Semantic Knowledge Graphs

Veronika Heimsbakk

Semantic Knowledge Graphs

Veronika Heimsbakk
Knowledge Graph Lead | I&D Nordics

veronika.heimsbakk@capgemini.com

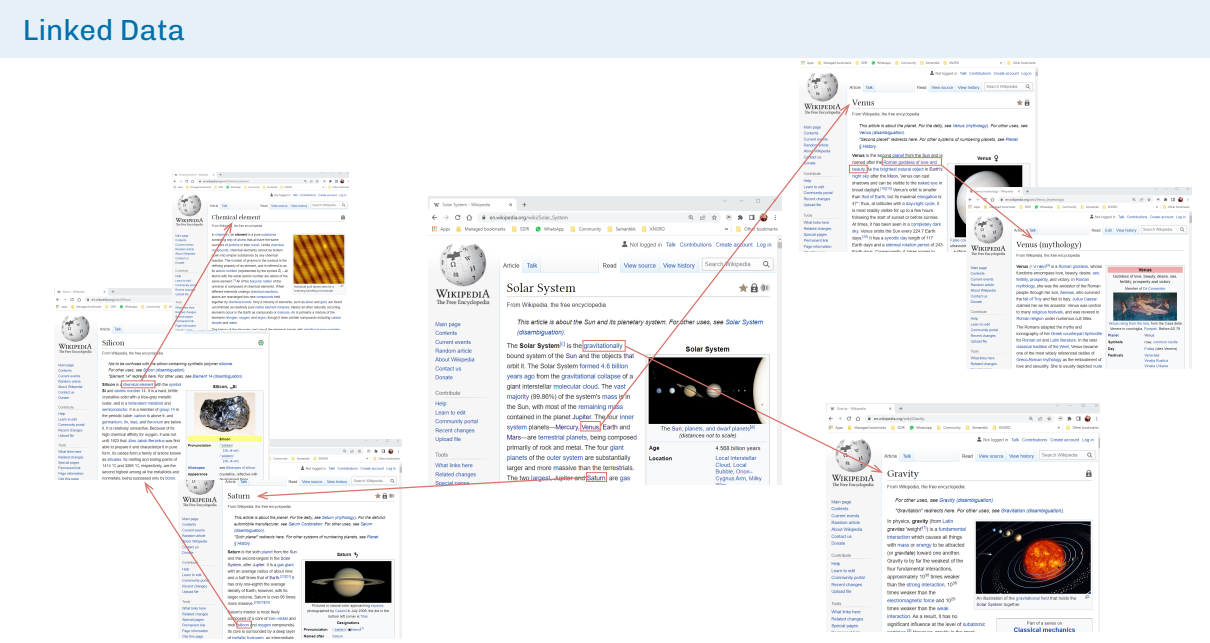
veleda
veronikaheim
veronahe.no

Capgemini

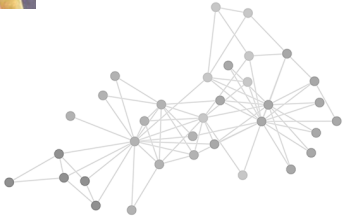
Observations

Observations

Linked Data


Chemical element
Silicon
Solar System
Solar System
Saturn
Venus
Venus (mythology)
Gravity

Linked Data



Semantics?

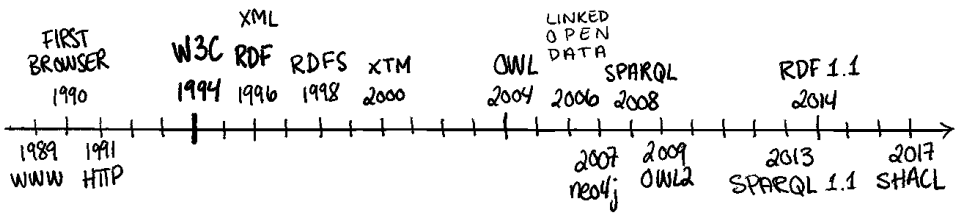
What about JSON?

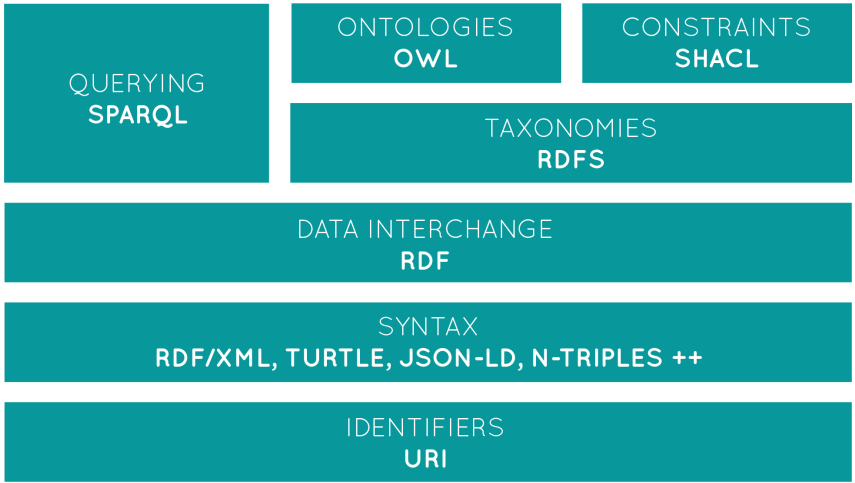
```
{
  "planet": "Earth" ,
  "moon": "Moon" ,
  "averageOrbitalSpeed": "29.78"
}
```

