Foundation of triple terms and reification

An Introduction to RDF and SPARQL 1.2
ISWC 2025
2 November 2025

Pierre-Antoine Champin
Enrico Franconi
Ora Lassila
Ruben Taelman

https://www.w3.org/Talks/2025/iswc-tutorial-rdfsparql-12/

Outline

- Triples, propositions, and facts
- Triple Terms
- Reification
- Examples
- Reification in formal ontologies and natural language
- Relation with UML, ER, ORM, Databases, Description Logics
- Relation with Property Graphs
- Difference with RDF 1.1 and RDF-star reification

In the beginning, W3C created triples...

RDF Triples and Propositions

The denotation of an **RDF triple** is a **proposition** --
namely a **relation**, denoted by the predicate,
relating the resources denoted by the subject and object of the triple.

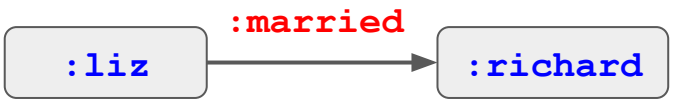
An RDF triple **may or may not** be **asserted**,
namely it may be unknown whether the proposition it denotes
holds true in the context of an RDF graph.

Facts

A proposition that holds in an interpretation is called a **fact**,
corresponding to an **RDF triple** which is **true** (via `IEXT`) in that interpretation.

Interpretation $\mathcal{I}$:

IR = {richard, liz}
IP = {married}
IEXT = {married ↦ <liz, richard>}
IS = {`:liz` ↦ liz, `:richard` ↦ richard, `:married` ↦ married}



this RDF triple denotes a **fact** in $\mathcal{I}$,
it denotes a **proposition true** in $\mathcal{I}$

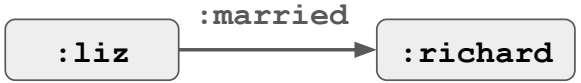
Asserted Facts

An **asserted fact** in an RDF graph is the proposition denoting
an **asserted RDF triple** in the graph – namely an RDF triple in the graph –
so that the proposition it denotes holds true (via IEXT)
in *all* the interpretations satisfying the RDF graph.

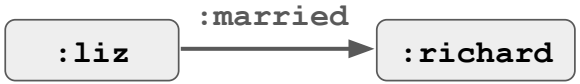
Interpretation $\mathcal{I}$:

IR = {richard, liz}
IP = {married}
IEXT = {married ↦ <liz, richard>}
IS = {`:liz` ↦ liz, `:richard` ↦ richard,
`:married` ↦ married}

Graph $\mathcal{G}$ = {



$\mathcal{I} \models \mathcal{G}$



denotes an asserted fact in $\mathcal{G}$

Propositions in RDF 1.1?

RDF 1.1 can only deal with asserted triples, namely it is possible to describe a context only by listing all the true propositions in it, and it is impossible to refer to propositions independently of their truth value.

...and then W3C said,
“Let there be triple terms”...

Triple terms

RDF 1.2 introduces the ability to use an **RDF triple**
as an **RDF term** within another RDF triple, without being an asserted triple.
This new kind of RDF term is called a **triple term**.
In this way, an RDF graph can include facts about propositions,
without committing to the truth value of the proposition in the context of the graph.

this is a term,
it is not an asserted triple,
it denotes a proposition
with unknown truth value.

Extending RDF with triple terms

In RDF 1.2 triple terms are restricted, just as literal terms,
to appear only in object position of an RDF triple.

```
triple          ::= subject predicate object
subject         ::= iri | blankNode
predicate       ::= iri
object          ::= term
term            ::= iri | blankNode | literal | tripleTerm
tripleTerm      ::= triple
assertedTriple  ::= triple
graph           ::= assertedTriple*
```

In **Turtle 1.2** a triple term is written as: `<<( subject predicate object )>>`

```
<<( :moon :made_of :cheese )>>
```

Formal semantics of triple terms

Interpretation $\mathcal{I}$:

IR = {richard, liz, moon, cheese, p1, p2, ...}
IP = {married, made_of}
IEXT = {married ↦ <liz, richard>}
IS = { `:liz` ↦ liz, `:richard` ↦ richard, `:married` ↦ married,
`:moon` ↦ moon, `:cheese` ↦ cheese, `:made_of` ↦ made_of, ...}
IT = { <moon, made_of, cheese> ↔ p1, <liz, married, richard> ↔ p2, ...}

this triple denotes the **proposition p1** in $\mathcal{I}$, which is **not true** in $\mathcal{I}$

this triple denotes the **proposition p2** in $\mathcal{I}$, which is a **fact** in $\mathcal{I}$ and it is **true** in $\mathcal{I}$

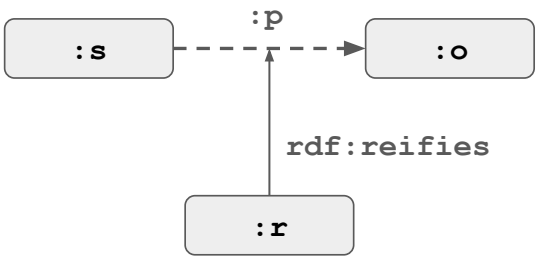
...and then W3C said,
“Let there be reifiers
to separate propositions
from their occurrences”...

