Stream and Complex Event Processing
A (brief) introduction to the semantic Web technologies

G. Cugola  E. Della Valle  A. Margara

Politecnico di Milano  cugola@elet.polimi.it  dellavalle@elet.polimi.it
Vrije Universiteit Amsterdam  a.margara@vu.nl
Course outline

- History and principles of stream computing and complex event processing
  - Description of the area
  - Typical applications
  - Challenges
- A modeling framework for IFP systems
  - Functional model
  - Processing model
  - Deployment model
  - Interaction model
  - Data model
  - Time model
  - Rule model
  - Language model
- The realm of stream reasoning
  - A brief introduction to the semantic Web technologies
  - From stream processing to stream reasoning
- Distributing to survive: The "operator placement" problem
  - Theory
  - Algorithms
- On managing uncertainty in data and rules
  - A model of uncertainty for information flow processing systems
- Discovering existing systems
  - Complex event processing systems in practice
  - Data streaming systems in practice
  - Stream reasoning systems in practice
- On benchmarking Information Flow Processing Systems
  - The problem
  - Possible solutions
- Putting it all together
  - A practical scenario to test IFP systems
- Experience report
Introduction

Computer should understand more

- Large number of integrations
  - ad hoc
  - pair-wise

- Millions of Applications

- Too much information to browse, need for searching and mashing up automatically

Each site is “understandable” for us

Computers don’t “understand” much
Introduction

Do We Really Know What “Understanding” Means?

[ source http://www.thefarside.com/ ]
The Problem: “Semantic Gap”

Symbolic Description

Sensor Data

Semantic Gap
“Understanding” Means Bridging the Gap

Symbolic Description

Sensor Data

Stream & Complex Event Processing - Semantic Web Intro
Introduction

Two ways for computer to “understand”

• Smart Machine
• Smart Data
Introduction

Smart Machines can bridge the gap

- Working examples found on the Web
  - Image Processing
    - retrievr: find by sketching
      - http://labs.systemone.at/retrievr/
  - Audio Processing
    - midomi: find by singing
      - http://www.midomi.com/
  - Natural Language Processing
    - semantic proxy:
      - http://semanticproxy.opencalais.com/about.html

- [...]
... but do they talk each other?

Natural Language Processing (NLP) meets Image Processing (IP)

NLP: What does your eye see?
IP: I see a sea
NLP: You see a “c”?
IP: Yes, what else could it be?

[Source NLP Related Entertainment
http://www.cl.cam.ac.uk/Research/NL/amusement.html]
Introduction

... smart data are need

Natural Language Processing (NLP) meets Image Processing (IP)

NLP: What does your eye see?
IP : I see a wordnet:word-sea
NLP: mmm, I see a wordnet:word-c
IP : I believe we have different understanding of the world ... 
NLP: So do I

The Semantic Web offers a set of standards that lowers the barriers to employ smart data at large scale
The Semantic Web is not a separate Web, but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation.

http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9B09EC588EF21

Key concepts
- an extension of the current Web
- in which information is given well-defined meaning
- better enabling computers and people to work in cooperation.
  - Both for computers and people
Introduction

The Semantic Web

• “The Semantic Web is not a separate Web, but an extension of the current one [...]”
• “The Semantic Web [...] in which information is given well-defined meaning [...]”

**Human understandable but “only” machine-readable**

**Human and machine “understandable”**
Introduction

The Semantic Web

Fewer Integration
- standard
- multi-lateral

Even More Applications

Even More Applications

Easier to understand for people

More “understandable” for computers

[...] better enabling computers and people to work in cooperation.
The Semantic Web “in the wild”
The Semantic Web “in the wild”

Google Knowledge Graph

Google Rich Snippet
Introduction

The Semantic Web “in the wild” 3/5
The Semantic Web “in the wild”

• **Schema.org**
  • an initiative launched on 2 June 2011 by Bing, Google and Yahoo!
  • to “create and support a common set of schemas for structured data markup on web pages.”

