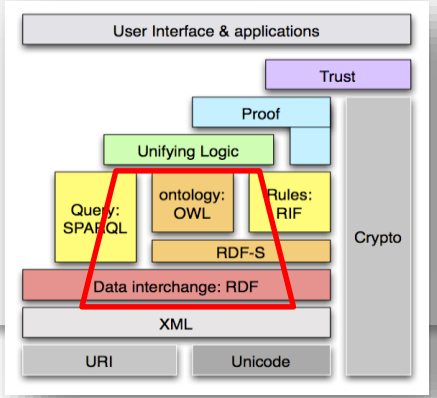
OWL (8)



- **OWL Dialects**
  1. **OWL Full:** all expressive possibilites of OWL and RDF; not decidable
  2. **OWL-DL:** restricted to enable decidability and to distinguish between classes and individuals
  3. **OWL Lite:** Simplified version of OWL-DL, based on Frames Notation

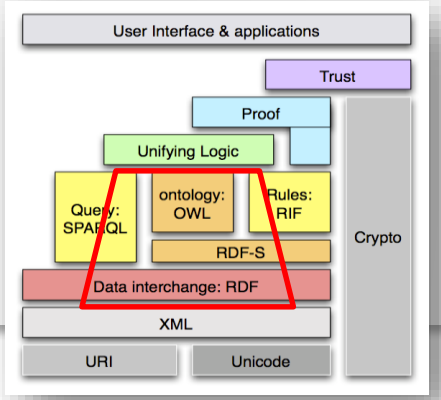
- **OWL 2 Profiles**
  - **OWL 2 EL:** Useful in applications employing **ontologies that contain very large numbers of properties and/or classes**. A subset of OWL 2 for which the basic reasoning problems can be performed in time that is polynomial with respect to the size of the ontology. Dedicated reasoning algorithms for this profile are available and are implementable in a highly scalable way.
  - **OWL 2 QL:** For applications that use very **large volumes of instance data**, and where query answering is the most important reasoning task. Using a suitable reasoning technique, sound and complete conjunctive query answering can be performed in logarithmic space with respect to the size of the data (assertions).
  - **OWL 2 RL:** For applications that require **scalable reasoning** without sacrificing too much expressive power. OWL 2 RL reasoning systems can be implemented **using rule-based reasoning engines**. The ontology consistency, class expression satisfiability, class expression subsumption, instance checking, and conjunctive query answering problems can be solved in time that is polynomial with respect to the size of the ontology.

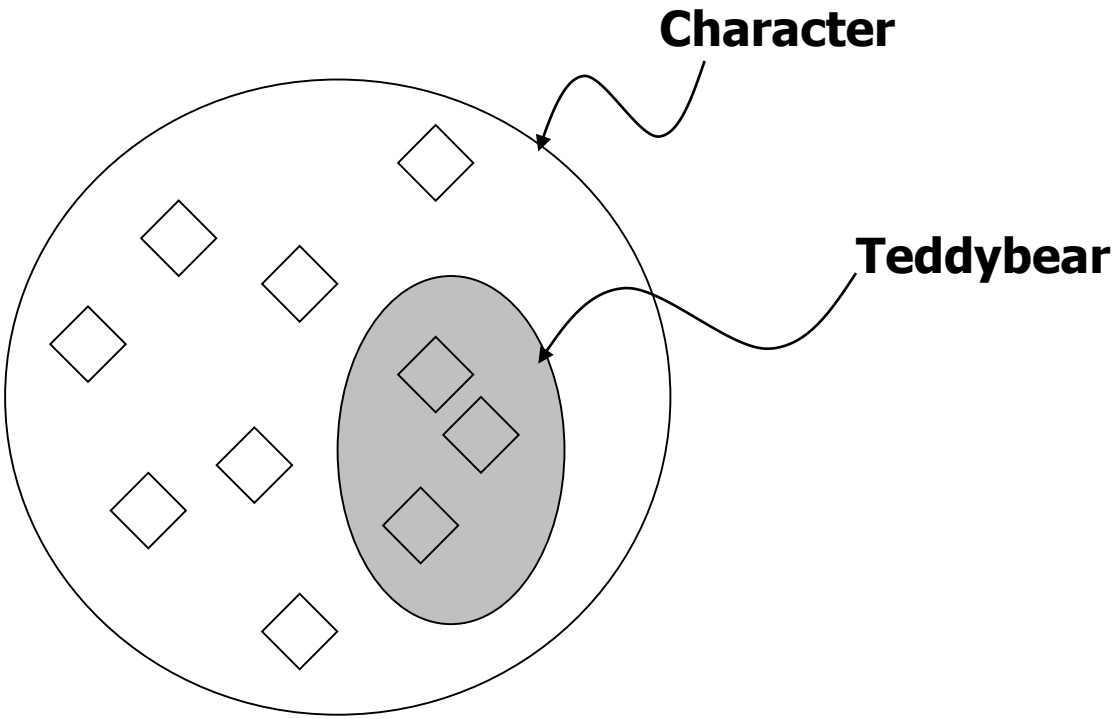
Ontologies in OWL (1) Classes



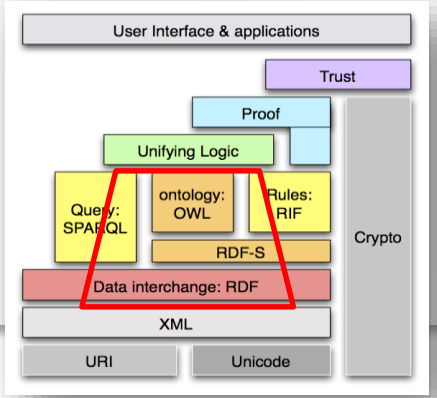
- **Class Types**
  - **Named Classes**
    - `red wine, white wine, rosé`
  - **Disjoint Classes**
    - Created by `owl:disjointWith` statement
  - **Average Classes**
    - `Burgundy ∏ WhiteWine = WhiteBurgundy`
  - **Union Classes**
    - `Wine = RedWine + WhiteWine + RoséWine`
  - **Complementary Classes**
    - `RedWine = Wine + !WhiteWine + !RoséWine`
  - **Anonymous Classes**
    - Created by restrictions on properties
  - **Enumerations**
    - `Color = red, white, rosé`
  - **Primitive and Defined Classes**
    - Necessary vs. Necessary & Sufficient Conditions
  - **Equivalent Classes**
    - Class A is equivalent to (Intersection of Class B and Class C)