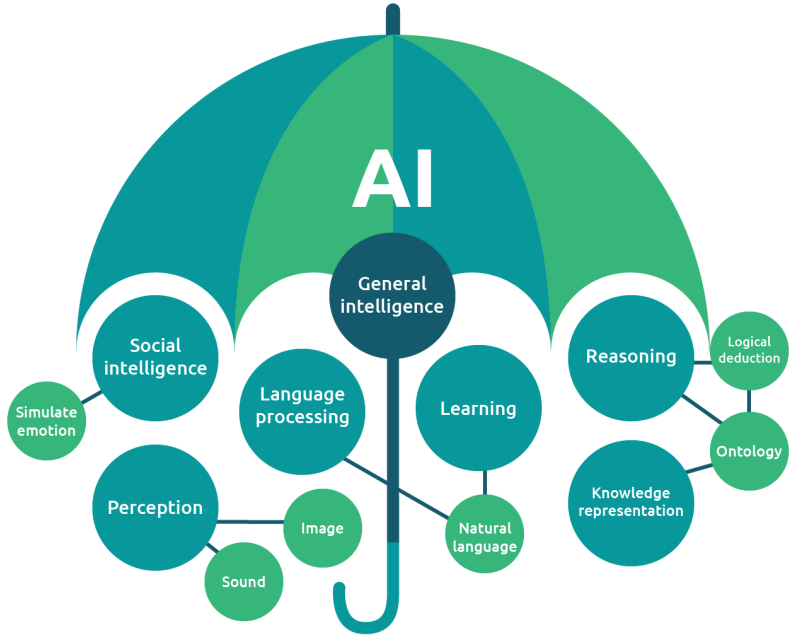
```
{
  "bolygó": "Föld" ,
  "hold": "Hold" ,
  "átlagosKeringésiSebesség": "29.78"
}
```


link
meaning
URN
new
web
translate
structure
question
DL
content
W3C
query
ontology
relations
link
data
information
meaning
RDF
rules
entailment
DL
reasoning
OWL
URI
web
classes
RDFS
knowledge
link
relevant
URI
concept
code
connections
RDF
web
RDF
OWL
relations
logic
information
URN
different
knowledge
need
concept
data
statements
relations
rules
code
URN
technologies
semantic
query
information
terms
reasoning
vocabulary
SPARQL
language
content
URI
structure
token
pizza
new
web
link
properties
facts
foaf
URN
search
word
URI
RDFS

TIMELINE OF GRAPH ON THE WEB




QUERYING
SPARQL
ONTOLOGIES
OWL
CONSTRAINTS
SHACL
TAXONOMIES
RDFS
DATA INTERCHANGE
RDF
SYNTAX
RDF/XML, TURTLE, JSON-LD, N-TRIPLES ++
IDENTIFIERS
URI


AI
General intelligence
Social intelligence
Simulate emotion
Language processing
Learning
Reasoning
Logical deduction
Ontology
Knowledge representation
Perception
Image
Sound
Natural language

Resource Description Framework

Think about data as a *directed graph*, and that all *things* has a relation to other things.

Terminology

Think about data as a *directed graph*, and that all *things* has a relation to other things.

- A model for describing data as directed graphs.

- Data described as *triples*.
- A triple is also called a *fact* or a *statement*.
- The elements of a triple are also called *resources*.

```
subject predicate object
```

- Use of *Uniform Resource Identifiers* (URI) to tell resources from another.

- Is **only** a *name*. Does not need to link to anything.

```
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
```

Example

```
http://capgemini.data.com/planets/
```

RDF syntax

There a several ways of writing RDF.

- Turtle, N-triples, RDF/XML, JSON-LD ...

easyrdf.org/converter

RDF Turtle

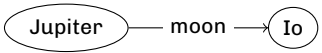
```
<http://capgemini.data.com/planets/Jupiter>
  <http://capgemini.data.com/planets/moon>
  <http://capgemini.data.com/planets/Io> .
```

RDF Turtle, prefix

```
@prefix planets: <http://capgemini.data.com/planets/> .

planets:Jupiter
  planets:moon planets:Io .
```

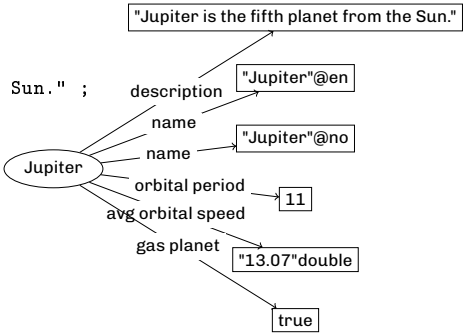
RDF Turtle, prefix

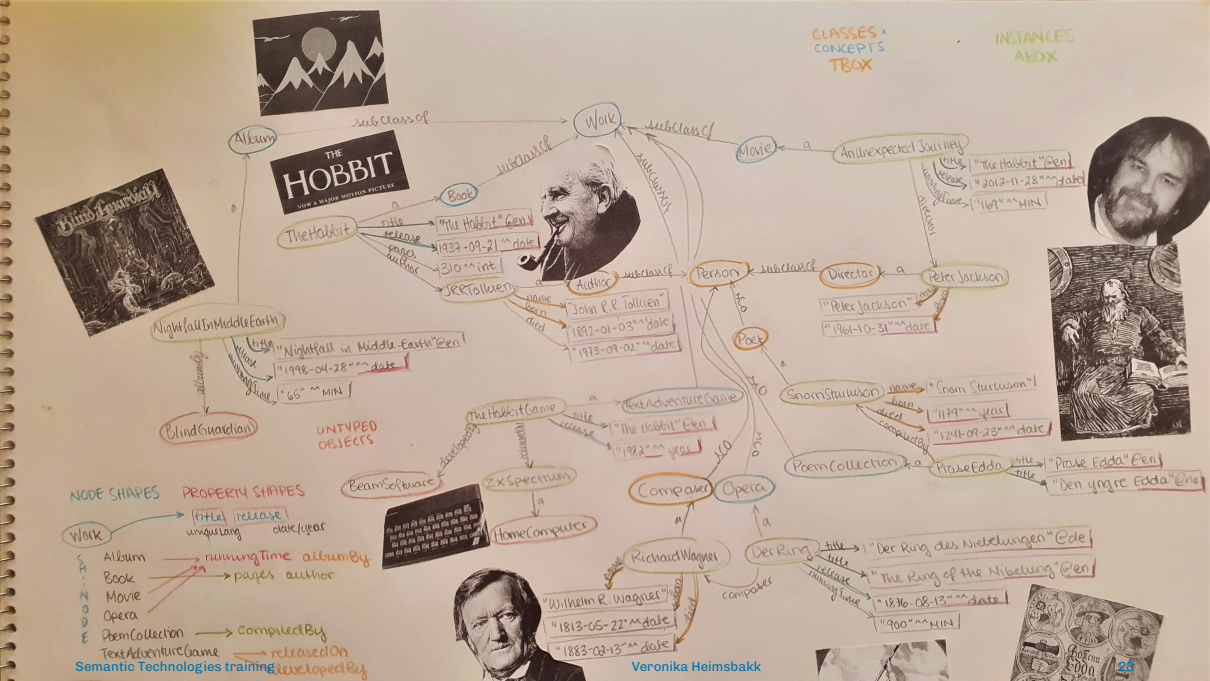
```
@prefix : <http://capgemini.data.com/planets/> .

:Jupiter
  :moon :Io .
```