• A **collection of vocabularies** organized in a broad type hierarchy
  • See [http://schema.org/docs/full.html](http://schema.org/docs/full.html)
The Semantic Web “in the wild”

318,393 sites!
Introduction

Semantic Web “layer cake”

Already Possible

Under Investigation

Standardized

User Interface & Applications

Trust

Proof

Unifying Logic

Query: SPARQL

Ontology: OWL

Rule: RIF

RDFS

Data interchange:

RDF

XML

URI/IRI

[source http://www.w3.org/2007/03/layerCake.png]
Architectural view of the Semantic Web

[source http://www.w3.org/DesignIssues/diagrams/sw-double-bus.png]
Data Interchange: RDF
Looking for a flexible data model

- Why
  - Application are always changing (competitive environment)
  - People are always adding more features
  - Graceful evolution is important

- Optimal: relational model
  - Relational model is remarkably flexible
  - Supports graceful evolution
    - Change => Add another table
    - Existing queries are unaffected
  - Easily accommodates new data
    - Without affecting existing queries
  - Allows data to be easily combined ("joined") in new ways
  - 25+ years of relational database experience
RDF in a nutshell

Resource Description Framework

- The adaptation of the relational model to the Web give rise to RDF
- From *tuples* to *triples*

- Any relational data can be represented as triples
  - Row Key --> Subject
  - Column --> Property
  - Value --> Value
RDF in a nutshell

Other data structure in RDF

• Trees can be represented in RDF

• Anything can be represented in RDF
RDF in a nutshell

Representing relational data in RDF (almost)

• E.g., city-related data

<table>
<thead>
<tr>
<th>City</th>
<th>Country</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT.2</td>
<td>Italy</td>
<td>1,298,972</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT.2</td>
<td>Milano</td>
</tr>
<tr>
<td>IT.2</td>
<td>Milan</td>
</tr>
<tr>
<td>IT.2</td>
<td>Mailand</td>
</tr>
</tbody>
</table>

• Represented in RDF (almost)
RDF in a nutshell

Representing relational data in RDF (almost)

- Two important problems
  - Once out of the database internal ID (e.g., BB.2) becomes useless
  - Once out of the database internal names of schema element (e.g., Category) becomes useless as well

- **RDF solves it by using URI**
  - Internal ID should be replaced by URI
  - Internal schema names should be replaced by URI
  - Values do (always) not need to be URI-fied

```
http://www.geonames.org/ontology#P
http://www.w3.org/1999/02/22-rdf-syntax-ns#type

http://sws.geonames.org/3173435/
http://www.geonames.org/ontology#inCountry

http://www.geonames.org/countries/#IT

http://www.geonames.org/ontology#population

http://www.geonames.org/countries/#IT

http://www.geonames.org/ontology#Population

http://www.w3.org/2000/01/rdf-schema#label

Legend
- resource
- literal

Milano
Milan
Mailand

1.298.972
```
RDF in a nutshell

Why should I use URI?  1/2

• Data will merge automatically!

http://sws.geonames.org/3173435/
http://www.geonames.org/ontology#inCountry
http://www.geonames.org/countries/#IT

http://sws.geonames.org/3173435/
http://dbpedia.org/resource/Postalcode
20100

http://sws.geonames.org/3173435/
http://www.geonames.org/ontology#inCountry
http://www.geonames.org/countries/#IT

20100
Why should I use URI?

- What is a value? When shall we URI-fy a value?
  - Literals cannot be used to merge different data set
  - E.g., having chosen to represent postal codes as a string, merging different data sets using postal codes is impossible

- 20100 may refer to lots of different thing on the Web e.g., try [http://images.google.com/images?q=20100](http://images.google.com/images?q=20100)

- URI-fy any value that can be eventually used to merge different dataset and leave the other values as literals
Serializing RDF

• Three alternatives to write triples, in
  1. RDF/XML: Standard serialization in XML
     <Description about="subject">
     <property>value</property>
     </Description>
     • e.g., http://www.geonames.org/3173435/about.rdf
     • Check-out the triples using http://www.w3.org/RDF/Validator/
  2. NTriples: Simple (verbose) reference serialization (for specifications only)
     <http://...subject> <http://...predicate> “value”.
  3. N3 and Turtle: Developer-friendly serializations
     :subject :property “value”.

