Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

Sematic Web

Reasoning

Holger Wache
SS 2008

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

Lecture Outline

1. Axiomatic Semantics for RDF and RDFS
2. Direct Semantics based on Inference Rules
3. Querying of RDF/RDFS Documents using RQL

Copyright notice

- Some slides are taken from the “RDF” lectures by Antoine Isaac (VU, Amsterdam).
- I would like to thank him for the allowance to use his slides in this course!

The Semantics of RDF(S)

- The **semantics** of RDFS can be expressed in natural language:
  - 2.3.2 **rdfs:subClassOf** [old RDFS specs]
    “This property specifies a subset/superset relation between classes. The rdfs:subClassOf property is transitive. If class A is a subclass of some broader class B, and B is a subclass of C, then A is also implicitly a subclass of C. Consequently, resources that are instances of class A will also be instances of class C, since A is a subset of both B and C. Only instances of rdfs:Class can have the rdfs:subClassOf property and the property value is always of rdf:type rdfs:Class. A class may be a subclass of more than one class.”
- Question: is A a subclass of A?

Axiomatic Semantics

- We formalize the meaning of the modeling primitives of RDF and RDF Schema
- By translating into first-order logic
- We make the semantics unambiguous and machine accessible
- We provide a basis for reasoning support by automated reasoners manipulating logical formulas

The Approach

- All language primitives in RDF and RDF Schema are represented by constants:
  - **Resource**, **Class**, **Property**, **subClassOf**, etc.
- A few predefined predicates are used as a foundation for expressing relationships between the constants
- We use predicate logic with equality
- Variable names begin with ?
- All axioms are implicitly universally quantified

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

An Auxiliary Axiomatisation of Lists

- Function symbols:
  - **nil** (empty list)
  - **cons(x,l)** (adds an element to the front of the list)
  - **first(l)** (returns the first element)
  - **rest(l)** (returns the rest of the list)
- Predicate symbols:
  - **item(x,l)** (tests if an element occurs in the list)
  - **list(l)** (tests whether l is a list)
- Lists are used to represent containers in RDF

7

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

Basic Predicates

- **PropVal(P,R,V)**
  - A predicate with 3 arguments, which is used to represent an RDF statement with resource **R**, property **P** and value **V**
  - An RDF statement (triple) **(R, P,V)** is represented as **PropVal(P,R,V)**.
- **Type(R,T)**
  - Short for **PropVal(type,R,T)**
  - Specifies that the resource **R** has the type **T**
- **Type(?r,?t) ↔ PropVal(type,?r,?t)**

8

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

RDF Classes

- Constants: **Class**, **Resource**, **Property**, **Literal**
  - All classes are instances of **Class**

Type(Class,Class)
Type(Resource,Class)
Type(Property,Class)
Type(Literal,Class)

9

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

RDF Classes (2)

- **Resource** is the most general class: every class and every property is a resource

Type(?p,Property) → Type(?p,Resource)
Type(?c,Class) → Type(?c,Resource)

- The predicate in an RDF statement must be a property

PropVal(?p,?r,?v) → Type(?p,Property)

10

The type Property

- **type** is a property

PropVal(type,type,Property)

- **type** can be applied to resources (domain) and has a class as its value (range)

Type(?r,?c) → (Type(?r,Resource) ∧ Type(?c,Class))

The Auxiliary FuncProp Property

- **P** is a functional property if, and only if,
  - it is a property, and
  - there are no **x**, **y1** and **y2** with **P(x,y1)**, **P(x,y2)** and **y1≠y2**

Type(?p, FuncProp) ↔
(Type(?p, Property) ∧
∀?r ∀?v1 ∀?v2
(PropVal(?p,?r,?v1) ∧
PropVal(?p,?r,?v2) → ?v1 = ?v2))

Containers

- Containers are lists:

Type(?c,Container) → list(?c)

- Containers are bags or sequences or alternatives:

Type(?c,Container) ↔

(Type(?c,Bag) ∨ Type(?c,Seq) ∨ Type(?c,Alt))

- Bags and sequences are disjoint:

¬(Type(?x,Bag) ∧ Type(?x,Seq))

Containers (2)

- For every natural number n > 0, there is the selector **_n,** which selects the n^{th} element of a container
- It is a functional property:

