Semantics, knowledge graphs and ontologies in practice

Jose Emilio Labra Gayo

WESO Research group

University of Oviedo, Spain

Schedule

Day	Title	Topics
Day 1.	Semantic Technologies and Knowledge graphs	Semantic Web Linked data Knowledge graphs RDF data model Property graphs Wikibase graphs Examples and applications
Day 2.	RDF data modelling and SPARQL	Data modelling exercises with RDF and turtle SPARQL
Day 3.	Validating RDF data	Shape Expressions (ShEx) SHACL Validating Knowledge Graphs
Day 4.	Advanced topics	ShEx and SHACL compared Reasoning RDFS OWL Nanopublications

Representing information in RDF

RDF = data model to exchange information in the Web

Some considerations & trade-offs

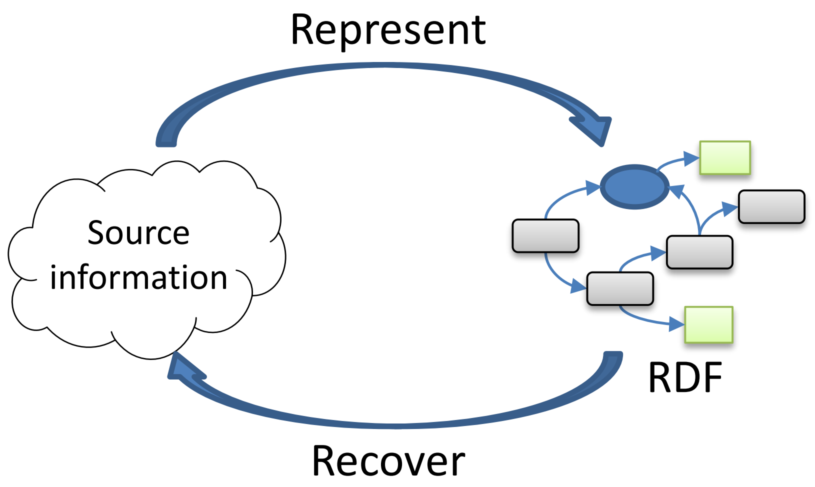
- Semantic accuracy
- Human readability
- Interoperability
- Performance

Semantic accuracy

Avoid semantic loss

Round-tripping

- From original representation to RDF
- From RDF recover original representation



Human readability

- RDF as a communication language
  - Turtle can be human readable
  - Useful for debugging

Interoperability

RDF data should be machine processable

- Adopt common vocabularies and URIs
- Don't reinvent the wheel
- Avoid ambiguity
- Provide context and provenance for assertions

Performance

- Find the right level of granularity
  - It may be difficult for some kind of content
    - Examples: audiovisual content

Some RDF data modeling patterns

N-ary relationships

Tabular data

Representing order

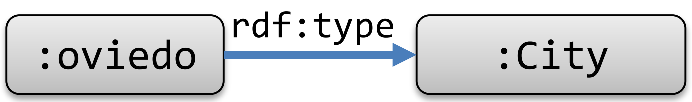
Reification and provenance

Grouping RDF triples and datasets

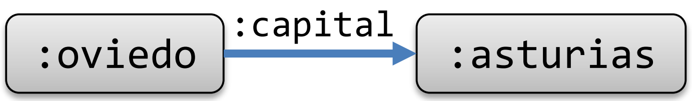
N-ary relationships

RDF can only express relationships between 1, 2 elements

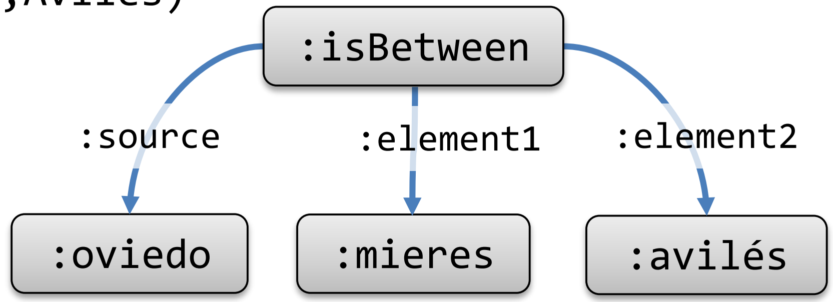
1-ary: *Oviedo is a city* `city(Oviedo)`



2-ary: *Oviedo is the capital of Asturias* `capital(Oviedo,Asturias)`



3-ary: *Oviedo is between Mieres and Avilés* `isBetween(Oviedo,Mieres,Aviles)`



Typical approach (*reify the relationship*)