RDF Turtle, literals

```
@prefix : <http://capgemini.data.com/planets/> .

:Jupiter
  :description "Jupiter is the fifth planet from the Sun." ;
  :name "Jupiter"@no ,
        "Jupiter"@de ;
  :orbitalPeriod 11 ;
  :avgOrbitalSpeed "13.07"^^xsd:double ;
  :gasPlanet true .
```

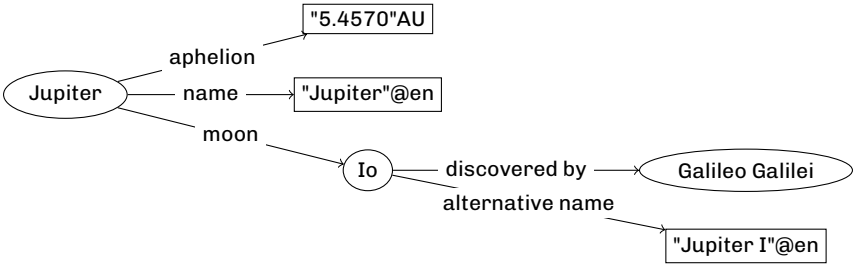



CLASSES & CONCEPTS
TBOX
INSTANCES
ABOX
Work
subClassOf
Album
Book
Movie
AnUnexpectedJourney
"The Hobbit" @en
"2012-11-28"^^date
"169" ^^MIN
TheHobbit
"The Hobbit" @en
"1937-09-21"^^date
"310"^^int
JRRTolkien
Author
Person
Director
PeterJackson
"Peter Jackson"
"1961-10-31"^^date
"John R.R. Tolkien"
"1892-01-03"^^date
"1973-09-02"^^date
NightfallInMiddleEarth
"Nightfall in Middle-Earth"@en
"1998-04-28"^^date
"65" ^^MIN
BlindGuardian
Poet
SnorriSturluson
"Snorri Sturluson"
"1179"^^year
"1241-09-23"^^date
PoemCollection
ProseEdda
"Prose Edda"@en
"Den yngre Edda"@no
UNTYPED OBJECTS
TheHobbitGame
TextAdventureGame
"The Hobbit" @en
"1982"^^year
BeamSoftware
ZxSpectrum
HomeComputer
Composer
Opera
RichardWagner
DerRing
"Der Ring des Nibelungen"@de
"The Ring of the Nibelung"@en
"1876-08-13"^^date
"900"^^MIN
WilhelmR.Wagner
"1813-05-22"^^date
"1883-03-13"^^date
NODE SHAPES
PROPERTY SHAPES
Work
title
release
uniqueLang
date/year
SHAPE
Album
Book
Movie
Opera
PoemCollection
TextAdventureGame
runningTime
albumBy
pages
author
compiledBy
releasedOn
developedBy

Exercises

40 min

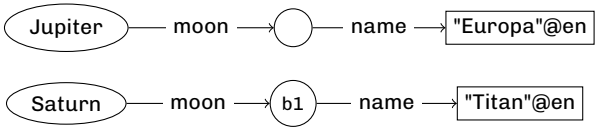
- Draw a knowledge graph in your domain of choice!



- **Remember** draw objects (defined by URIs) as circles, and data values (literals) as rectangles.
- Remember to type your data values.

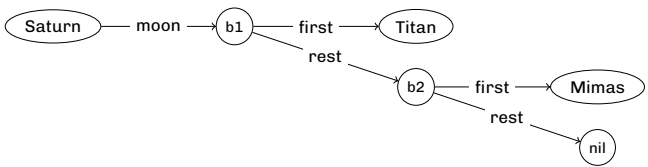
RDF Turtle, blank nodes

```
@prefix : <http://capgemini.data.com/planets/> .

#1
:Jupiter
  :moon [
    :name "Europa"@en .
  ] .

#2
:Saturn
  :moon _:b1 .

_:b1 :name "Titan"@en .
```



RDF Turtle, collections

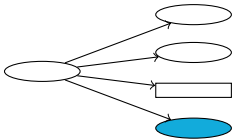
```
@prefix : <http://capgemini.data.com/planets/> .

#1
:Jupiter :moon ( :Europa :Io :Callisto ) .

# 2
:Saturn :moon _:b1 .

_:b1 rdf:first :Titan .
_:b1 rdf:rest _:b2 .

_:b2 rdf:first :Mimas .
_:b2 rdf:rest rdf:nil .
```