Type(_n,FuncProp)

- It applies to containers only:

PropVal(_n,?c,?o) → Type(?c,Container)

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

Subclass

- **subClassOf** is a property:

Type(subClassOf,Property)

- If a class C is a subclass of a class C', then all instances of C are also instances of C':

PropVal(subClassOf,?c,?c') ↔
(Type(?c,Class) ∧ Type(?c',Class) ∧
∀?x (Type(?x,?c) → Type(?x,?c')))

15

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

Subproperty

- P is a subproperty of P', if P'(x,y) is true whenever P(x,y) is true:

Type(subPropertyOf,Property)

PropVal(subPropertyOf,?p,?p') ↔
(Type(?p,Property) ∧ Type(?p',Property) ∧
∀?r ∀?v (PropVal(?p,?r,?v) →
PropVal(?p',?r,?v)))

16

Domain and Range

- If the domain of P is D, then for every P(x,y), x∈D

PropVal(domain,?p,?d) →
∀?x ∀?y (PropVal(?p,?x,?y) → Type(?x,?d))

- If the range of P is R, then for every P(x,y), y∈R

PropVal(range,?p,?r) →
∀?x ∀?y (PropVal(?p,?x,?y) → Type(?y,?r))

Lecture Outline

1. Basic Ideas of RDF
2. XML-based Syntax of RDF
3. Basic Concepts of RDF Schema
4. The Language of RDF Schema
5. Axiomatic Semantics for RDF and RDFS
6. Direct Semantics based on Inference Rules
7. Querying of RDF/RDFS Documents using RQL

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

Semantics based on Inference Rules

- Semantics in terms of RDF triples instead of restating RDF in terms of first-order logic
- … and sound and complete inference systems
- This inference system consists of inference rules of the form:

IF E contains certain triples

THEN add to E certain additional triples

- where **E** is an arbitrary set of RDF triples

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

Examples of Inference Rules

IF (?x,?p,?y)
THEN (?p,rdf:type,rdf:property)

IF (?u,rdfs:subClassOf,?v),
(?v,rdfs:subclassOf,?w)
THEN (?u,rdfs:subClassOf,?w)

IF (?x,rdf:type,?u),
(?u,rdfs:subClassOf,?v)
THEN (?x,rdf:type,?v)

Examples of Inference Rules (2)

- Any resource **?y** which appears as the value of a property **?p** can be inferred to be a member of the range of **?p**
  - This shows that range definitions in RDF Schema are not used to restrict the range of a property, but rather to infer the membership of the range

IF (?x,?p,?y), (?p,rdfs:range,?u)
THEN (?y,rdf:type,?u)

RDF(S) Semantics: Examples

- IF (Netherlands, **type,** EuropeanCountry),
  (EuropeanCountry, **subClassOf,** Country)
  THEN (Netherlands, **type,** Country)

- IF (aspirin, alleviates, headache),
  (alleviates, **range,** symptom)
  THEN (headache, **type,** symptom)

RDF(S) Semantics: Examples

- IF (Νετηερλανδσ, **type,** ΕυροπεανΧουντρψ),
  (ΕυροπεανΧουντρψ, **subClassOf,** Χουντρψ)
  THEN (Νετηερλανδσ, **type,** Χουντρψ)

- IF (ασπιριν, αλλεωιατεσ, ηεαδαχηε),
  (αλλεωιατεσ, **range,** σψμπτομ)
  THEN (ηεαδαχηε, **type,** σψμπτομ)

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

Further Examples of Inference Rules

- IF (?X, ?R, ?Y), (?R, **domain,** ?T)
  THEN (?X, **type,** ?T)
- IF (?X, ?R, ?Y), (?R, **range,** ?T)
  THEN (?Y, **type,** ?T)
- IF (?T1, **subClassOf,** ?T2), (?T2, **subClassOf,** ?T3)
  THEN (?T1, **subClassOf,** ?T3)
- IF (?X, **type,** ?T1), (?T1, **subClassOf,** ?T2)
  THEN (?X, **type,** ?T2)
- IF (?X, **type, Class**)
  THEN (?X, **subClassOf,** ?X)

Lecture Outline

1. Basic Ideas of RDF
2. XML-based Syntax of RDF
3. Basic Concepts of RDF Schema
4. The Language of RDF Schema
5. Axiomatic Semantics for RDF and RDFS
6. Direct Semantics based on Inference Rules
7. Querying of RDF/RDFS Documents using RQL

Querying

- Databases: formulate a query on the relational model
  - (table z, row x, column y)
- HTML: formulate a location and get back an entire document
  - http://www.cs.vu.nl/index.html
- XML: formulate a query on the tree: *path expressions*
  - /country/geography/capital@name
- RDF?