Create an auxiliary node that represents the relationship

Add new relationships between nodes and the auxiliary node

Defining N-ary Relations on the Semantic Web:
https://www.w3.org/TR/swbp-n-aryRelations/

Tabular data

Example

Course

CID	Code	Title	Room	Teacher
23	CS101	Programming	A1	144
34	A102	Algebra	B2	144

Teacher

TeacherID	FirstName	LastName
144	Alice	Cooper

```
prefix : <http://example.org/>

:23 a :Course ;
    :code "cs101" ;
    :title "Programming"@en ;
    :room "A1" ;
    :teacher :144 .
:34 a :Course ;
    :code "A102" ;
    :title "Algebra"@en .
    :room "B2" ;
    :teacher :144 .
:144 a :Teacher ;
     :firstName "Alice" ;
     :lastName "Cooper" .
```

Each table can be seen as an n-ary relationship

RDB2RDF: A Direct Mapping of Relational Data to RDF.
https://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/

Representing order

RDF can easily represent sets but not lists

Several solutions

- Linked lists (RDF collections)
- Order-indicating properties (RDF containers)
- Add order annotations to values
- Give up order

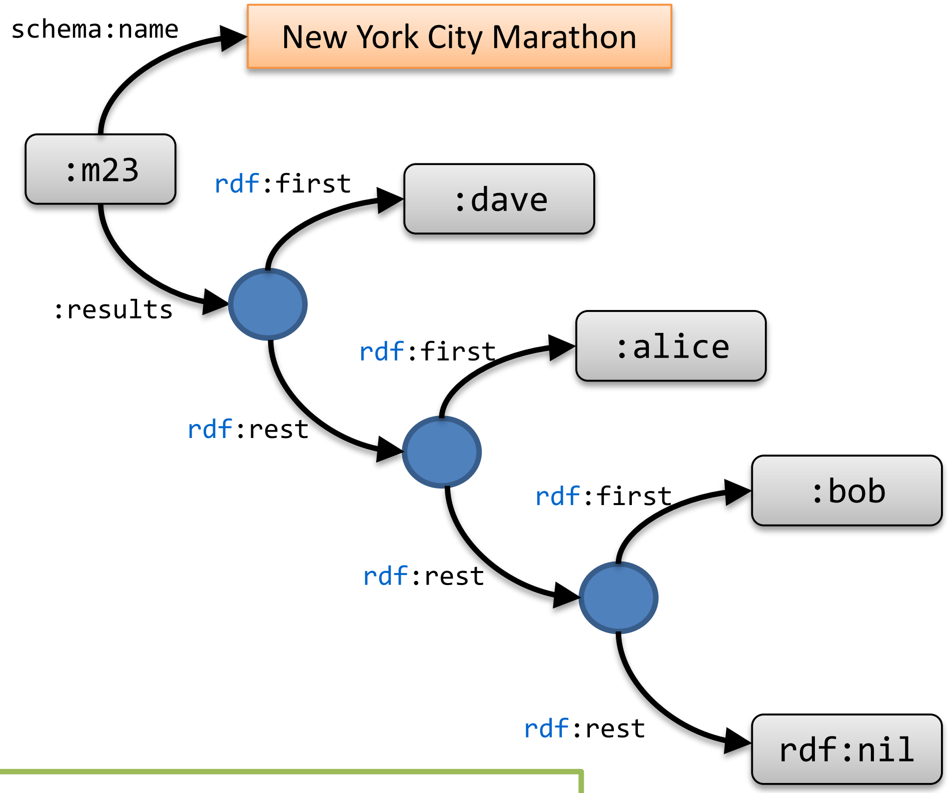
Solution 1
Representing order with linked lists

Ordered lists

```
:m23 schema:name "New York City Marathon ";
     :results    ( :dave :alice :bob ) .
```

Internally, represented as linked lists

```
:m23 schema:name "New York City Marathon ";
     :results _:1 .
_:1 rdf:first :dave ;
    rdf:rest _:2 .
_:2 rdf:first :alice ;
    rdf:rest _:3 .
_:3 rdf:first :bob ;
    rdf:rest rdf:nil .
```