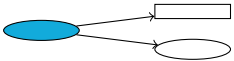


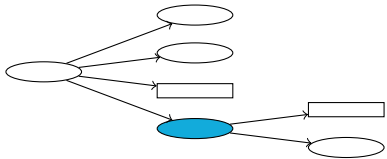
Vocabularies

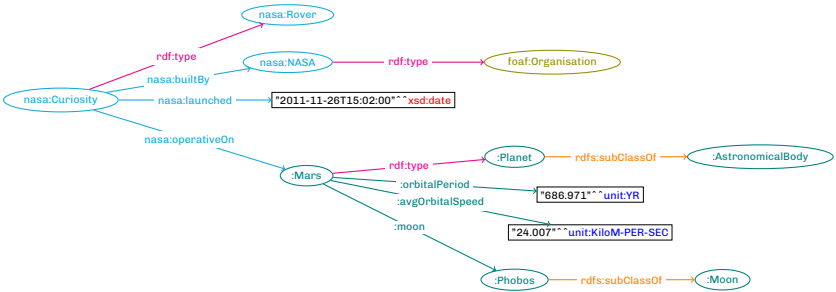
- Groups of related terms are gathered in *vocabularies*.
- Usually under one *namespace*.
- Important, and well known namespaces:
  - RDF
  - RDFS
  - DCTerms
  - FOAF
  - XSD
  - Unit
  - FIBO
  - DCAT
  - ...

`https://prefix.cc/`
Twitter shoutout









prefix : for resuorces from our dataset.
prefix **nasa:** for resuorces from NASA.
prefix **foaf:** for resuorces from Friend-of-a-friend.
prefix **rdf:** for resuorces from RDF.
prefix **rdfs:** for resuorces from RDFS.
prefix **unit:** for resuorces from Qudt Unit.
prefix **xsd:** for resuorces from XSD.

Classifying knowledge graphs

RDF Schema

- Adds the concept of *classes*, sets of resources.
- Predefined vocabulary that let us describe this sets.

```
:AstronomicalBody a rdfs:Class .
:Planet rdfs:subClassOf :AstronomicalBody .
:Jupiter rdf:type :Planet .
```

Classes

`rdfs:Resource`	The class of *everything*.
`rdfs:Class`	Declares a resource as a class for other resources.

```
rdfs:Class rdf:type rdfs:Class
```

`rdfs:Literal`	Plain or types literal values.
`rdfs:Datatype`	Instance of `rdfs:Class`, and subclass of `rdfs:Class` and `rdfs:Literal`.
`rdf:Property`	Class of properties.

Properties in RDFS

Instances & hierarchy

`rdf:type`	State that resource is an *instance* of a class.
`rdfs:subClassOf`	All resources related by one property are also related by another.
`rdfs:subPropertyOf`	All resources related by one property are also related by another.

Annotation properties

`rdfs:label`	Short title of resource.
`rdfs:comment`	Broader description of resource.

References

`rdfs:seeAlso`	Resource that might provide additional information about the subject.
`rdfs:isDefinedBy`	Indicate a resource defining the subject resource.

Property Semantics

`rdfs:domain` declares class of the *subject* in a triple where this predicate is present.

`rdfs:range` declares class or datatype of the *object* in a triple where this predicate is present.

```
:moon rdfs:domain :Planet .
:moon rdfs:range :Moon .

:Jupiter :moon :Io .
```

Inference

- Adds the concept of *classes*, sets of resources.
- Predefined vocabulary that let us describe this sets.

```
:Planet rdfs:subClassOf :AstronomicalBody .
:Jupiter rdf:type :Planet .
```

```
:Jupiter rdf:type :AstronomicalBody .
```

$\forall a, b, c \in X : (aRb \wedge bRc) \Rightarrow aRc$

Semantics!

- The *meaning* of the vocabulary is given through *inference rules*.
- May draw logic conclusions from premisses to conclusion.

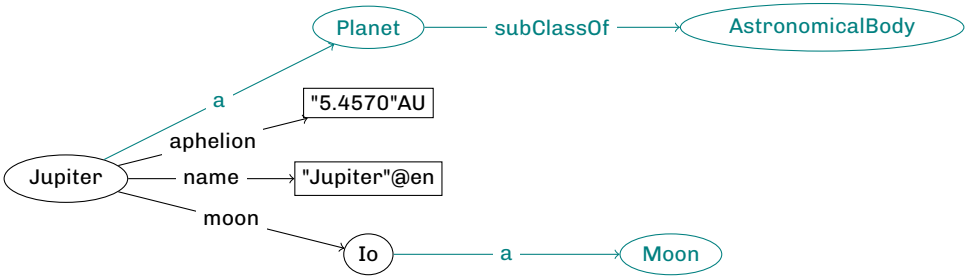
$$\frac{\texttt{C rdfs:subclassOf D} \qquad \texttt{x rdf:type C}}{\texttt{x rdf:type D}}$$

- There exists 20 inference rules to draw these logic conclusions.

Exercises

15 min

- Classify your knowledge graph drawing.



Storing and querying graphs

- Data stored in a *triplestore* or *graph database*.
- Query with **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage (SPARQL).
- Some triplestores does inference/reasoning on your data.
- Beware: different triplestores may have different interpretation of SPARQL.

- https://en.wikipedia.org/wiki/Comparison_of_triplestores
- https://db-engines.com/en/ranking/rdf+store

SPARQL

- SPARQL 1.0 recommendation in 2008, SPARQL 1.1 in 2013.
- Queries in the form of graph patterns.
- Keywords similar to other query languages.
- Advanced queries for filtering conditions.
- Advanced queries for formatting final output.