Some Properties for QLs

- Adequacy
  - The QL makes use of all the primitives in the data model
- Expressiveness
  - "Able to formulate the questions we want to ask"
  - Operators and functions
- Closure
  - Result set in the same model as the source model
  - We can "compose" queries

Adequacy for an RDF QL

- Path expressions
  - Traverse the RDF graph, allowing both nodes and edges to be searched
- Optionals
  - RDF is semistructured: values of properties may not be specified for every instance
- Reification
- Namespaces

Adequacy for an RDFS QL

- Support for RDF schema mechanisms
  - RDFS constructs
  - Statements using these constructs
  - Awareness to RDFS formal semantics
- [XSD datatypes, Language]
- [Containers and collections]

Why not XML Query Language? Different XML Representations

- XML at a lower level of abstraction than RDF
- There are various ways of syntactically representing an RDF statement in XML
- Thus we would require several XQuery queries, e.g.
  - **//uni:lecturer/uni:title** if **uni:title** element
  - **//uni:lecturer/@uni:title** if **uni:title** attribute
  - Both XML representations equivalent!

Example in RDF: Retrieving Capital Cities

```
<rdf:Description rdf:about="#Netherlands">
   <rdf:type  rdf:resource="#Country"/>
   <geo:hasCapital rdf:resource="#Amsterdam"/>
</rdf:Description>
```

XML path expression
/rdf:Description/[rdf:type/@rdf:resource="#Country"]/geo:hasCapital/@rdf:resource

```
<geo:Country rdf:about="#Netherlands">
   <geo:hasCapital rdf:resource="#Amsterdam"/>
</geo:Country>
```

XML path expression
/geo:Country/geo:hasCapital/@rdf:resource

More possibilities if nested descriptions…

31

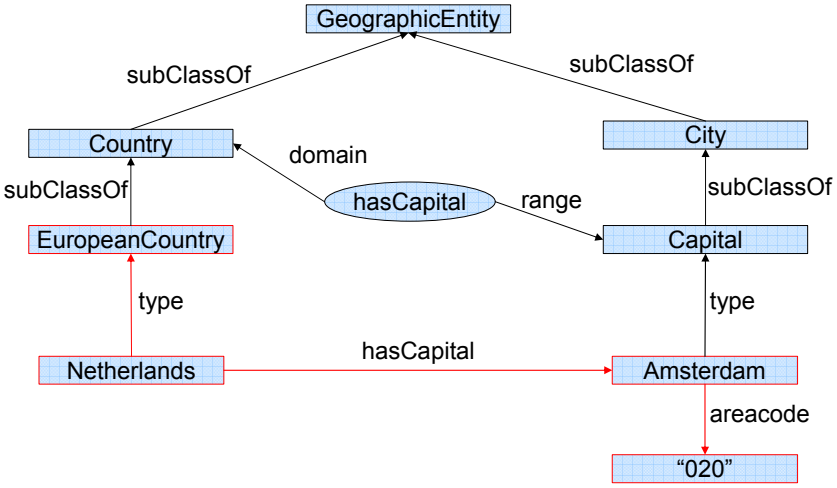
Why not XML Query Language? Understanding the Semantics

```
<uni:lecturer rdf:ID="949352">
   <uni:name>Grigoris Antoniou</uni:name>
</uni:lecturer>

<uni:professor rdf:ID="949318">
        <uni:name>David Billington</uni:name>
</uni:professor>

<rdfs:Class rdf:about="#professor">
   <rdfs:subClassOf rdf:resource="#lecturer"/>
</rdfs:Class>
```

- A query for the names of all lecturers should return both Grigoris Antoniou and David Billington

32

Paths in RDF



33

Path Expressions in RDF

- No root element (cf. XML Paths)
  Paths can start at any location in the graph
- Both nodes and edges of graph are labeled!
  More than one type of relation between nodes, not just "nested inside"
- So we need a more powerful type of path expression

34

Requirements for an RDF QL

- Understand the data model
  - directed, labeled graphs
  - semi-structured
- Path expressions
  - through the RDF graph, not the XML tree
  - specifying both node and edge labels
- Compositionality
  - complex queries can be 'built up' by combining simpler queries
- Support for RDF Schema
  - understand the semantics of subClassOf, Class, etc.