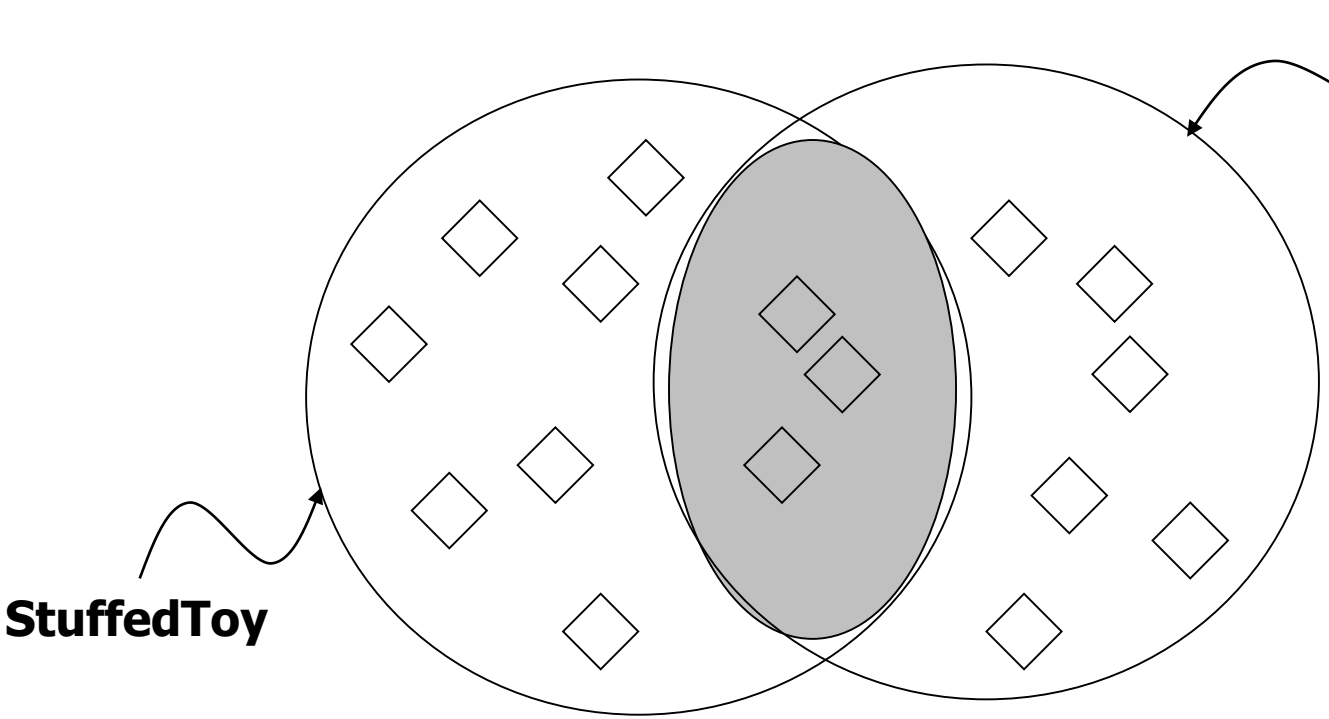
Ontologies in OWL (2)
Named Classes





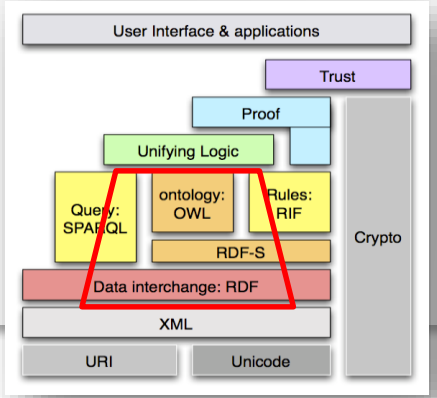
Ontologies in OWL (3) Intersection Classes

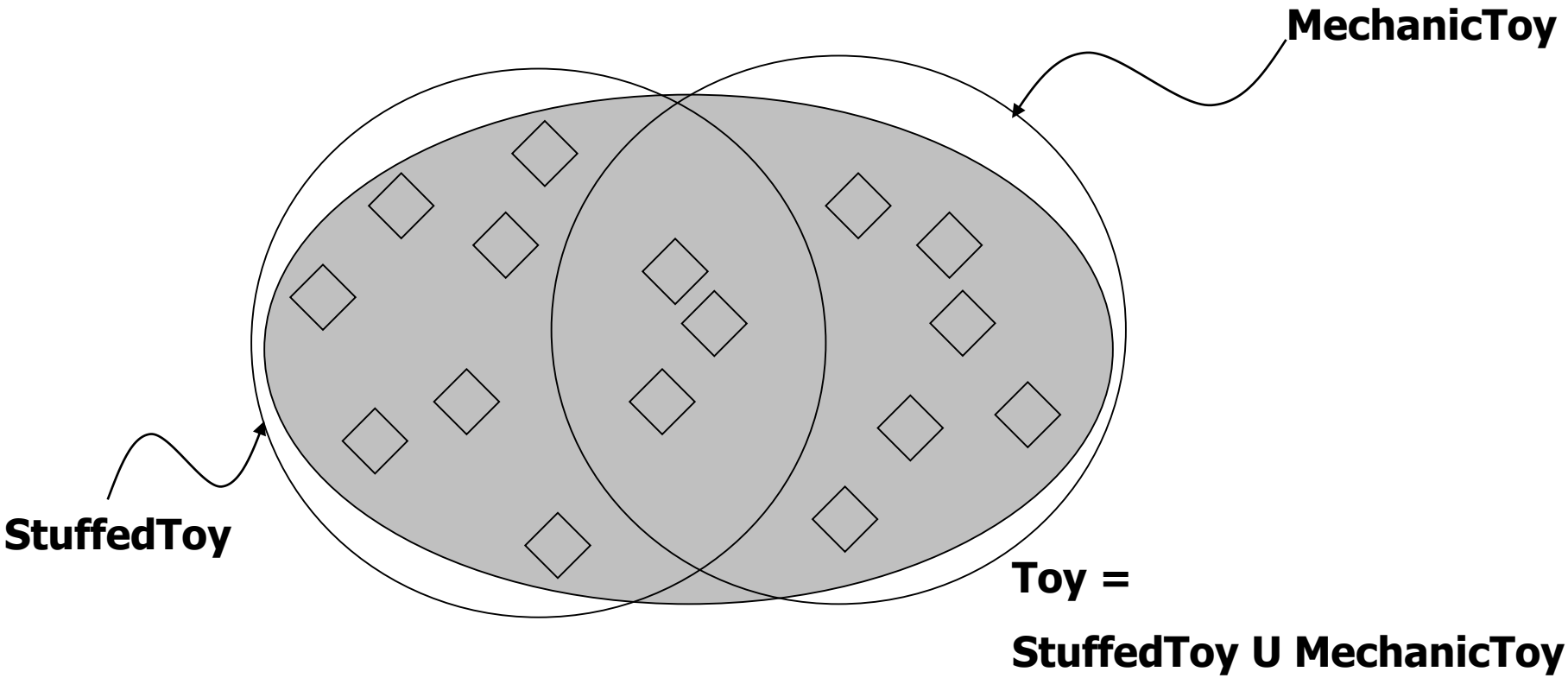




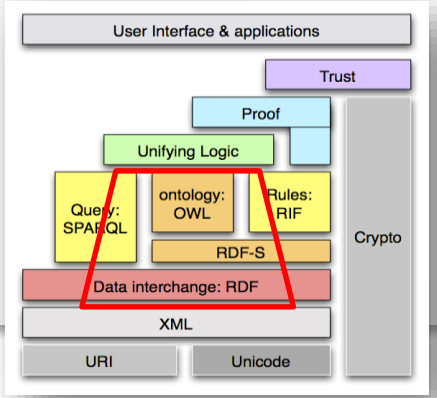
Teddybear = StuffedToy ∩ Bear (?)