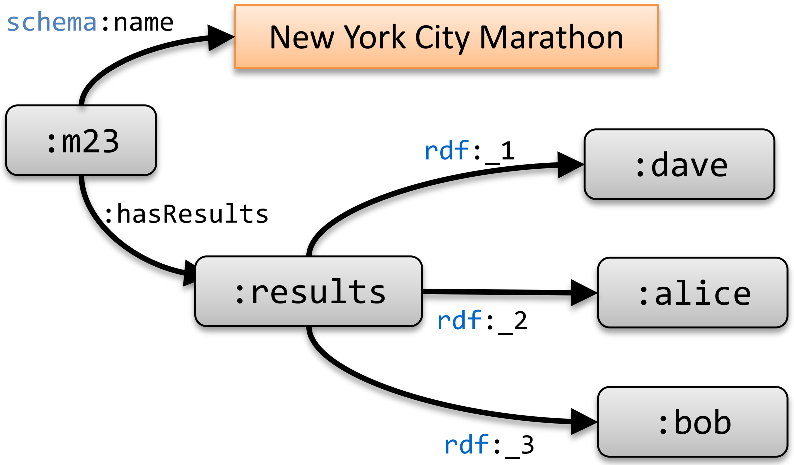
Pros: Elegant representation, easy insert/delete, mark end of list
Cons: Inefficient access to a given element

Solution 2
Representing order with properties

Use properties that indicate the order

RDF already has some specific properties: `rdf:_1, rdf:_2, ...`

```
:m23 schema:name "New York City Marathon ";
     :hasResults :results .
:results rdf:_1 :dave ;
         rdf:_2 :alice ;
         rdf:_3 :bob   .
```



Pros: Direct access to each element

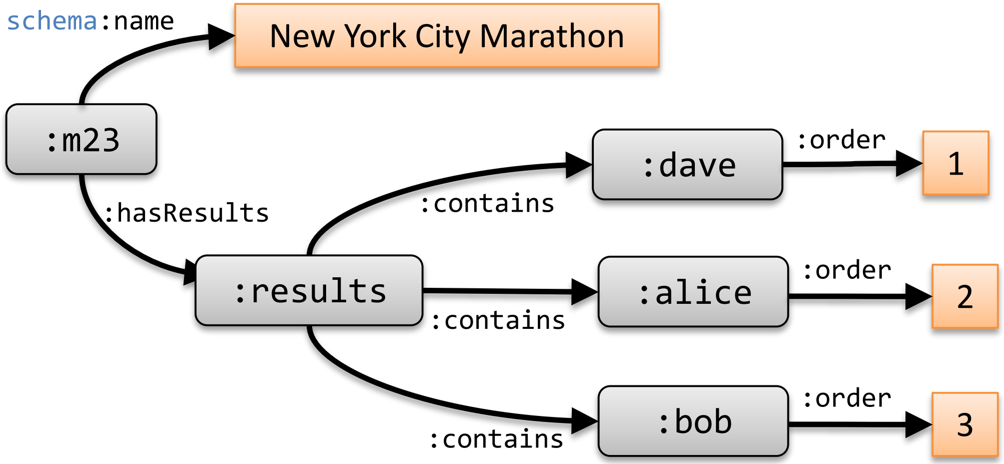
Cons: Not easy to detect the structure of list (length of the list, missing values, ...)

Harder to insert/delete elements

Solution 3
Representing order with annotated values

Annotate the elements with a value that indicates order

```
:m23 schema:name "New York City Marathon ";
     :hasResults :results .
:results :contains :dave, :alice, :bob.
:dave  :order 1 .
:alice :order 2 .
:bob   :order 3 .
```



Pros: Direct access to each element is possible, length of list available

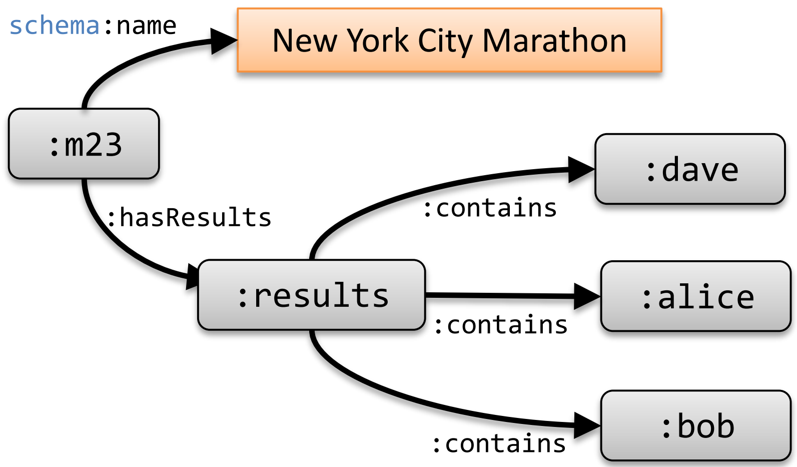
Cons: It is possible to create inconsistencies (elements with same order)
Harder to insert/delete elements