Query Language Proposals

- No standardized query language for RDF, but many proposals:
  - RQL (RDF Schema Query Language)
  - RDQL (RDF Data Query Language)
  - SeRQL (Sesame RDF Query Language)
  - SPARQL (W3C standard, being developed)

SeRQL

- Language proposal based on best practices
  - Redesign of RQL, incorporating ideas from many other query languages
  - Developed in the Sesame project
- Expressive language, but still fairly easy to use
- Support for RDF Schema
- Implementation: *Sesame RDF framework* (Aduna, VU)

SeRQL Query Syntax

SeRQL uses a select-from-where syntax (like SQL):

- *select*: the entities (variables) you want to return
  ```
  select X
  ```
- *from*: the (sub)graph you want to get the information from
  ```
  from  {X} geo:areacode {Y}
  ```
- *where*: additional constraints on objects, using operators
  ```
  where  Y like "020"
  ```
- *Using namespace*: prefix information
  ```
  using namespace
       geo=<http://www.geography.org/schema.rdfs#>
  ```

SeRQL Query Syntax

```
SELECT X
FROM  {X} geo:areacode {Y}
WHERE  Y = "020"
USING NAMESPACE
       geo =
  <http://www.geo.org/schema.rdfs#>
```

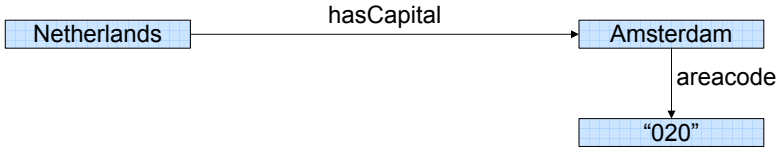
Composing complex queries from different elementary specifications

SeRQL Path Expressions

- Path expressions for the RDF model
  - Both edges and nodes can be specified
  - *Variables*
  - *Branches*: one node can have several outgoing edges
  - *Optionals*: for some objects, a value may or may not be specified

SeRQL Path Expressions

- `{X} geo:hasCapital {geo:Amsterdam}`
- `{X} geo:hasCapital {Y}`
- `{X} P {Y}`



SeRQL: Chaining and Branching

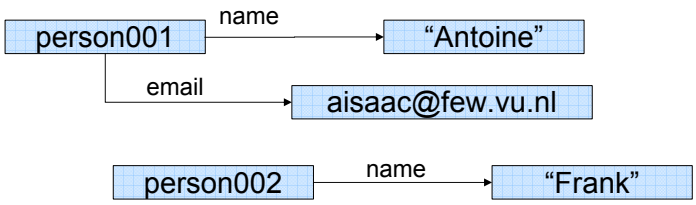
- Chaining:
  - `{X} geo:hasCapital {Y} geo:areacode {Z}`

  Equivalent to
  - `{X} geo:hasCapital {Y}, {Y} geo:areacode {Z}`
- Branching:
  - `{X} geo:name {Y};`
    `geo:areacode {Z}`

  Equivalent to
  - `{X} geo:name {Y}, {X} geo:areacode {Z}`

Optional Path Expressions

- RDF is *semi*-structured
  - Even when the schema says some object should have a particular property, it may not always be present in the data
  - Example: persons can have names and email addresses, but Frank is a person without a known email address



Optional path expressions (2)

- "give me all persons, their first names, and *if known their email address*"
- an optional path expression is needed

```
select
  Person, Name, Email
from
  {Person} my:name {Name};
           [my:email {Email}]
```