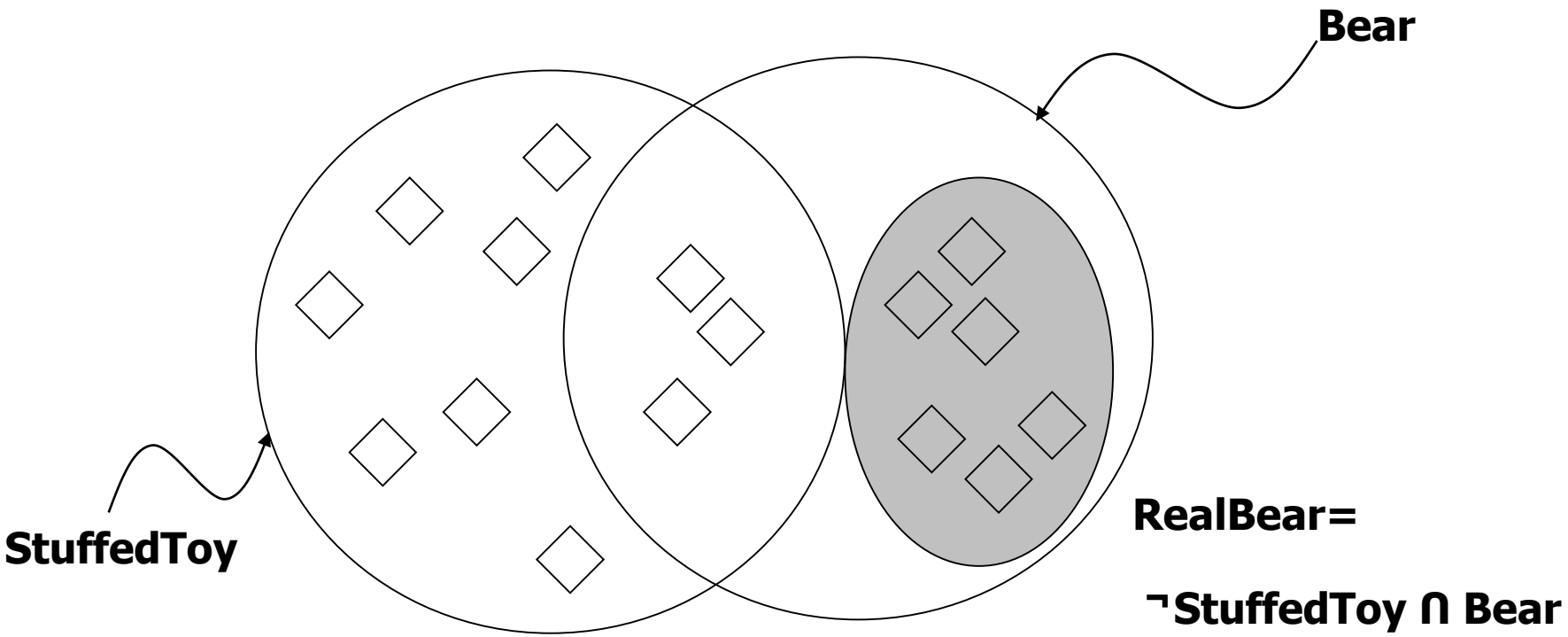
Ontologies in OWL (4)
Union Classes



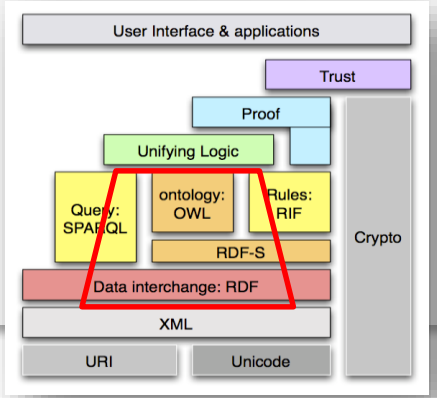


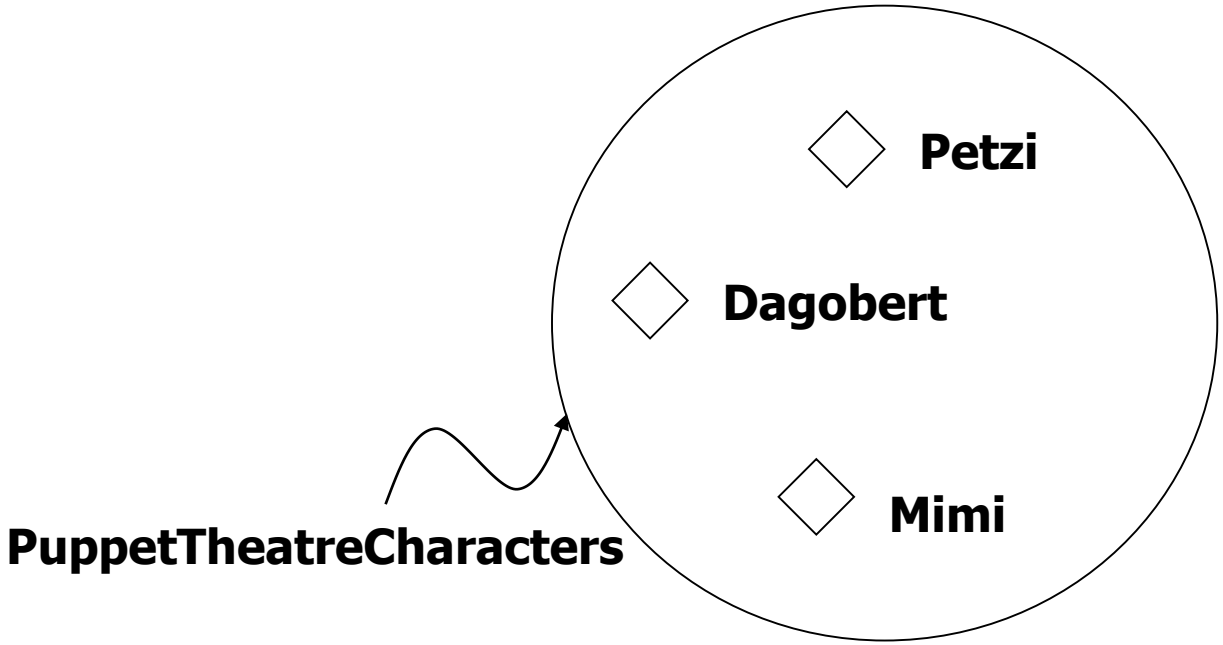
Ontologies in OWL (5) Complementary Classes



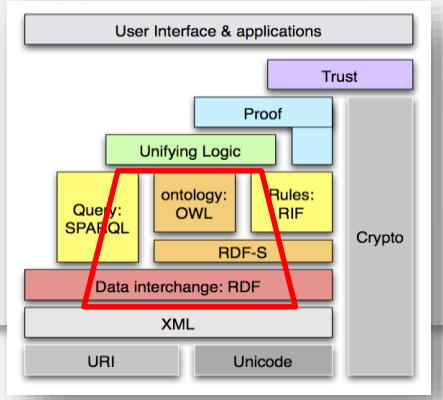


Ontologies in OWL (6) Enumerations



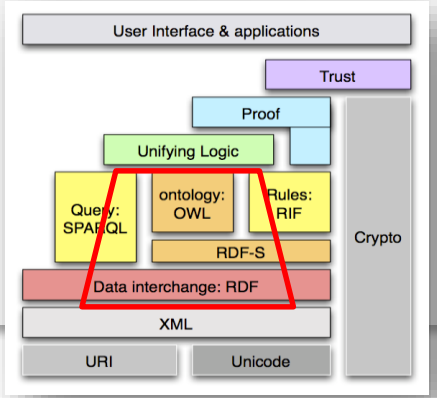


Ontologies in OWL (7) Anonymous Classes



- **Limitations on properties (property restrictions)**
  - Union of individuals (as anonymous classes) because of the nature and number of relationships in which they participate, by:
    - **Quantifiers**: existential, universal
    - **Cardinality:** min, max, exactly
    - **Explicit Value Assignment** (hasValue)