`https://www.w3.org/TR/rdf-sparql-query/`

Simple example

```
PREFIX : <http://capgemini.data.com/planets>
SELECT ?planet ?name
WHERE {
  ?planet a :Planet ;
    :name ?name .
}
```

`PREFIX` declares namespace prefix

`SELECT` general result format

`WHERE` actual query

Query

```
SELECT ?planet
WHERE {
  ?planet a :Planet .
}
```

```
SELECT *
WHERE {
  ?planet a :Planet .
}
```

?planet
:Mercury
:Venus
:Earth
:Mars
:Jupiter
:Saturn
:Uranus
:Neptune

More keywords

`FROM`	RDF dataset to query
`DISTINCT`	eliminate duplicates
`ORDER BY`	re-arrange result, related to `LIMIT` and `OFFSET`
`OPTIONAL`	alternative parts of a query
`UNION`	combine graphs
`REDUCE`	remove some or all duplicates
`FILTER`	filter result with numerical functions, regex and more

Filtering

```
SELECT ?planet ?aphelion
WHERE {
  ?planet a :Planet ;
      :aphelion ?aphelion .

  FILTER (?aphelion > "10.0"unit:AU)
}
```

?planet	?aphelion
:Saturn	10.07
:Uranus	20.09
:Neptune	30.32

Functions and operations

Binary	`||`, `&&`, `=`, `!=`, `<`, `>`, `<=`, `>=`, `+`, `-`, `*`, `/`
Unary	`!`, `+`, `-`
Unary tests	`BOUND(?var)`, `isURI(?var)`, `isBLANK(?var)`, `isLITERAL(?var)`
String	`STR(?var)`, `LANG(?var)`, `langMATCHES(LANG(?var), "lang")`, `REGEX(?var, "pattern")`
Other	`DATATYPE(?var)`, `sameTERM(?var)`

Aggregation

```
SELECT ?name (COUNT(?moon) AS ?moonCounter)
WHERE {
  ?planet a :Planet ;
    :name ?name ;
    :moon ?moon .
  ?moon a :Moon .
  FILTER (langMATCHES(LANG(?name), "is"))
}
GROUP BY ?name HAVING (?moonCounter > 20)
```

?name	?moonCounter
Júpíter	79
Satúrnus	62
Úranus	27

Optional

```
SELECT ?planet (COUNT(?moon) AS ?moonCounter)
WHERE {
  ?planet a :Planet .
  OPTIONAL { ?planet :moon ?moon . }
}
```

?planet	?moonCounter
:Mercury	
:Venus	
:Earth	1
:Mars	2
:Jupiter	79
:Saturn	62
:Uranus	27
:Neptune	14

SPARQL 1.1

New keywords for handling graph update and management introduced.

- `INSERT`, `DELETE`, `LOAD`, `CLEAR`
- `CREATE`, `DROP`, `COPY`, `MOVE`, `ADD`

Note! Not all triplestores contain support for all SPARQL 1.1 names.

End of workshop

part 1

Workshop
part 2

Exercises

30 min

- Download the open source graph database Fuseki.
  - `https://jena.apache.org/documentation/fuseki2/`
  - Run .bat-file to start the database on localhost:3030
- Do a sanity check of your RDF with `https://www.easyrdf.org/converter`.
- Upload data into Fuseki and play around with SPARQL.
  - e.g. `SELECT * WHERE { ?s ?p ?o . }`

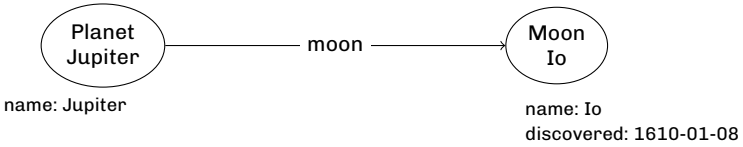
Trouble setting up and running Fuseki? Explore SPARQL using `https://dbpedia.org/sparql`

Summary of Labelled Property Graphs (LPG)

Think about data as a *directed graph*, and that all *things* has a relation to other things.

- *Nodes* are entities in the graph.
- A node may hold any number of *properties* (key-value pairs).
- Nodes may be tagged with *labels*.
- *Relationships* provide directed, named connections between two nodes.
- Relationships may also contain properties.

Summary of LPG




```
CREATE(Planet:Jupiter { name:"Jupiter".})
CREATE(Moon:Io {
         name:"Io",
         discovered:"1610-01-08".})

CREATE(Planet:Jupiter)-[moon]->(Moon:Io)
```

Using Neo4j Cypher

RDF vs LPG

RDF	LPG
Directed graph.	Directed graph.
Designed for data exchange.	Designed for fast querying & storage.
Atomic breakdown of data.	Internal data structures (key-value properties).
Denoted using URIs.	Differs.
Standardized by W3C.	No standardizations.


STARDOG
ontotext
AllegroGraph
Franz Inc.
RDFOX
Amazon
Neptune
neo4j
ANZO
Azure
Cosmos DB
TigerGraph
ArangoDB
JanusGraph

Some more terminology...

TBox & ABox

TBox

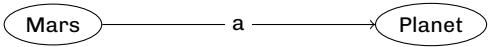
TBox (terminological component)

- Sets of individuals.
- Classes and concepts.

ABox

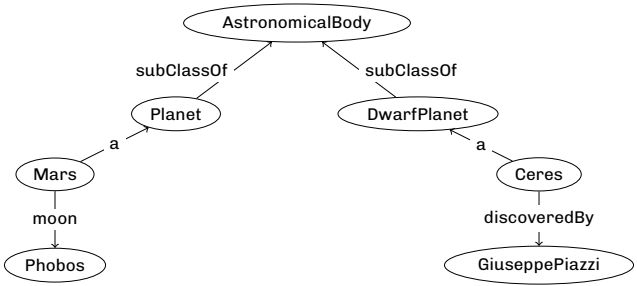
ABox (assertion component)

- Individuals and instances.



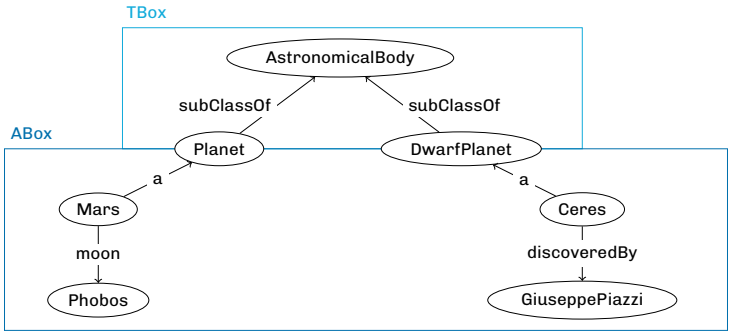
Knowledge graph

TBox + ABox = Knowledge graph ❤



Knowledge graph

TBox + ABox = Knowledge graph ❤



Exercises

10 min

- Identify TBox and ABox in your drawing.