Boolean comparisons

- Comparison operators in **`where`** clause
  - *String comparison*:
    - **`WHERE X like "The Netherlands"`**
    - **`WHERE X like "*Netherlands"`**
  - *Boolean comparison*:
    - X < Y, X <= Y, Z < 20, Z = Y, etc.
  - Boolean *combination* of those operators
    - AND, OR, NOT
    - **`WHERE (Y > 10 AND Y < 30)`**
      `OR (NOT X LIKE “Rott*”)`

Boolean comparisons and datatypes

- RDF has basic datatypes for literals
  - Actually re-uses XML Schema datatypes:
    - xsd:integer, xsd:float, xsd:string
  - A datatyped literal looks like this:
    - “20”^^xsd:integer
- You can use this in SeRQL queries to compare values:
  - X < “21”^^xsd:integer

Query Results

- SeRQL select-queries return variable bindings
  - For each variable in the query, it gives a value.
  - The result is a table, where each column represents a variable and each row a set of values

Query result: example

- Query: “return all countries with the cities they contain, and their areacodes, if known”

```
select X, Y, Z
from  {X} geo:containsCity {Y} [geo:areacode {Z}]
```

- Result (table of bindings):

X	Y	Z
Netherlands	Amsterdam	“020”
Netherlands	DenHaag	“070”
France	Paris	

Query result: example

- Query: return all capital cities

```
select Y
from  {X} geo:hasCapital {Y}
```

- Result as an RDF/XML document :

```
<rs:ResultSet rdf:about=''>
    <rs:resultVariable>Y</rs:resultVariable>
    <rs:solution>
        <rs:ResultSolution>
            <rs:binding rdf:parseType='Resource'>
              <rs:variable>Y</rs:variable>
              <rs:value rdf:resource='http://www.geo.com/cities#London'/>
            </rs:binding>
        </rs:ResultSolution>
        <rs:ResultSolution> ....
     </rs:solution>
<rs:resultSet>
```

49

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

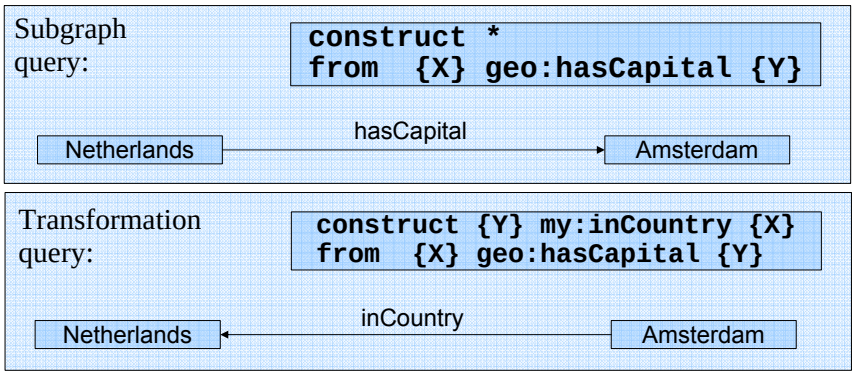
Query Result forms

- SeRQL select-queries return variable bindings
- Do we need something else?
  - Statements from RDF original graph
    - Data extraction
  - New statements derived from original data according to a specific need
    - Data conversion, *view*

50

SeRQL Construct-queries

- Construct-queries return RDF statements
  - The query result is either a ***subgraph*** of the original graph, or a ***transformed*** graph