Ontologies in OWL (8) Anonymous Classes

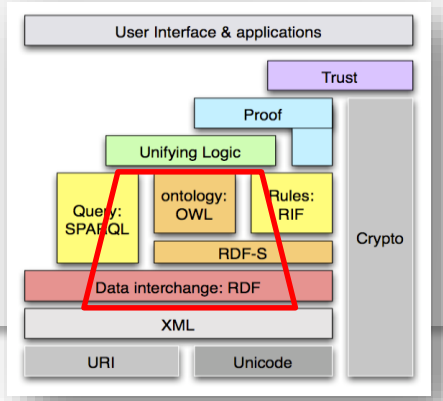


- ***Restrictions*** **– Existantial Quantifier**

$\exists$

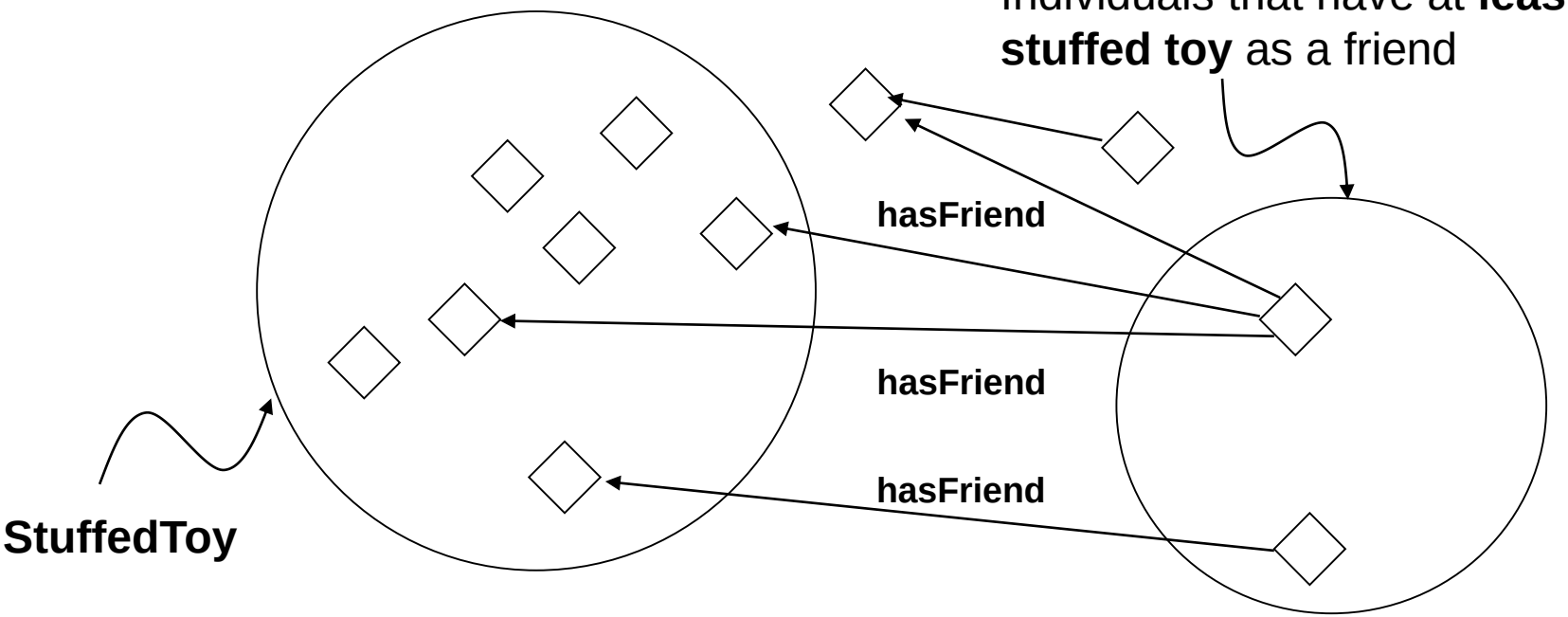
  - Individuals that have at least one certain relationship to another particular individual
  - *someValuesFrom*

Ontologies in OWL (9) Anonymous Classes

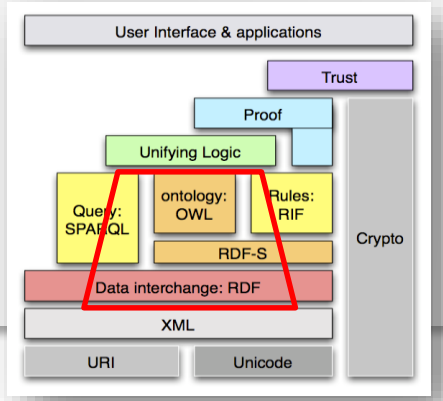


- ***Restrictions*** **– Existantial Quantifier:**
  ∃ `hasFriend StuffedToy`

Anonymous Class:
Individuals that have at **least one stuffed toy** as a friend



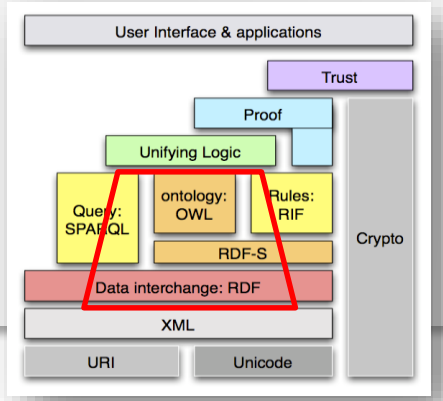
Ontologies in OWL (10) Anonymous Classes



- ***Restrictions*** **– Universal Quantifier**

  - Individuals that have only certain relationships to other specific individuals, or none.
  - *allValuesFrom*