Stream & Complex Event Processing - Semantic Web Intro
Serializing RDF in Turtle - namespaces

• URI terms can be abbreviated using namespaces

```turtle
@prefix geo: <http://www.geonames.org/ontology#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .


• <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> = 'a'

```
RDF in a nutshell

Serializing RDF in Turtle - Literals

- Literals: "Milano"
  - Typed literals: "3.14"^^xsd:float
  - Literals with language tags: "日本語"@ja

@prefix geo: <http://www.geonames.org/ontology#> .

<http://www.geonames.org/3173435/> geo:alternateName "Milano"@IT .
Ontology: RDF-S and OWL
What is an Ontology?

- A model of (some aspect of) the world
  - Introduces **vocabulary** relevant to domain
    - e.g., anatomy
  - Specifies **meaning** (semantics) of terms
    - **Heart** is a **muscular organ** that is **part of** the **circulatory system**
  - **Formalised** using suitable logic
    - $\forall x. \left( \text{Heart}(x) \rightarrow \text{MuscularOrgan}(x) \land \exists y. \left( \text{isPartOf}(x, y) \land \text{CirculatorySystem}(y) \right) \right)$

- **Shared** among multiple people organizations
RDF-S/OWL in a nutshell

That should clear up a few things...
A simple ontology

Artist

Painter

creates

paints

Piece

Paint

Sculptor

Sculptor

Sculpt

creates

sculpts
RDF-S/OWL in a nutshell

Specifying classes, sub-classes and instances

• Creating a class
  • RDFS: Artist rdf:type rdfs:Class .
  • FOL: ∃x Artist(x)

• Creating a subclass
  • RDFS: Painter rdfs:subClassOf Artist .
  • RDFS: Sculptor rdfs:subClassOf Artist .
  • FOL: ∀x [Painter(x) ∨ Sculptor(x) → Artist(x)]

• Creating an instance
  • RDFS: Rodin rdf:type Sculptor .
  • FOL: Sculptor(Rodin)
Specifying properties and sub-properties

- **Creating a property**
  - RDFS: creates rdf:type rdf:Property .
  - FOL: \( \exists x \ \exists y \ \text{Creates}(x, y) \)

- **Using a property**
  - RDFS: Rodin creates TheKiss .
  - FOL: Creates(Rodin, TheKiss)

- **Creating subproperties**
  - RDFS: paints rdfs:subPropertyOf creates .
  - FOL: \( \forall x \ \forall y \ [\text{Paints}(x, y) \rightarrow \text{Creates}(x, y)] \)
  - RDFS: sculpts rdfs:subPropertyOf creates .
  - FOL: \( \forall x \ \forall y \ [\text{Sculpts}(x, y) \rightarrow \text{Creates}(x, y)] \)
RDF-S/OWL in a nutshell

Specifying domain/range constrains

- Checking which classes and properties can be used together
- RDFS:
  
  creates rdfs:domain Artist.
  creates rdfs:range Piece.
  paints rdfs:domain Painter.
  paints rdfs:range Paint.
  sculpts rdfs:domain Sculptor.
  sculpts rdfs:range Sculpt.

- FOL:
  
  $\forall x \forall y \ [\text{Creates}(x,y) \rightarrow \text{Artist}(x) \land \text{Piece}(y)]$
  $\forall x \forall y \ [\text{Paints}(x,y) \rightarrow \text{Painter}(x) \land \text{Paint}(y)]$
  $\forall x \forall y \ [\text{Sculpts}(x,y) \rightarrow \text{Sculptor}(x) \land \text{Sculpt}(y)]$
RDF-S/OWL in a nutshell

The ontology we specified

Artist
Painter

creates

paints

Sculptor

Piece
Paint

sculpts

Sculpt

Stream & Complex Event Processing - Semantic Web Intro
**RDF semantics (a part of it)**