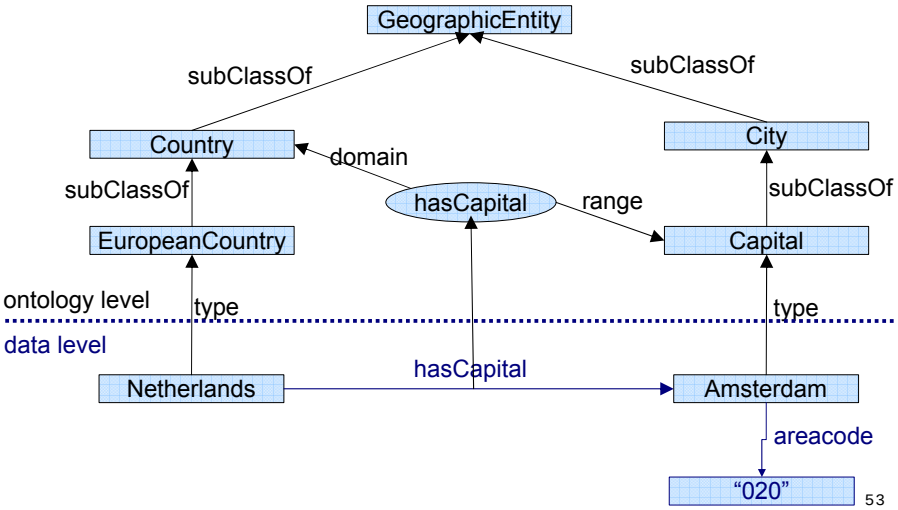


Schema Querying

- SeRQL has support for Schema querying
  - Class instances
  - Subclasses, Subproperties
  - etc.
- SeRQL "interprets" RDF(S) semantics
  - RDF and RDFS predicates explicitly mapped to their formal semantics
    - Transitivity of subClassOf property, inheritance of class instances, etc.
  - So it is not just querying the data graph

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

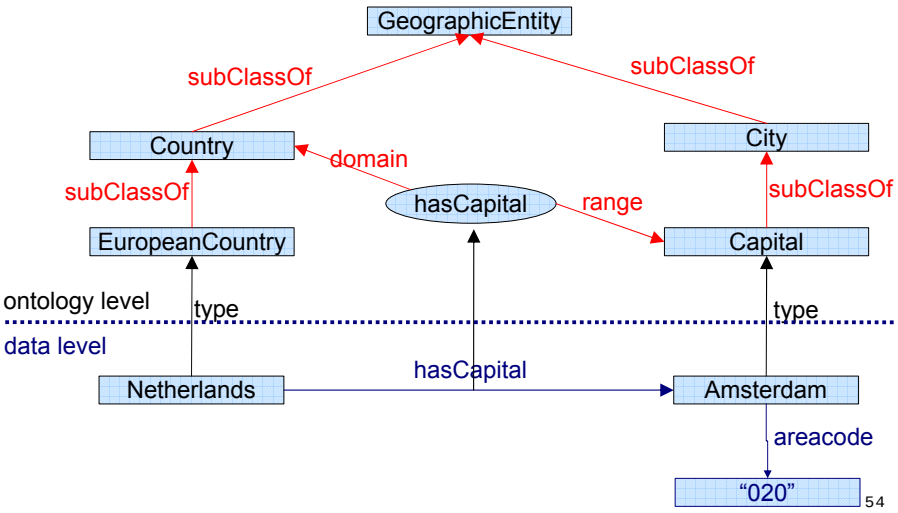
Schema Querying



53

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

Schema Querying



54

Schema querying example

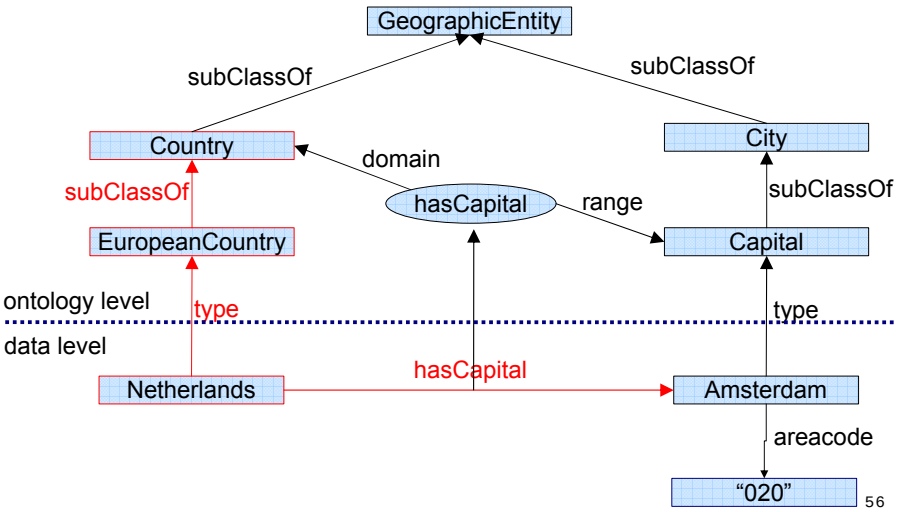
- Query: “return the range of the property hasCapital”

```
select X
from  {geo:hasCapital} rdfs:range {X}
```