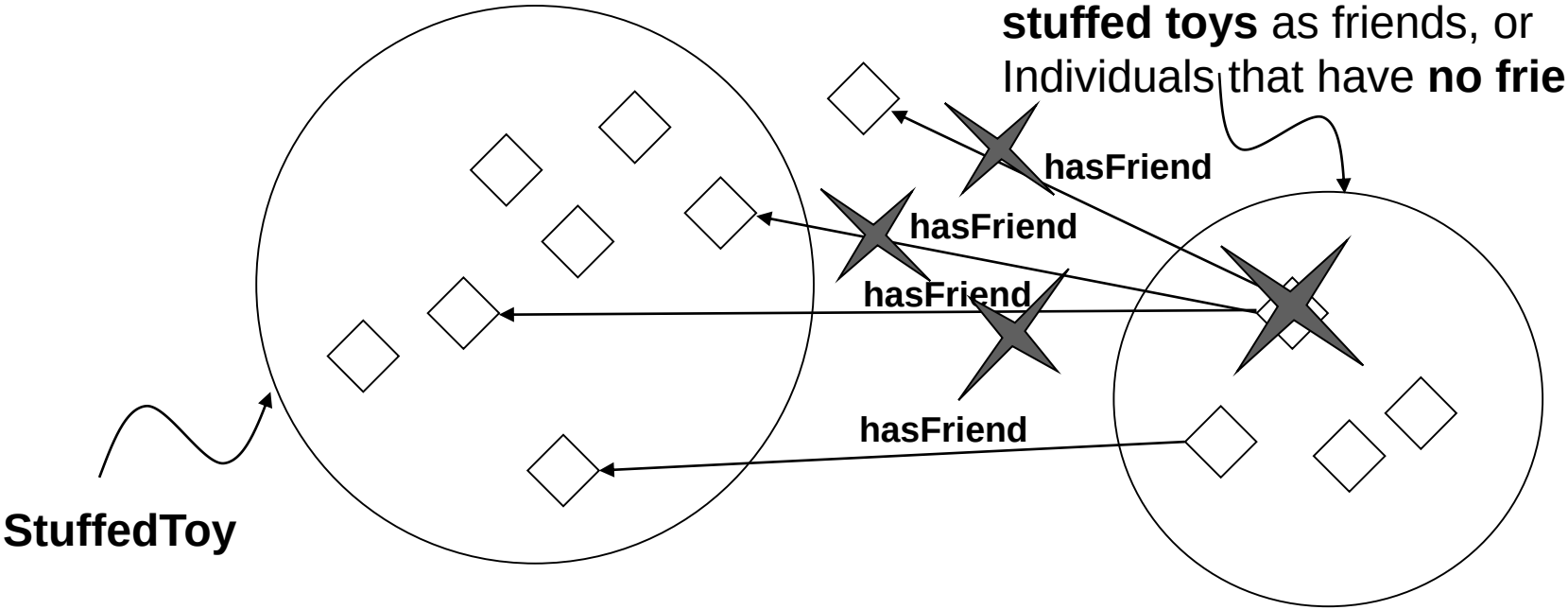
Ontologies in OWL (11) Anonymous Classes



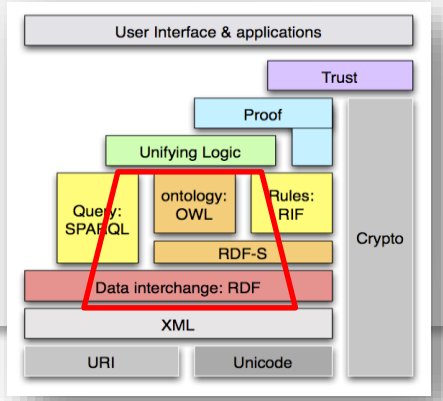
- ***Restrictions*** **– Universal Quantifier:**

  ∀ hasFriend StuffedToy

Anonymous Class:
Individuals that have **only stuffed toys** as friends, or
Individuals that have **no friends**



Ontologies in OWL (12) Anonymous Classes

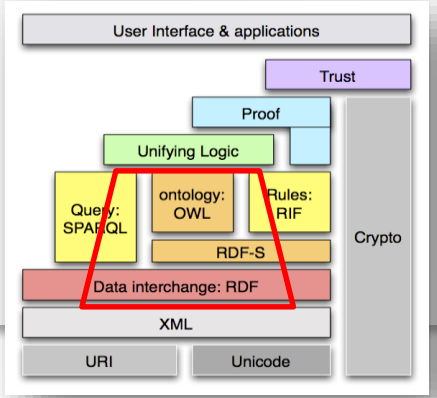


- ***Restrictions*** **– Combination of quantifiers:**

```
Class Character (
   (hasFriend someValuesFrom StuffedToy) ^
   (hasFriend allValuesFrom stuffedToy)
)
```

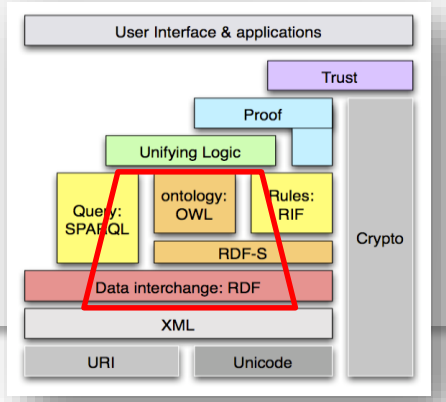
Restriction that characters have friends (at least one) AND that every friend has to be a stuffed toy.

Ontologies in OWL (13) Anonymous Classes

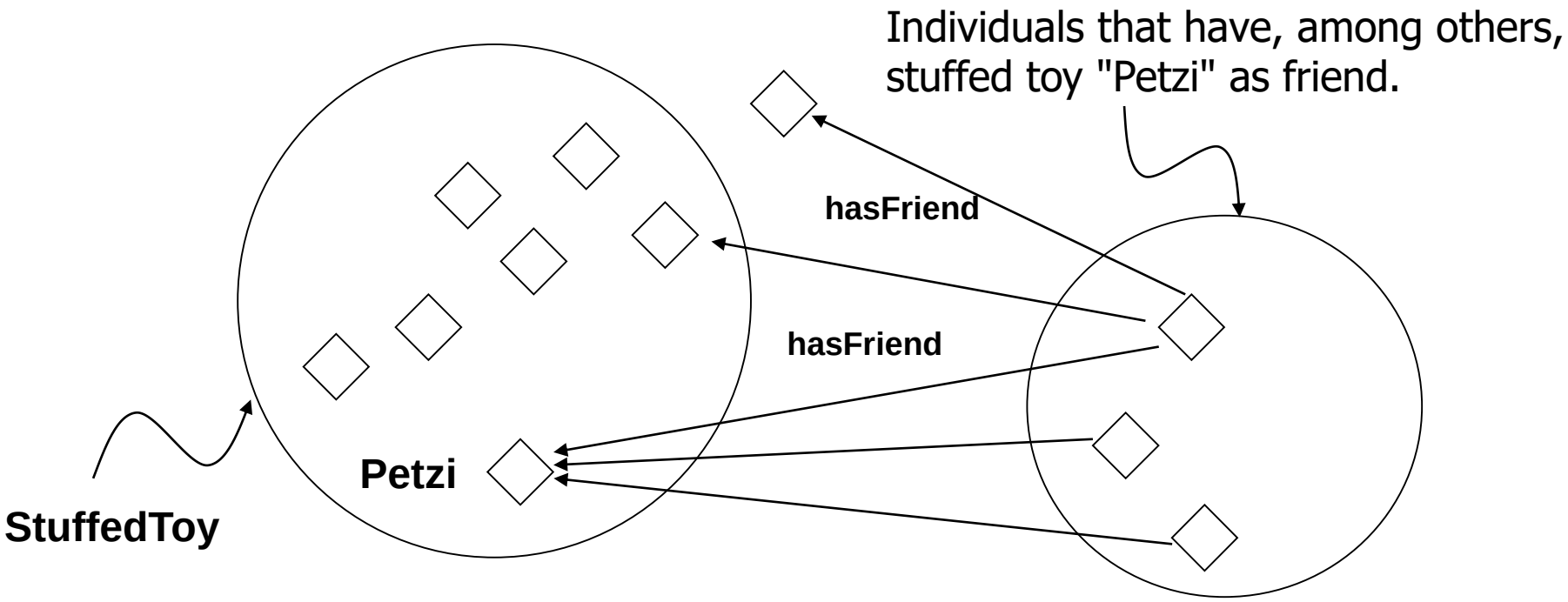


- ***Restrictions*** **– Cardinality:**
  - <= … maximum
  - >= … minimum
  - = … exact number