Reification

But what can we do with **triple terms** in RDF 1.2?

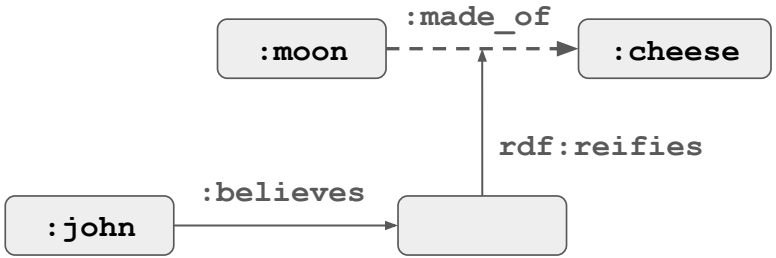
The main use case is **reification**, taken seriously.

Triple terms can serve as the object of a reifying triple
— with the new predicate `rdf:reifies` —
to introduce a **reifier** (the subject of the reifying triple),
which can then be used to assert facts about an occurrence of the proposition
denoted by the triple term itself.




```
:r rdf:reifies <<( :s :p :o )>> .
```

Asserting facts about occurrences of propositions



Turtle 1.2 shortcut:

```
_:b rdf:reifies <<( :moon :made_of :cheese )>> .
:john :believes _:b .
```

⇝

```
:john :believes << :moon :made_of :cheese ~ _:b >> .
:john :believes << :moon :made_of :cheese >> .
```

Example: knowledge graphs

```
@PREFIX cs: <https://cosmopolitan.com/resource/> .
@PREFIX schema: <https://schema.org/> .

<< cs:liz cs:married cs:richard ~ _:wlr_1 >> a cs:marriage ;
   schema:duration [schema:startTime 1964 ; schema:endTime 1974] .

<< cs:liz cs:married cs:richard ~ _:wlr_2 >> a cs:marriage ;
   schema:duration [schema:startTime 1975 ; schema:endTime 1976] .

<< cs:richard cs:married cs:liz ~ _:wlr_1 >> a cs:marriage .
<< cs:richard cs:married cs:liz ~ _:wlr_2 >> a cs:marriage .


cs:richard owl:sameAs cs:rburton .

  ⊨  << cs:rburton cs:married cs:liz ~ _:wlr_2 >> a cs:marriage .
```

Example: multiple support

```
<< :bill-clinton :related-to :hillary-rodham  ~  _:w3 >>
  :starts 1975 .

<< :42nd-potus :husband :1st-female-NY-senator  ~  _:w3 >>
  :starts 1975 .
```

Example: statements about statements

```
<< :paul :loves :mary >> a :belief ; :uttered-by :john .
<< :paul :loves :mary >> a :belief ; :refuted-by :paul .

:john :believes
  << << :liz :spouse :richard ~ :s1 >>
     :starts 1964 >>.
:s1 :certified-by :us-census .

:paul :believes
  << << :liz :spouse :richard >>
     :starts 1955 >>.
```

Example: provenance (I)

```
@PREFIX prov: <http://www.w3.org/ns/prov#> .
@PREFIX lvn: <https://lasvegasnevada.gov/resource/> .

<< :liz :married :richard  ~  lvn:t1 >>
   prov:hadPrimarySource <http://lasvegasnevada.gov/archive/weddings> ;
   prov:wasGeneratedBy lvn:mary ;
   prov:generatedAtTime "1964-03-12Z"^^xsd:date ;
   rdfs:seeAlso lvn:t2 .

<< :liz :married :richard  ~  lvn:t2 >>
   prov:hadPrimarySource <http://lasvegasnevada.gov/weddings> ;
   prov:wasGeneratedBy lvn:paul ;
   prov:generatedAtTime "1975-03-13Z"^^xsd:date ;
   rdfs:seeAlso lvn:t1 .
```

Example: provenance (II)

```
@PREFIX prov: <http://www.w3.org/ns/prov#> .

_:a rdf:reifies <<( :s :p :o )>> .

_:a prov:invalidatedAtTime "2024-09-02T01:31:00Z"^^xsd:dateTime .

_:b rdf:reifies <<( _:a rdf:reifies <<( :s :p :o )>> )>> .

_:b prov:wasAttributedTo :lab-technician-FE-56 .
```

Example: provenance (III) & knowledge graph

```
@PREFIX lvn: <https://lasvegasnevada.gov/resource/> .

<< :liz :married :richard  ~  lvn:t1 >> a lvn:TripleToken .
lvn:t1 :added-in lvn:rdf-registry ; :stored-by :john .
lvn:t1 :on-date "2006-03-12Z"^^xsd:date .
lvn:t1 :provenance lvn:database-1975-123973q2 .

<< :liz :married :richard  ~  lvn:c1 >> a lvn:WeddingCertificate .
lvn:c1 lvn:wedding-date 1975 .

<< :liz :married :richard  ~  lvn:c2 >> a lvn:WeddingCertificate .
lvn:c2 lvn:wedding-date 1964 .

<< :paul :married :susan  ~  lvn:c3 >> a lvn:WeddingCertificate .
lvn:c3 lvn:wedding-date 1971 .

lvn:c1 :printed-on "2020-10-11Z"^^xsd:date .
lvn:c2 :printed-on "2021-03-12Z"^^xsd:date .
lvn:c3 :printed-on "2021-03-12Z"^^xsd:date .
```

...and W3C said, “Let us make reification in our image, after our cognition”...