- Query: “return all subclasses of GeographicEntity ”

```
select X
from  {X} rdfs:subClassOf
{geo:GeographicEntity}
```


Ontology/Data Querying



Ontology/Data Querying Example

- Query: “return all instances of the class Country”

```
select X
from  {X} rdf:type {geo:Country}
```

- Query: “return all countries, and the assertions (properties and values) for each”

```
select X, P, Y
from  {X} rdf:type {geo:Country};
          P {Y}
where  P != rdf:type
```


Ontology/Data Example (2)

- Query: “return all things that have a capital, and their type”

```
select X, Z
from  {X} geo:hasCapital {Y};
          rdf:type {Z}
```

Problem?

Z=GeographicEntity

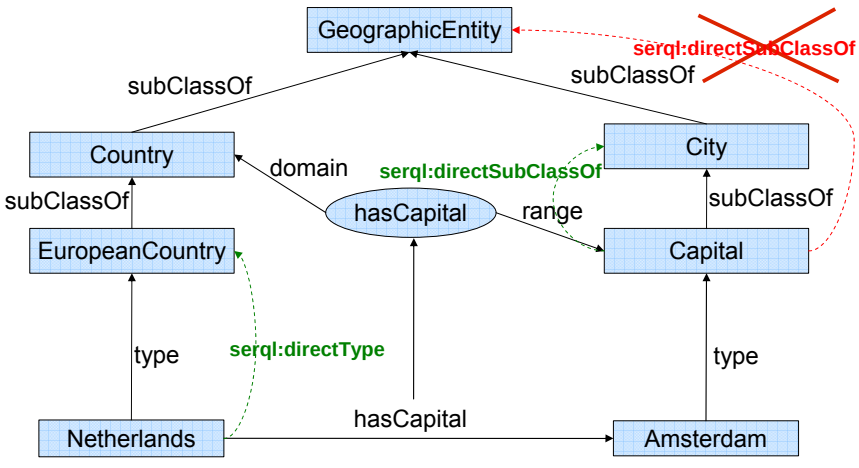
Z=Country

Z=EuropeanCountry

SeRQL Schema Built-ins

- SeRQL offers three built-in predicates that act as 'virtual' properties:
  - **`{X} serql:directSubClassOf {Y}`**
    true if and only if:
    - X subclassOf Y
    - X not equal to Y
    - There is no other class Z such that
      X subClass Z and Z subClass Y
  - **`{X} serql:directSubPropertyOf {Y}`**
  - **`{X} serql:directType {Y}`**

Built-ins (2)



Ontology/Data Example (2)

- Query: “return all things that have a capital, and their type”

```
select X, Z
from  {X} geo:hasCapital {Y};
          serql:directType {Z}
```


SeRQL: Other Features

- Composition of query results
  - **UNION**, **INTERSECT**, **MINUS**
- “Return all countries that are not European countries”

```
select X
from  {X} rdf:type {geo:Country}
MINUS
Select Y
from  {Y} rdf:type {geo:EuropeanCountry}
```

SeRQL: Other Features (2)

- Nested queries

```
construct {Y} my:newProperty {X}
from  {X} rdfs:subClassOf {Y}
Where X!=Y and NOT EXISTS
    (Select Z
    from  {X} rdfs:subClassOf {Z},
          {Z} rdfs:subClassOf {Y}
    where Z!=X AND Z!=Y
    )
```

Already seen newProperty?

`serql:directSubclassOf`

Summary

- We need a specific query language for RDF and RDF Schema
  - XQuery won't do the job
- SeRQL is a proposal language
  - Very expressive
    - Path expressions, schema/data querying, etc.
  - Formal semantics
  - Ease of use
  - Implemented

Complement: SPARQL

- W3C Candidate recommendation
- Already implemented
  - Jena RDF framework, etc…
- Similar features
- (Slightly) different syntax

```
PREFIX geo: <http://www.geo.org/schema.rdfs#>
SELECT ?X
WHERE
  { ?X geo:areacode "020" }
```

65