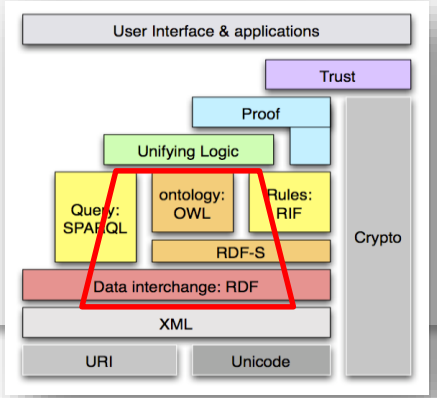
Ontologies in OWL (14) Anonymous Classes



- Beschränkungen (restrictions) –Wertzuweisung (hasValue): hasFriend Petzi



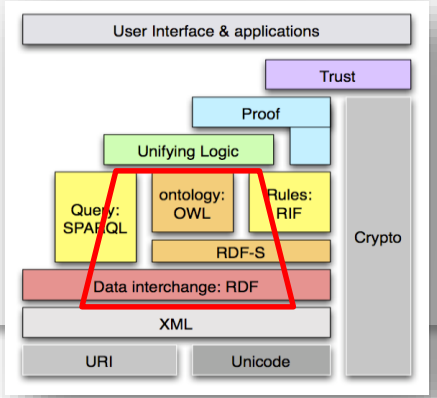
Ontologies in OWL (15)
Primitive and Defined Classes



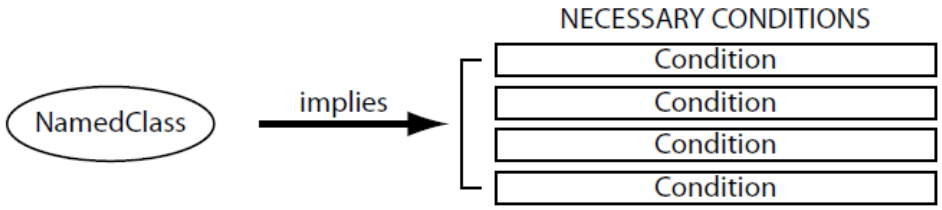
- Primitive Classes vs. Defined Classes
- Necessary vs. Necessary and Sufficient Conditions
- Partial vs. Complete Definitions

- For defining necessary and sufficient conditions OWL offers the `owl:equivalentClass` construct.

Ontologies in OWL (16)
Primitive and Defined Classes

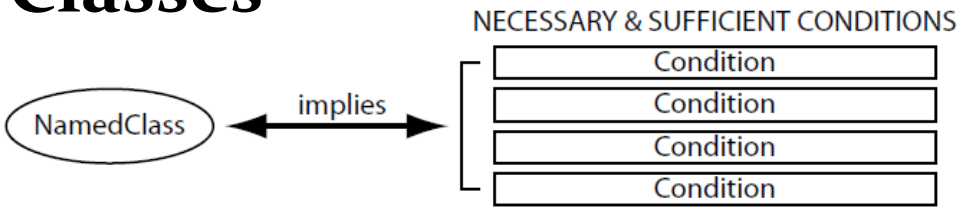


- **Primitive Classes**



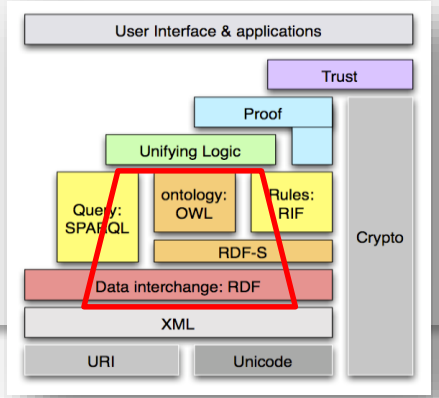
If an individual is a member of 'NamedClass' then it must satisfy the conditions. However if some individual satisfies these necessary conditions, we cannot say that it is a member of 'Named Class' (the conditions are not 'sufficient' to be able to say this) - this is indicated by the direction of the arrow.

- **Defined Classes**

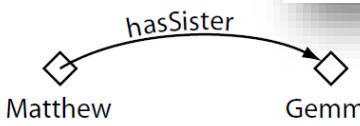


If an individual is a memeber of 'NamedClass' then it must satisfy the conditions. If some individual satisfies the condtions then the individual must be a member of 'NamedClass' - this is indicated by the double arrow.

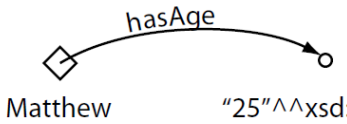
Ontologies in OWL (17) Properties



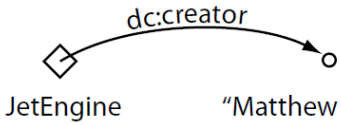
- ***Object Properties*** link an individual to an individual.
- ***Datatype properties*** link an individual to an XML Schema Datatype value or an rdf literal.
- ***Annotation Properties*** can be used to add information (metadata, data about data) to classes, individuals and object/datatype properties.



An object property linking the individual Matthew to the individual Gemma

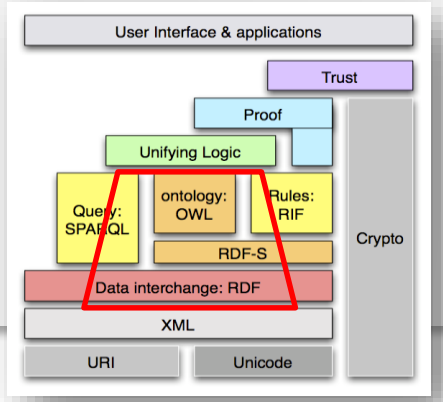


A datatype property linking the individual Matthew to the data literal '25', which has a type of an xsd:integer.



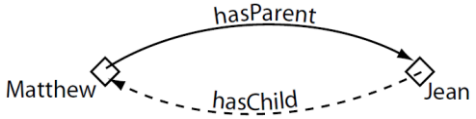
An annotation property, linking the class 'JetEngine' to the data literal (string) "Matthew Horridge".

Ontologies in OWL (18)
Object Property Characteristics



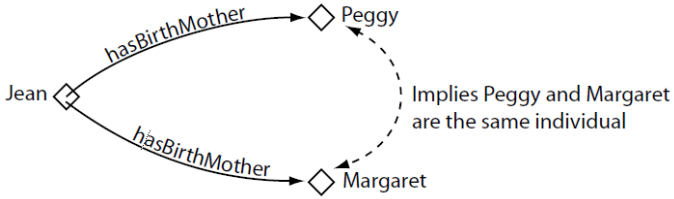
- **inverse**

  Each object property may have a corresponding *inverse property*. If some property links individual a to individual b then its *inverse property* will link individual b to individual a.



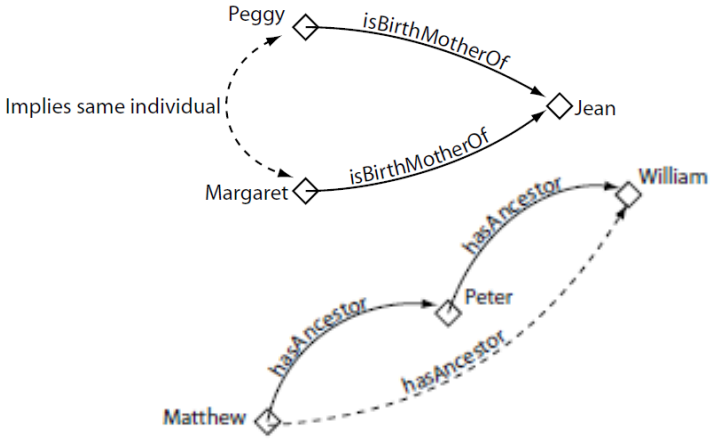
- **functional**

  Exactly one individual is in relation via this property (single valued property).