Solution 4
Ignore the order

Sometimes order is not really required

Give up the order and represent lists as sets

```
:m23 schema:name "New York City Marathon ";
     :hasResults :results .
:results :contains :dave, :alice, :bob.
```



Pros: Easy to do in RDF, may suffice for many use cases
Cons: No order

Solution 5
Combine several approaches

RDF is very versatile

- It is possible to combine several approaches

```
:m23 schema:name "New York City Marathon ";
     :hasResults :results .
:results :contains :dave, :alice, :bob.
         :order    ( :dave, :alice, :bob ) .
```

Pros: May offer the pros of the different approaches
Cons: Increased data volumen, redundancy and possible inconsistencies

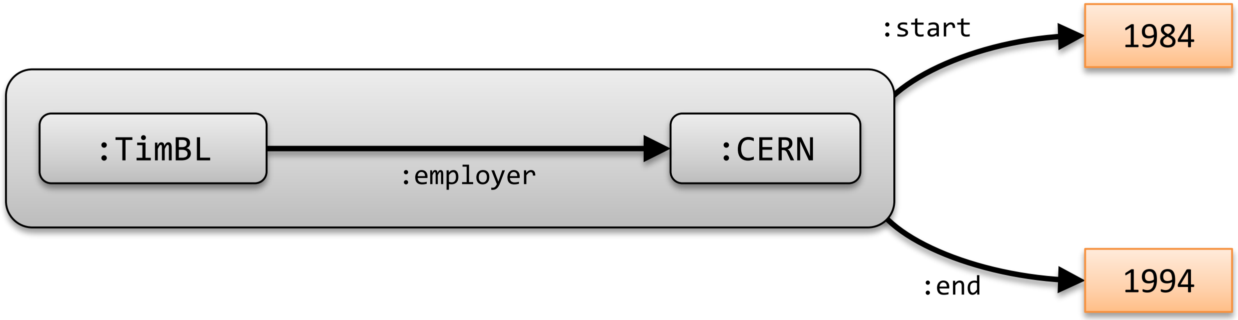
Reification

Reification: add statements about statements

Example: *Tim Berners-Lee is employed at CERN* *(between 1984 and 1994)*

Some approaches

- Standard RDF reification
- N-ary relations
- RDF-*
- Named graphs



Reification approach 1
Standard RDF reification

Introduced already in RDF 1.0

Predicates `rdf:subject, rdf:predicate, rdf:subject`

Class `rdf:Statement`

```
:s1 a rdf:Statement ;
    rdf:subject   :TimBl ;
    rdf:predicate :employer ;
    rdf:object    :CERN ;
    :start        "1984"^^xsd:gYear ;
    :end          "1994"^^xsd:gYear .
```

Pros: It is part of RDF, since RDF 1.0
Cons: Not easy to manage and not very flexible. Not compatible with OWL DL

Reification approach 2
Statements as n-ary relations

- Create an auxiliary node to represent the statement
  - Add properties to relate the nodes with that auxiliary node

```
:timBl :employer :e .
:e :organization :CERN ;
   :start        "1984"^^xsd:gYear ;
   :end          "1994"^^xsd:gYear .
```

Pros: It can be directly expressed in RDF

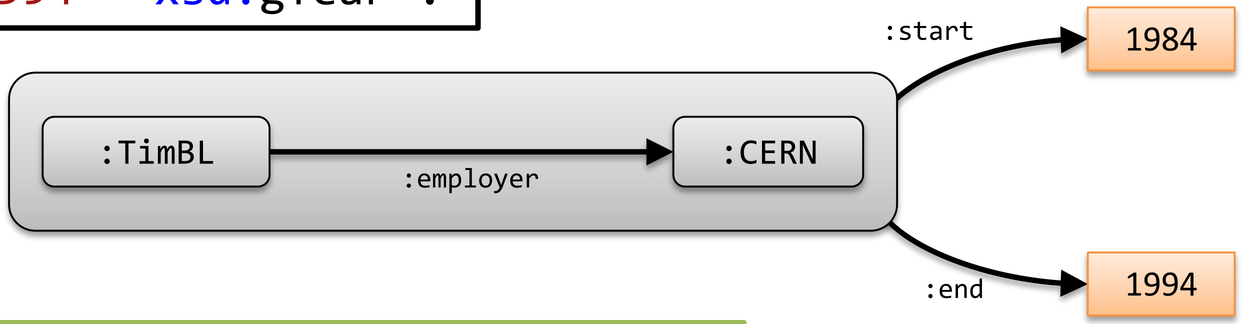
Cons: Requires the creation of auxiliary nodes and properties