<table>
<thead>
<tr>
<th>if</th>
<th>then</th>
</tr>
</thead>
<tbody>
<tr>
<td>x rdfs:subClassOf y .</td>
<td>a rdf:type y .</td>
</tr>
<tr>
<td>a rdf:type x .</td>
<td>x rdfs:subClassOf y .</td>
</tr>
<tr>
<td></td>
<td>x rdfs:subClassOf z .</td>
</tr>
<tr>
<td></td>
<td>y rdfs:subClassOf z .</td>
</tr>
<tr>
<td></td>
<td>x a y .</td>
</tr>
<tr>
<td></td>
<td>x b y .</td>
</tr>
<tr>
<td></td>
<td>a rdf:type y .</td>
</tr>
<tr>
<td></td>
<td>a rdfs:subPropertyOf b .</td>
</tr>
<tr>
<td></td>
<td>a rdfs:subPropertyOf b .</td>
</tr>
<tr>
<td></td>
<td>a rdfs:subPropertyOf c .</td>
</tr>
<tr>
<td></td>
<td>b rdfs:subPropertyOf c .</td>
</tr>
<tr>
<td></td>
<td>a rdfs:subPropertyOf c .</td>
</tr>
<tr>
<td></td>
<td>a rdfs:domain z .</td>
</tr>
<tr>
<td></td>
<td>x rdf:type z .</td>
</tr>
<tr>
<td></td>
<td>x rdf:type z .</td>
</tr>
<tr>
<td></td>
<td>u rdf:type z .</td>
</tr>
</tbody>
</table>

Read out more in RDF Semantics [http://www.w3.org/TR/rdf-mt/](http://www.w3.org/TR/rdf-mt/)
RDF-S/OWL in a nutshell

RDF semantics at work

• Shared the ontology ...

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://www.ex.org/schema#> .

ex:Sculptor rdfs:subClassOf ex:Artist .
ex:Painter rdfs:subClassOf ex:Artist .
ex:Sculpt rdfs:subClassOf ex:Piece.
ex:Painting rdfs:subClassOf ex:Piece .
ex:creates rdfs:domain ex:Artist .
ex:creates rdfs:range ex:Piece.
ex:sculpts rdfs:subPropertyOf ex:creates .
ex:sculpts rdfs:domain ex:Sculptor .
ex:sculpts rdfs:range ex:Sculpt .
```

• ... when transmitting the following triple ...

```
ex:Rodin ex:sculpts ex:TheKiss .
```
RDF-S/OWL in a nutshell

Without Inference

- A recipient, that only understands XML syntax, receiving
  
  `<RDF>
    <Description about="Rodin">
      <sculpts resource="TheKiss"/>
    </Description>
  </RDF>`

- can answer the following queries
  - What does Rodin sculpt?
    ```
    RDF/Description[@about='Rodin']/sculpts/@resource
    ```
  - Who does sculpt TheKiss?
    ```
    RDF/Description[sculpts/@resource='TheKiss']/@about
    ```
  - Try out your self at `http://www.mizar.dk/XPath/`

- but it cannot answer
  - Who is Rodin?
  - What is TheKiss?
  - Is there any Sculptor/Scupts?
  - Is there any Artist/Piece?
Knowing the ontology and RDF semantics ...

• A recipient, that knows the ontology and “understands” RDF semantics,

- Receiving Rodin sculpts TheKiss.
RDF-S/OWL in a nutshell

... a reasoner can answer

- the previous queries
  - What does Rodin sculpt?
    ex: TheKiss
  - Who does sculpt TheKiss?
    ex: Rodin

- and it can also answer
  - Who is Rodin?
    ex: Artist, ex: Sculptor, rdfs:Resource
  - What is TheKiss?
    ex: Sculpt, ex: Piece, rdfs:Resource
  - Is there any Sculptor?
    ex: Rodin
  - Is the any Artist?
    ex: Rodin
  - Is there any Sculpt?
    ex: TheKiss
  - Is there any Piece?
    ex: TheKiss
Query: SPARQL
SPARQL in a nutshell

What is SPARQL?

• SPARQL
  • is the query language of the Semantic Web
  • stays for **SPARQL Protocol and RDF Query Language**

• A Query Language ...:
  **Find named places:**
  • PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
    SELECT ?poi
    WHERE { ?poi a sr:NamedPlace . }

• ... and a Protocol.
  http://lod.openlinksw.com/sparql?&query=PREFIX+sr%3A+%3Chttp%3A%2F%2Fwww.streamreasoning.org%2Fsr4ld2011%2Fonto%2F%3E%0D%0ASELECT+%3Fpoi +WHERE+{+%3Fpoi+a+sr%3ANamedPlace+.+}
SPARQL in a nutshell

Anatomy of a SPARQL SELECT query

Declare prefix shortcuts (optional)