Open world assumption & closed world assumption

Open world assumption (OWA)

- Admits incomplete knowledge.
- Ontologies with Web Ontology Language (OWL).

The assumption that the truth value of a statement may be true irrespective of whether or not it is known to be true.

Example

Statement:	Mars is a planet in the Solar System.
Question:	Is Proxima Centauri a planet in the Solar System?
OWA:	Unknown

Closed world assumption (CWA)

- Shape constraints with Shape Constraint Language (SHACL).

Any statement that is true is known to be true. What is not currently known to be true is false.

Example

Statement:	Mars is a planet in the Solar System.
Question:	Is Proxima Centauri a planet in the Solar System?
CWA:	No

Web Ontology Language

Ontology?

An ontology is an explicit specification of a conceptualization.

- The study of existence, categories and relations between what *is*.
- Formal names and definitions of entities, properties and relations.
- Web Ontology Language (OWL) is a collection of languages to represent *knowledge*.

Web Ontology Language

- Released in 2004, OWL2 in 2009.
- More expressive knowledge representation than RDFS.
- Designed for inference and reasoning, not validation.
- Open world assumption.

OWL syntax

OWL	$\mathcal{ALC}$	**Manchester**
`intersectionOf`	$C \sqcap D$	C **AND** D
`unionOf`	$C \sqcup D$	C **OR** D
`complementOf`	$\neg C$	**NOT** C
`oneOf`	$\{a\} \sqcap \{b\} \ldots$	{a b ...}
`someValuesFrom`	$\exists RC$	R **SOME** C
`allValuesFrom`	$\forall RC$	R **ONLY** C
`minCardinality`	$\geq NR$	R **MIN** 3
`maxCardinality`	$\leq NR$	R **MAX** 3
`cardinality`	$= NR$	R **EXACTLY** 3
`hasValue`	$\exists R\{a\}$	R **VALUE** a
`NegativePropertyAssertion`	$\neg R(a, b)$	a **NOT** R b
`hasSelf`	$\exists R.Self$	**SELF**

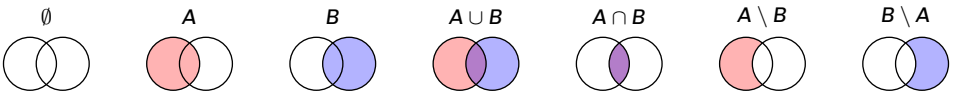
Mathematical properties and operations

May assert characteristics on resources to tell how relations behave.

R is a relation on the set *X*(*R SUB X* $\times$ *X*), and R are

- reflexive, if $\langle a, a \rangle \in R$ for all $a \in X$
- irreflexive, if $\langle a, a \rangle \notin R$ for all $a \in X$
- symmetric, if $\langle a, b \rangle \in R$ implies $\langle b, a \rangle \in R$
- asymmetric, if $\langle a, b \rangle \in R$ implies $\langle b, a \rangle \notin R$
- transitive, if $\langle a, b \rangle, \langle b, c \rangle \in R$ implies $\langle a, c \rangle \in R$
- functional, if $\langle a, b \rangle, \langle a, c \rangle \in R$ implies $b = c$

Set theory



Properties in OWL

OWL introduce three mutually disjoint properties.

`owl:DatatypeProperty`	All properties where `rdfs:range` are a typed literal.
`owl:ObjectProperty`	All properties where `rdfs:range` is an URI.
`owl:AnnotationProperty`	All properties meant for annotation, no logical implications.

Object properties may be

transitive	asymettric	symmetric	equvalent
reflexive	irreflexive	functional	disjoint

Datatype properties may be functional, equivalent or disjoint.

Individuals

`owl:NamedIndividual` instance of a class, but not a class itself.

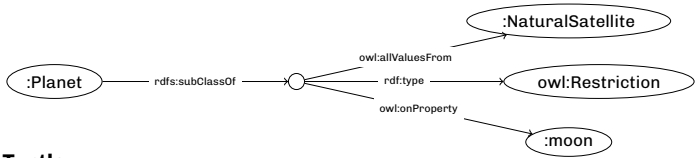
Defined by axioms/facts:

- facts on class relations
- facts on the individual's identity

ALC

Planet $\sqsubseteq \forall$ moon.NaturalSatellite

Visualization



Turtle

```
:Planet
  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :moon ;
    owl:allValuesFrom :NaturalSatellite
  ] .
```

OWL functional

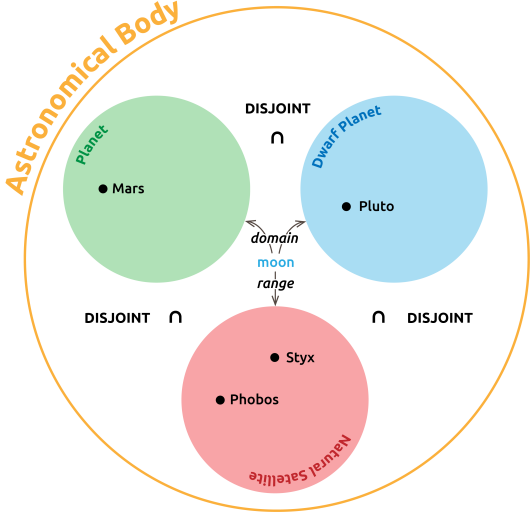
```
SubClassOf(Planet ObjectAllValuesFrom(moon NaturalSatellite))
```

Exercises

30 min

- Download the ontology modelling tool Protégé.
  - `https://protege.stanford.edu/`
- Plot your drawing into Protégé.

Demo time!