Reification approach 3
RDF-*

RDF-* = RDF extension

where graphs can be either subjects or objects of a statement

```
<< :timBl :employer :CERN >> :start "1984"^^xsd:gYear ;
                             :end   "1994"^^xsd:gYear .
```



Pros: It expresses directly reification

Cons: Not yet widely adopted. It may require tools to convert to RDF

RDF-* Community group specification: https://w3c.github.io/rdf-star/cg-spec

Reification approach 4
Named graphs

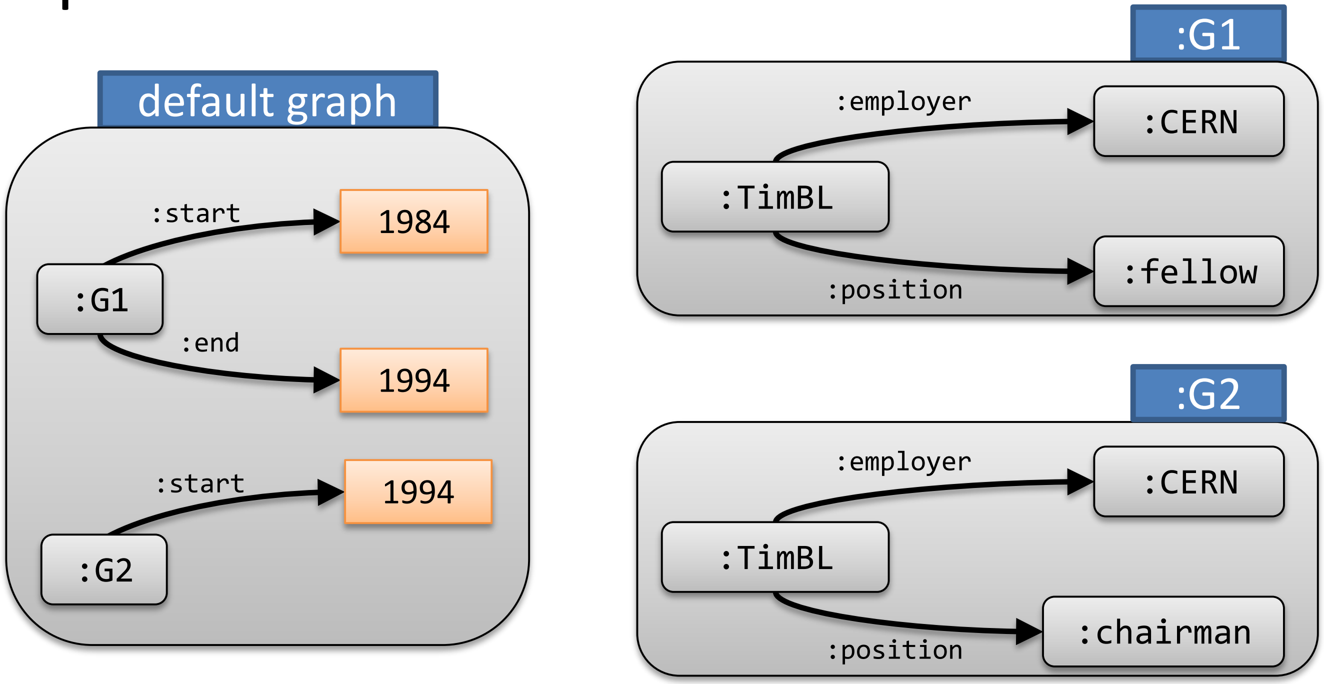
RDF datasets = collection of RDF graphs (supported also by SPARQL)

- A default graph
- Zero or more named graphs (name = IRI/Blank node)

TRIG = Turtle extension that can express RDF datasets

```
:G1 :start "1984"^^xsd:gYear .
:G1 :end   "1994"^^xsd:gYear .
:G2 :start "1994"^^xsd:gYear .

:G1 {
  :timBl :employer :CERN ;
         :position :fellow
}
:G2 {
  :timBl :employer :W3C .
         :position :chairman
}
```


default graph
:start
1984
:G1
:end
1994
:start
1994
:G2
:G1
:employer
:CERN
:TimBL
:fellow
:position
:G2
:employer
:CERN
:TimBL
:chairman
:position