```
PREFIX foo: <...>
PREFIX bar: <...>
...
```

Define the dataset (optional)

```
SELECT ...
FROM ...
FROM NAMED <...>
WHERE {
...
}
```

Query modifiers (optional)

```
ORDER BY ...
LIMIT ...
OFFSET ...
```

Query result clause

```
```
SPARQL in a nutshell

Triple Pattern Syntax

- Turtle-like: URIs, QNames, literals, convenience syntax.
- Adds variables to get basic patterns
  - ?var
  - Variable names are a subset of NCNames (no "-" or ".")
- E.g.,
  - simple
    - ?poi a sr:NamedPlace .
  - complex
    - ?poi a geo:NamedPlace .
- Adds
  - OPTIONAL to cope with semi-structured nature of RDF
  - FILTER to select solution according to some criteria
  - UNION operator to get complex patterns
Test data: Data Model

- **sioc:UserAccount**
  - `sioc:id(xsd:string)`
  - `sioc:creator_of` to `sioc:Post`
- **sioc:Post**
  - `sioc:content(xsd:string)`
- **sr:TwitterUser**
  - `sr:screenName(xsd:string)`
  - `sr:following` to `sr:Tweet`
- **sr:Tweet**
  - `sr:messageID(xsd:string)`
  - `sr:messageTimeStamp(xsd:string)`
  - `sr:talksAbout`
- **geo:SpatialThing**
  - `geo:lat(xsd:float)`
  - `geo:long(xsd:float)`
- **sr:NamedPlace**
  - `sr:NamedPlace` to `sr:Tweet`
  - `sr:NamedPlace` to `sioc:UserAccount`

*Stream & Complex Event Processing - Semantic Web Intro*
Writing a Simple Query

Data

@prefix sr: <http://www.streamreasoning.org/sr4ld2011/onto#>.

sr:LaScala a sr:NamedPlace .
sr:GalleriaVittorioEmanueleII a sr:NamedPlace .
sr:Duomo a sr:NamedPlace .

Query

PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>

SELECT ?poi
WHERE { ?poi a sr:NamedPlace . }

Results

<table>
<thead>
<tr>
<th>?poi</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII">http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII</a></td>
</tr>
<tr>
<td><a href="http://www.streamreasoning.org/sr4ld2011/data#LaScala">http://www.streamreasoning.org/sr4ld2011/data#LaScala</a></td>
</tr>
<tr>
<td><a href="http://www.streamreasoning.org/sr4ld2011/data#Duomo">http://www.streamreasoning.org/sr4ld2011/data#Duomo</a></td>
</tr>
</tbody>
</table>
SPARQL in a nutshell

Writing a bit more complex query

Query

```sparql
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>

SELECT ?poi ?category
WHERE { ?poi a geo:NamedPlace .
  ?poi skos:subject ?category . }
```

Results

<table>
<thead>
<tr>
<th>?drug</th>
<th>?category</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
SPARQL in a nutshell

RDF Term Constraints

- SPARQL allows restricting solutions by applying the FILTER clause.
- An RDF term bound to a variable appears in the results if the FILTER expression, applied to the term, evaluates to TRUE.

Query