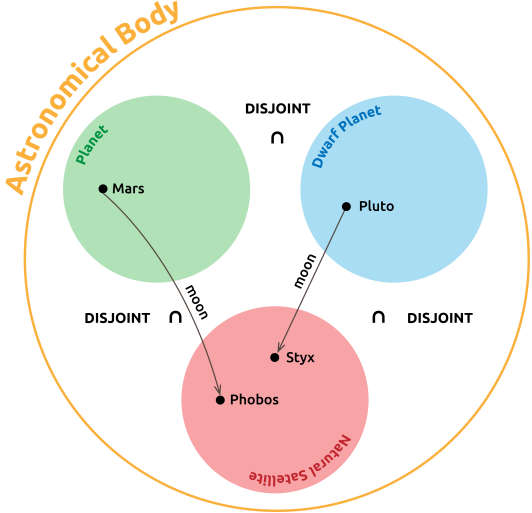


Classes
Planet
Dwarf Planet
Natural Satellite

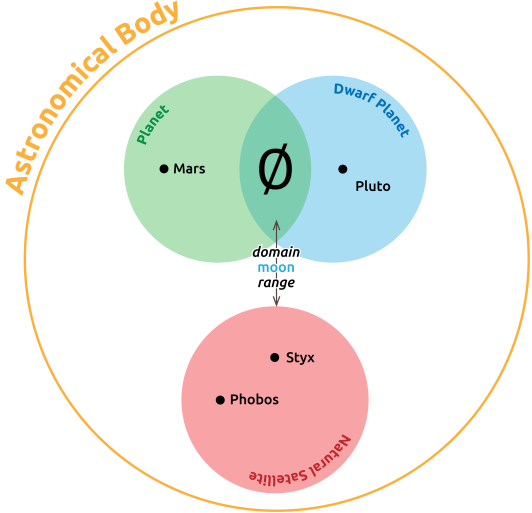
Individuals
Planet(Mars)
Dwarf Planet(Pluto)
Natural Satellite(Phobos)
Natural Satelitte(Styx)

Property
moon⟨⟨Planet, Dwarf Planet ⟩, Natural Satellite⟩

Axioms
Planet ∩ Dwarf Planet
Planet ∩ Natural Satellite
Dwarf Planet ∩ Natural Satellite



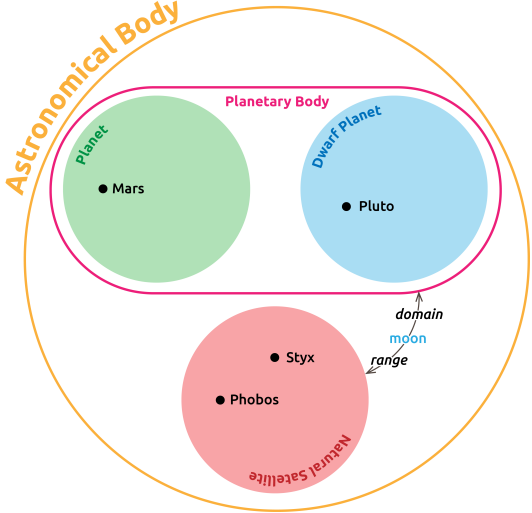
moon⟨Mars, Phobos⟩
moon⟨Pluto, Styx⟩



If we keep the domain to two classes, we say that the domain must be an intersection between the two.

The intersection between two disjoint classes will always be the empty set (∅).

Therefore, we can never use the property moon, as the domain is the empty set.



Solution

Introduce a new abstraction level; Planetary Bode. Update domain of moon to be this new class.

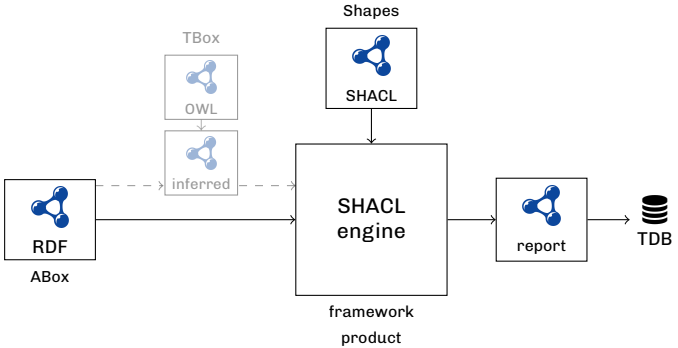
Shape Constraint Language

A language for describing and validating RDF graphs

Validation of RDF – a brief history

- Prior to SHACL; no W3C standard for validating RDF.
- SPARQL Inferencing Notation (SPIN), IBM Resource Shapes, Shape Expressions (ShEx)
- W3C recommendation in July 2017.

Workflow


Shapes
TBox
SHACL
OWL
inferred
RDF
ABox
SHACL
engine
framework
product
report
TDB

Comparing SHACL and OWL

Common

- RDF & URIs
- Infer new triples
- Rely on RDF Schema (RDFS)

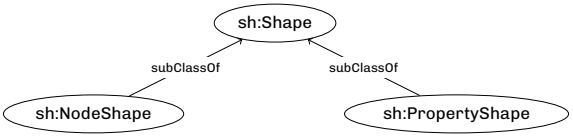
Difference

OWL	SHACL
Designed for inference	Designed for validation
Open world assumption	Closed world assumption
Limited features	Add your own constraints
	SHACL resourses are distinct by default
Logical contradictions	Conforms to given schema

SHACL Shape

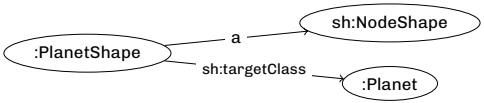
A collection of constraints for given RDF resource.