- **inverse functional**

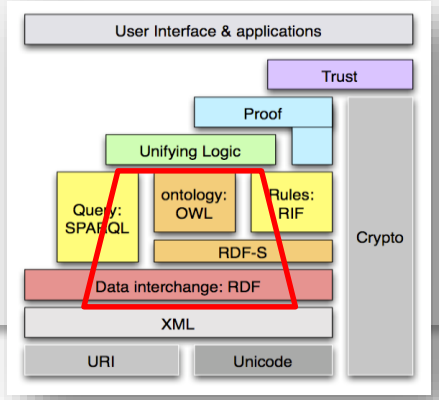
  The *inverse property* is *functional* (e.g.: isBirthMotherOf).



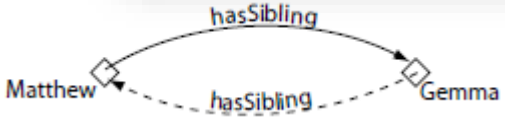
- **transitive**

  If a property P is *transitive*, and the property relates individual a to individual b, and also individual b to individual c, then we can infer that individual a is related to individual c via property P.

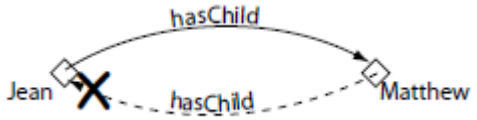
Ontologies in OWL (19)
Object Property Characteristics



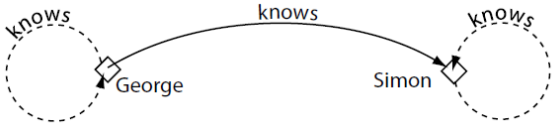
- **symmetric**
  - If a property P is symmetric, and the property relates individual a to individual b then individual b is also related to individual a via property P.



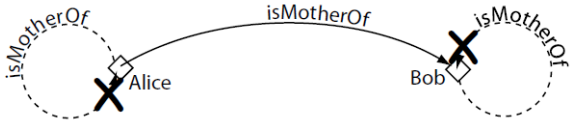
- **asymmetric**
  - If a property P is asymmetric, and the property relates individual a to individual b then individual b cannot be related to individual a via property P.



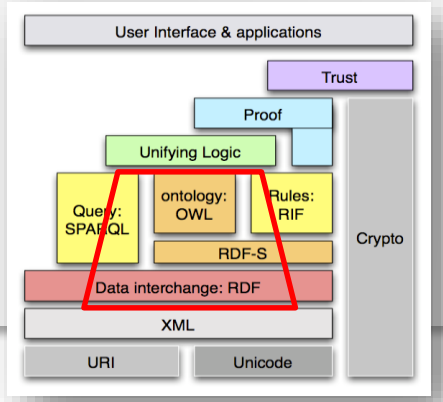
- **reflexive**
  - A property P is said to be reflexive when the property must relate individual a to itself.



- **irreflexive**
  - If a property P is irreflexive, it can be described as a property that relates an individual a to individual b, where individual a and individual b are not the same.



Ontologies in OWL (20)
Object Property Characteristics

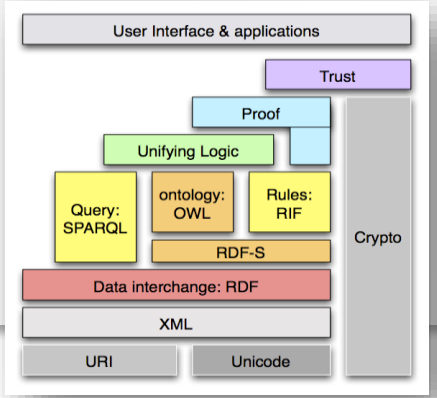


- **Domain and Range**
  - Properties may have a domain and a range specified. Properties link individuals from the domain to individuals from the range.

- **Important:** in OWL domains and ranges should not be viewed as constraints to be checked. They are used as 'axioms' in reasoning.

  Example: The property hasName has the domain set Person. When we then applied the hasName property to Pet (individuals that are members of the class Pet), this would generally not result in an error. It would be used to infer that the class Pet must be a subclass of Person!

Ontologies in OWL (21)
Datatype Properties



- *Datatype Properties – Attributes*
  - Datatype properties link an individual to an XML Schema Datatype value or an rdf literal.
  - In other words, they describe relationships between an individual and data values.
  - Most of the property characteristics described before cannot be used with datatype properties.
    Exception: By describing a datatype property as functional we state that any given individual can only ever have one value.
  - They can also be used to create defined classes that specify a range of interesting values using minInclusive and maxExclusive facets that can be applied to numeric datatypes.

    Example: A HighPerfomanceCar will be defined to be *any car that has a high-speed value equal to or higher than 350* adding the restriction: 'hasHighSpeedValue Value some integer[>= 350]'

OBERÖSTERREICH
University of Applied Sciences

Web-Semantik-Technologien

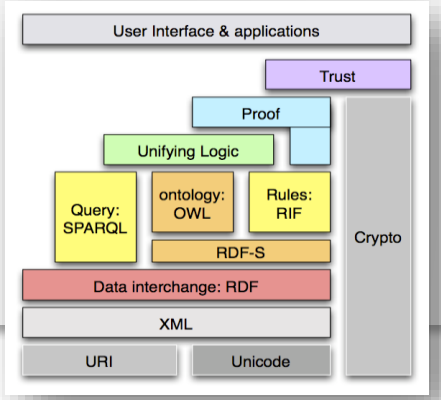
Vorlesung WS 2017/18

thomas.kern@fh-hagenberg.at

Semantic Technologies

Rules – SWRL

What is SWRL?



- The Semantic Web Rule Language (SWRL) is an expressive OWL-based rule language.
- SWRL provides more powerful deductive reasoning capabilities than OWL alone.
- SWRL is built on the same description logic foundation as OWL
- SWRL provides similar strong formal guarantees when performing inference.

SWRL Rules

- A SWRL rule contains an ***antecedent part(body)***, and a ***consequent (head)***.
- Both the body and head consist of positive conjunctions of atoms:

$$\underbrace{atom \wedge atom \ldots}_{\text{antecedent}} \rightarrow \underbrace{atom \wedge atom \ldots}_{\text{consequent}}$$

- An atom is an expression of the form:

$$p(arg1, arg2, \ldots, argn)$$

- where ***p*** is a predicate symbol (OWL classes, properties or data types)
- ***arg1, arg2, …, argn*** - arguments of the expression.(OWL individuals or data values, or variables referring to them)

SWRL Atoms

- SWRL provides seven types of atoms:
  - Class Atoms — ***owl:Class***
  - Individual Property atoms — ***owl:ObjectProperty***
  - Data Valued Property atoms — ***owl:DatatypeProperty***
  - Different Individuals atoms
  - Same Individual atoms
  - Built-in atoms
  - Data Range atoms

Class Atoms

- A class atom consists of an **OWL named class** or **class expression** and a single argument representing an OWL individual.

  Person(?p)
  Man(Fred)

  - ***Person***, ***Man***: OWL named classes
  - ***?p***: variable representing an OWL individual
  - ***Fred***: name of an OWL individual.