```sparql
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?poi ?lat ?log
WHERE {
}
```

- Results

<table>
<thead>
<tr>
<th>?poi</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleIi">http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleIi</a></td>
</tr>
<tr>
<td><a href="http://www.streamreasoning.org/sr4ld2011/data#LaScala">http://www.streamreasoning.org/sr4ld2011/data#LaScala</a></td>
</tr>
<tr>
<td><a href="http://www.streamreasoning.org/sr4ld2011/data#Duomo">http://www.streamreasoning.org/sr4ld2011/data#Duomo</a></td>
</tr>
</tbody>
</table>
Value Tests

- Notation for value comparison: <, >, =, <=, >= and !=
- Test functions
  - Check if a variable is bound: BOUND
  - Check the type of resource bound: isIRI, isBLANK, isLITERAL
- Accessing accessories: LANG, DATATYPE
- Logic operators: || and &&
- Comparing strings: REGEX, langMatches
- Constructor functions: bool, dbl, flt, dec, int, dT, str, IRI
- Extensible Value Testing
  - (see http://www.w3.org/TR/rdf-sparql-query/#extensionFunctions )
SPARQL in a nutshell

Value Tests - Extensible Value Testing

• Find all schools within a 5km radius around a specific location, and for each school find coffeeshops that are closer than 1km.

• PREFIX lgdo: <http://linkedgeodata.org/ontology/>
  WHERE {
    ?school a lgdo:School .
    ?coffeeshop a lgdo:CoffeeShop .
    FILTER ( bif:st_intersects(?schoolgeo, bif:st_point(4.892222, 52.373056), 5) &&
              bif:st_intersects(?coffeeshopgeo, ?schoolgeo, 1) ) .
  }

• Click here for query results on a Virtuoso endpoint used by LinkedGeoData project.
Optional Graph Patterns

- **OPTIONAL graph patterns** accommodate the need to add information to a result but without the query failing just because some information is missing.

**Query**

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX srd: <http://www.streamreasoning.org/sr4ld2011/data#>
SELECT ?t ?p ?poi
WHERE {
  ?t a sr:Tweet .
  ?poi a sr:NamedPlace .
  OPTIONAL { ?t ?p ?poi }
}
```

**Results**

<table>
<thead>
<tr>
<th>?t</th>
<th>?p</th>
<th>?poi</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:post1</td>
<td>sr:talksAbout</td>
<td>srd:GalleriaVittorioEmanueleII</td>
</tr>
<tr>
<td>_:post1</td>
<td>sr:talksAboutPositively</td>
<td>srd:LaScala</td>
</tr>
<tr>
<td>_:post1</td>
<td>null</td>
<td>srd:Duomo</td>
</tr>
</tbody>
</table>
SPARQL in a nutshell

Matching alternatives with UNION

- **UNION graph patterns** allows to match alternatives

  - **Query**

    ```sparql
    PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
    PREFIX srd: <http://www.streamreasoning.org/sr4ld2011/data#>
    SELECT ?t ?poi
    WHERE {
      ?t a sr:Tweet .
      ?poi a sr:NamedPlace .
      { ?t sr:talksAbout ?poi }
      UNION
      { ?t sr:talksAboutPositively ?poi }
    }
    ```

  - **Results**

    | ?t  | ?poi                       |
    |-----|----------------------------|
    | _:post1 | srd:GalleriaVittorioEmanueleII |
    | _:post1 | srd:LaScala                |
Besides selecting tables of values, SPARQL allows three other types of queries:

- **ASK** - returns a boolean answering, does the query have any results?
- **CONSTRUCT** - uses variable bindings to return new RDF triples
- **DESCRIBE** - returns server-determined RDF about the queried resources

- **SELECT** and **ASK** results can be returned as XML or JSON.
- **CONSTRUCT** and **DESCRIBE** results can be returned via any RDF serialization (e.g. RDF/XML or Turtle).
SPARQL in a nutshell

CONSTRUCT Form

- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>

CONSTRUCT { ?u sr:talksAboutPositively ?poi }
WHERE {
  ?u a sr:TwitterUser .
  ?poi a sr:NamedPlace .
}
```

- Meaning
  - it requires to compute a property chain

- Results

```
@prefix sr:<http://www.streamreasoning.org/sr4ld2011/onto#> .

sr:Alice sr:talksAboutPositively sr:LaScala .
```
SPARQL in a nutshell

Wrap up of SPARQL

Source: Pérez, Arenas and Gutierrez, Chapter 1: On the Semantics of SPARQL, Semantic Web Information Management: A Model Based Perspective, Springer 2010
How many points of interest are in the dataset?

Query

PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT (count(?poi) AS ?numberOfPOI)
WHERE {
  ?poi a sr:NamedPlace .}
• How many points of interest are in each category?

Query

PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category (count(?poi) AS ?numberOfPOI)
WHERE {
  ?poi a sr:NamedPlace .
}
GROUP BY ?category

• NOTES

• The available built-ins are SUM, AVG, COUNT, MIN, MAX
• A DISTINCT clause can be use to avoid processing duplicates
• Which are the categories of points of interest in which the points of interest are positively discussed in more than 100 tweets?

• Query