- Shapes about focus nodes (**sh:NodeShape**).
- Shapes about values of a property or path for the focus node (**sh:PropertyShape**).



sh:NodeShape

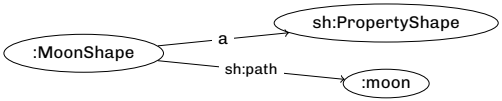
A *node shape* is a shape that is **not** the subject of a triple with *sh:path* as its predicate.




```
:PlanetShape
  a sh:NodeShape ;
  sh:targetClass :Planet .
```

sh:PropertyShape

A *property shape* is a shape that is the subject of a triple that has *sh:path* as its predicate.




```
:MoonShape
  a sh:PropertyShape ;
  sh:path :moon .
```

BookShape

```
:PlanetShape
  a sh:NodeShape ;
  sh:targetClass :Planet ;
  sh:property :MoonShape .

:MoonShape
  a sh:PropertyShape ;
  sh:path :moon .
```

SHACL Core Constraint Components

SHACL Core Constraint Components

Value type

sh:class	Each value node is an instance of a given type.
sh:datatype	Datatype of each value node.
sh:nodeKind	Node kind (IRI, blank node etc.) of each value node.

```
:PlanetShape
  a sh:NodeShape ;
  sh:targetClass :Planet ;
  sh:property [
    sh:path :aphelion ;
    sh:datatype unit:AU ;
  ] .
```

SHACL Core Constraint Components

Cardinality

sh:minCount	Minimum cardinality as xsd:integer.
sh:maxCount	Maximum cardinality as xsd:integer.

Value range

sh:minExclusive	$x <$ $value
sh:minInclusive	$x <=$ $value
sh:maxExclusive	$x >$ $value
sh:maxInclusive	$x >=$ $value

```
:StarShape
  a sh:NodeShape ;
  sh:targetClass :Star ;
  sh:property [
    sh:path :kelvinTemperature ;
    sh:minInclusive 2500 ;
    sh:maxExclusive 50000 ;
  ] .
```

SHACL Core Constraint Components

String-based

sh:minLength	Minimum length as xsd:integer.
sh:maxLength	Maximum length as xsd:integer.
sh:pattern	Regular expression.
sh:languageIn	A list of languages as per RFC5646.
sh:uniqueLang	One unique tag per language.

```
:PlanetShape
  a sh:NodeShape ;
  sh:targetClass :Planet ;
  sh:property [
    sh:path :name ;
    sh:languageIn ("en" "no" "de") ;
  ] .
```

SHACL Core Constraint Components

Property pair	Compare two IRIs where,
sh:equals	$x \equiv y$
sh:disjoint	$x \cap y = \emptyset$
sh:lessThan	$x < y$
sh:lessThanOrEquals	$x <= y$

```
:PlanetShape
  a sh:NodeShape ;
  sh:targetClass :Planet ;
  sh:property [
    sh:path :moon ;
    sh:disjoint :satellitesInOrbit ;
  ] .
```

SHACL Core Constraint Components

Logical	List of value nodes that,
sh:not	Cannot conform to given shape.
sh:and	Conforms to all provided shapes.
sh:or	Conforms to at least one of the provided shapes.
sh:xone	Conforms to exactly one of the provided shapes.

```
:PlanetShape
  a sh:NodeShape ;
  sh:targetClass :Planet ;
  sh:or (
    [ sh:path :aphelion ; sh:minCount 1 ; ]
    [ sh:path :perihelion ; sh:minCount 1 ; ]
  ) .
```

```
:PlanetShape
  a sh:NodeShape, owl:Class ;
  sh:property [
    sh:path :moon ;
    sh:or (
      [ sh:class :Moon ]
      [ sh:datatype xsd:string ]
    )
  ] .
```

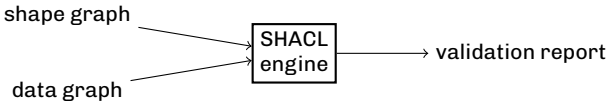
SHACL Core Constraint Components

Shape-based	Each value node,
sh:node	Conforms to the given node shape.
sh:property	Has a given property shape.

Other	
sh:closed	Boolean signalising a complete shape.
sh:ignoredProperties	List of properties to ignore.
sh:hasValue	At least one value node is equal to the given term.
sh:in	Value node is member of given list.

```
:PlanetShape
  a sh:NodeShape ;
  sh:targetClass :Planet ;
  sh:closed true ;
  sh:ignoredProperties (rdf:type) .
```

SHACL engine



Validation report

Each instance of *sh:ValidationReport* has exactly one value of *sh:conforms*.

sh:conforms is true iff the validation did not produce any **validation results**, and false otherwise.

Iff validation conforms false, the report will contain an instance of **sh:ValidationResult**.

```
[
  a sh:ValidationReport ;
  sh:conforms true ;
] .
```

Validation result

All properties described can be specified in a validation result.

sh:focusNode	Node that caused the result.
sh:resultPath	Pointing to value of **sh:path**
sh:value	Value node that violated constraint.
sh:sourceShape	Shape that given focus node validated against.
sh:sourceConstraintComponent	Constraint component that caused the result.
sh:detail	Parent result containing more details about the violation.
sh:message	Annotation property with textual details.
sh:severity	Default **sh:Violation**.

Other nice to knows about SHACL

- Deactivating shapes
- Non-validating property shape characteristics
  - sh:name & sh:description
  - sh:order & sh:group
  - sh:defaultValue
- Syntax checking of shapes graph

Sources

W3C Standards	
RDF	https://www.w3.org/TR/rdf11-primer/
RDFS	https://www.w3.org/TR/rdf-schema/
SPARQL	https://www.w3.org/TR/rdf-sparql-query/
OWL	https://www.w3.org/TR/2012/REC-owl2-primer-20121211/
SHACL	https://www.w3.org/TR/shacl/
Tools	
EasyRDF Converter	https://www.easyrdf.org/converter
Protégé	https://protege.stanford.edu/
Fuseki	https://jena.apache.org/documentation/fuseki2/
SHACL Playground	https://shacl.org/playground/
Tutorials	
SPARQL	https://kvistgaard.github.io/sparql/
SHACL	https://github.com/veleda/shacl-masterclass
OTTR	https://github.com/veleda/ottr-masterclass