- Simple Example: all individuals of type ***Man*** are also or type ***Person***:
  Man(?p) -> Person(?p)

  - Of course, this statement can also be made directly in OWL.

Individual Property Atoms

- Consists of an **OWL object property** and **two arguments** representing OWL individuals.

  hasBrother(?x, ?y)
  hasSibling(Fred, ?y)

  - ***hasBrother, hasSibling***: OWL object properties
  - ***?x, ?y:*** variables representing OWL individuals
  - ***Fred***: name of an OWL individual.

- person with a male sibling has a brother :

  Person(?p) ^ hasSibling(?p,?s) ^ Man(?s) -> hasBrother(?p,?s)

  - Person and male can be mapped to OWL class called Person with a subclass Man
  - The sibling and brother relationships can be expressed using **OWL object properties *hasSibling*** and ***hasBrother*** with a domain and range of ***Person***.

Data Valued Property Atoms

- A data valued property atom consists of an **OWL data type property** and **two arguments** (OWL individual , data value)

hasAge(?x, ?age) ***hasHeight(Fred, ?h)***
hasAge(?x, 232) ***hasName(?x, "Fred")***

- All persons that own a car should be classified as drivers:
Person(?p) ^ hasCar(?p, true) -> Driver(?p)

- This rule classifies all car-owner individuals of type ***Person*** also as members of the class ***Driver***.

- Named individuals can be referred directly :
Person(Fred) ^ hasCar(Fred, true) -> Driver(Fred)

  - This rule works with a known individual called ***Fred*** in an ontology. One can not create a new individual using rules of this form.

Different and Same Individuals

- SWRL supports ***sameAs*** and ***differentFrom*** atoms to determine if individuals refer to the same underlying individual or are distinct, and can use ***owl:sameAs , owl:allDifferents***

- **No unique name assumption!**

- A different individuals atom consists of the ***differentFrom*** symbol and two arguments representing OWL individuals.

  differentFrom(?x, ?y)
  differentFrom(Fred, Joe)
  sameAs(?x, ?y)
  sameAs(Fred, Freddy)

- If two OWL individuals of type ***Author*** cooperate on the same publication that they are collaborators:
  Publication(?p) ^ hasAuthor(?p, ?x) ^ hasAuthor(?p, ?y) ^ differentFrom(?x, ?y) -> cooperatedWith(?x, ?y)

Data Range Atoms

- A data range atom consists of a **datatype name** or a **set of literals** and a single argument representing a **data value**.

  xsd:int(?x)

  [3, 4, 5](?x)

- Here, ***?x*** is a variable representing a data value.

Built-In Atoms

- One of the most powerful features of SWRL is its ability to support user-defined built-ins.
- A built-in is a predicate that takes one or more arguments and evaluates to true if the arguments satisfy the predicate.
- Core SWRL built-ins are preceded by the namespace qualifier ***swrlb***.
- SWRL supports user-defined built-ins

- Person with an age of greater than 17 is an adult:

  Person(?p) ^ hasAge(?p, ?age) ^ swrlb:greaterThan(?age, 17) -> Adult(?p)

- Person's telephone number starts with the international access code "+"

  Person(?p)^hasNumber(?p, ?number) ^ swrlb:startsWith(?number, "+") -> hasInternationalNumber(?p,true)

Built-In Atoms

- Built-ins can take any number or combination of OWL datatype property values.
- They can not take object, class or property values. However, the SWRLTab implementation of SWRL has custom extensions to allow arguments of these types.
- Argument number and types checking is the responsibility of the built-in implementor.
- If an incorrect number or type of arguments is passed, built-in should evaluate to false.
- SWRL allows new libraries of built-ins to be defined and used in rules. Users can define built-in libraries to perform a wide range of tasks. Such tasks could, for example, include currency conversion, temporal manipulations, and taxonomy searches.

Binding Arguments

- Built-ins can also assign (or bind) values to arguments.

- ***swrlb:add(?x, 2, 3)*** uses the core SWRL built-in method to add two literals.
- If ***x*** is unbound when this built-in is invoked, it will be **assigned** the value 5
- If ***x*** is already bound when the built-in is invoked, it will simply **determine** if its value is 5.

- A built-in method that successfully assigns a value to an argument should return ***true***.
- If more than one unbound argument is present, all arguments must be bound.
- If the built-in returns ***false*** no assignments are expected.

- Calculate the area of a rectangle

  Rectangle(?r) ^ hasWidthInMeters(?r, ?w) ^ hasHeightInMeters(?r, ?h) ^ swrlb:multiply(?areaInSquareMeters, ?w, ?h) -> hasAreaInSquareMeters(?r, ?areaInSquareMeters)

Binding Arguments

- The use of a variable in any non built-in atom automatically ensures it is bound (assuming the rule fires).

- Classify a rectangle with an area of over 100 square meters as a ***BigRectangle***:

  Rectangle(?r) ^ hasWidthInMetres(?r, ?w) ^ hasHeightInMetres(?r, ?h) ^ swrlb:multiply(?areaInSquareMeters, ?w, ?h) ^ swrlb:greaterThan(?100, ?areaInSquareMeters) -> hasAreaInSquareMetres(?r, ?areaInSquareMeters) ^ BigRectangle(?r)

- This rule illustrates the binding of a variable by one built-in and its subsequent use by another built-in in the same rule. The variable ***areaInSquareMeters*** is unbound when it is passed to the ***multiply*** built-in so a value is assigned to it; when the ***greaterThan*** built-in is called, it is passed this bound value.

OWL Class Expressions in SWRL Rules

- SWRL also supports the use of OWL class descriptions in rules.
- Classify an individual as a parent if it is a member of a class with a hasChild property with a minimum cardinality of one:
  (hasChild >= 1)(?x) -> Parent(?x)
- It is important to note that this rule does not state that all individuals with a child are parents. It says, instead, that all individuals that are members of an OWL class with the restriction that its hasChild property has a minimum cardinality of one. Individuals with no **known** children may be classified as parents with this rule.
- Assert conclusions about individuals:
  Parent(?x) ->(hasChild >= 1)(?x)
  Publication(?p) ^ (hasAuthor = 1)(?p) -> SingleAuthorPublication(?p)
- SWRL adopts the **Open World Assumption** from OWL

Further Information about SWRL

- Protégé SWRLLanguageFAQ: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ#nid6ZK,

OWL/SWRL integration

Human Readable Syntax
hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) ⇒ hasUncle(?x1,?x3)

XML Concrete Syntax

```
<ruleml:imp>
<ruleml:_rlab ruleml:href="#example1"/>
<ruleml:_body>
<swrlx:individualPropertyAtom  swrlx:property="hasParent">
                <ruleml:var>x1</ruleml:var>
                <ruleml:var>x2</ruleml:var>
</swrlx:individualPropertyAtom>
<swrlx:individualPropertyAtom swrlx:property="hasBrother">
                <ruleml:var>x2</ruleml:var>
                <ruleml:var>x3</ruleml:var>
</swrlx:individualPropertyAtom> </ruleml:_body>
<ruleml:_head> <swrlx:individualPropertyAtom
swrlx:property="hasUncle">
                <ruleml:var>x1</ruleml:var>
                <ruleml:var>x3</ruleml:var>
</swrlx:individualPropertyAtom> </ruleml:_head>
</ruleml:imp>
```