```sparql
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category
WHERE {
  ?poi a sr:NamedPlace .
}
GROUP BY ?category
HAVING (count(?tweet) > 100)
```
• Which are the categories of points of interest in which the points of interest are positively discussed in more than 100 tweets? How many points of interest does each category contain?

• Query

```sparql
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category (count(?poi) AS ?numberOfPOI) 
WHERE {
  ?poi a sr:NamedPlace .
} 
GROUP BY ?category 
HAVING (count(?tweet) > 100)
```
SPARQL in a nutshell

Sub-queries

• Any group pattern can be a sub-query
• E.g., which are the categories of points of interest in which each point of interest is positively discussed in more than 10 tweets? How many points of interest does each category contain?

• PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>  PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category (count(?poi) AS ?numberOfPOI) WHERE {
  ?poi a sr:NamedPlace .
  {
    SELECT ?poi
    WHERE {
      ?poi a sr:NamedPlace .
    }
    GROUP BY ?poi
    HAVING (count(?tweet) > 10)
  }
} GROUP BY ?category
CONSTRUCT, Aggregates and Sub-queries

- When you need to output an RDF graph that contains the results of a query that uses aggregates, you need to use sub-queries.
- Output the points of interest in each category as an RDF graph

```sparql
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
CONSTRUCT 
{ 
  ?category sr:countedPOIs ?numberOfPOIs
} 
WHERE 
{ 
  
  SELECT ?category (count(?poi) AS ?numberOfPOIs)
  WHERE 
  { 
    ?poi a sr:NamedPlace .
  }
  
  GROUP BY ?category
}
```
Reasoning and Query Answering

- SPARQL alone cannot answer queries that require reasoning.
- but a reasoner can be exposed as a SPARQL service.
- Or a query can be rewritten in order to incorporate the ontology.
Reasoning and Information Integration

- Given ontology O and query Q, use O to rewrite Q as Q’ so that, for any set of ground facts A contained in multiple databases:
  - \( \text{answer}(Q, O, A) = \text{answer}(Q', \emptyset, A) \)
  - The answer of the query Q using the ontology O for any set of ground facts A is equal to answer of a query Q’ without considering the ontology O

- Use (Global As View) mapping M to map Q’ to multiple SQL queries to the various databases
RDF-S/OWL in a nutshell

Query Rewriting Technique (basics)

• Example:
  • Ontology
    • Doctors treats patients
      \[ \forall x \forall y \ [\text{treats}(x,y) \land \text{Patient}(y) \rightarrow \text{Doctor}(x)] \]
    • Consultants are doctors
      \[ \forall x \ [\text{Consultant}(x) \rightarrow \text{Doctor}(x)] \]
  • Query
    • Give me those that treats some patient
      \[ Q(x) \leftarrow \text{treats}(x,y) \land \text{Patient}(y) \]
    • For a profile of OWL named QL, the rewriting results in a union of conjunctive queries
      \[ Q(x) \leftarrow (\text{treats}(x,y) \land \text{Patient}(y)) \]
      \[ \lor \text{Doctor}(x) \]
      \[ \lor \text{Consultant}(x) \]
Query Rewriting Technique (basics)

• Relationship between ontology and databases defined by **mappings**, e.g.:
  
  Doctor $\rightarrow$ SELECT Name FROM Doctor  
  Patient $\rightarrow$ SELECT Name FROM Patient  
  treats $\rightarrow$ SELECT Dname, Pname FROM Treats  

• Note: the mapping can be partial, i.e., Consultant is non mapped

• Using the mapping the query resulting from the mapping can be translated in SQL

  $Q(x) \leftarrow (\text{treats}(x,y) \land \text{Patient}(y)) \lor \text{Doctor}(x) \lor \text{Consultant}(x)$

  $\downarrow$

  SELECT Name FROM Doctor UNION  
  SELECT Dname FROM Treats, Patient WHERE Pname = Name
Introduction

Architectural view of the Semantic Web

Does it make more sense?

[source http://www.w3.org/DesignIssues/diagrams/sw-double-bus.png ]