Reification and Formal Ontologies

- A **proposition** in a **truth-bearer**, while a **reifier** is a **truth-maker**.
- A reifier is a truth-maker for a **relation**: a **relator**.
- A relator is the reified, “thing-like” entity that makes a material relation true.
- A relator is an entity that has its own existence and properties;
  it is what makes a relational proposition true.

- **Example**: A specific weddding between Liz and Richard (the reifier/relator)
  is a truth-maker for the relationship "married”,
  and it is what makes the proposition "Liz married Richard" true.

  The relator is the existence of the marriage itself, not just the direct relationship.

Reification and Natural Language semantics

Richard married Liz.

It was in 1964.

Richard married Liz in 1975.

The former wedding was in Las Vegas.

Its best man was Jim Benton.

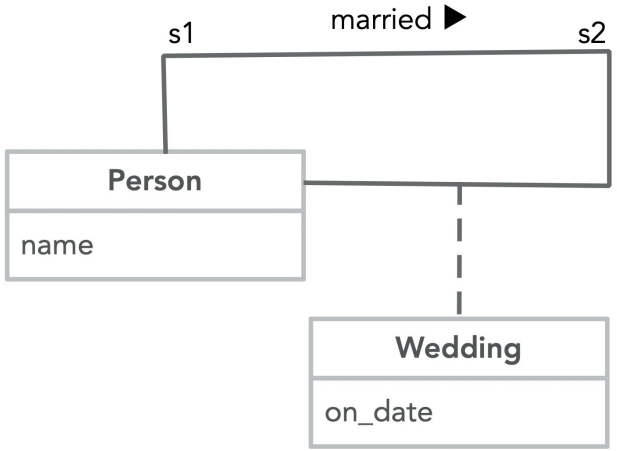
Paul believes that Richard married Liz in 1955, but this is not certified by US census.

… and W3C saw that it was good…

UML: association class

UML association class with OO instance (pointers/memory address)

The schema:



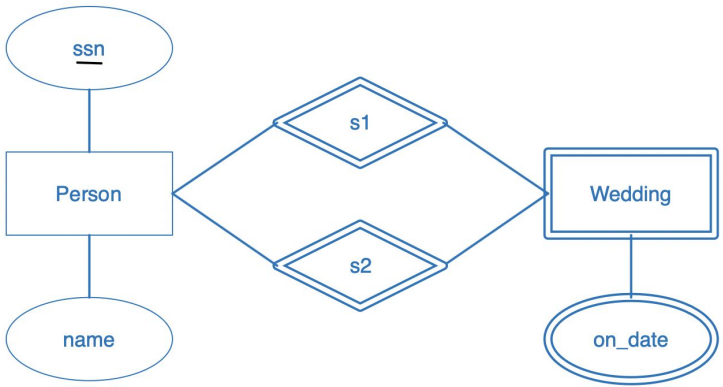
The instances:

MEMORY

location	content
1	a Person
2	name = liz
3	a Person
4	name = richard
5	a Wedding
6	s1 = 1
7	s2 = 3
8	on_date = 1964
9	a Wedding
10	s1 = 1
11	s2 = 3
12	on_date = 1975

ER: weak entity

The schema:



The relations:

Person

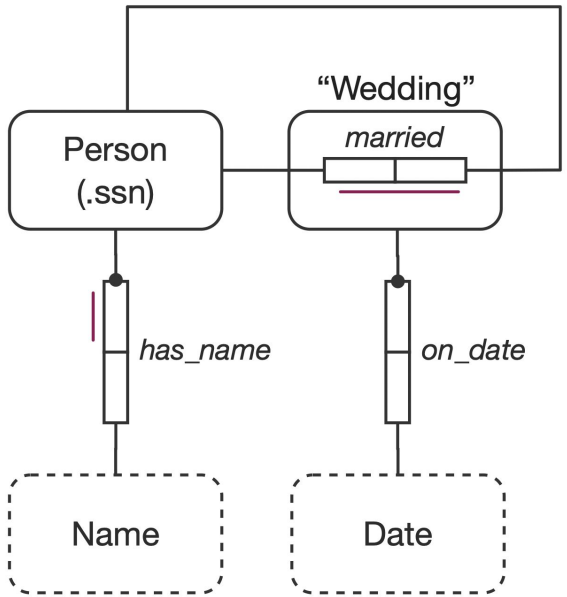
ssn	name
ssn1	liz
ssn2	richard
ssn3	mary
ssn4	paul

Wedding

s1(fk)	s2(fk)	on_date
ssn1	ssn2	1964
ssn1	ssn2	1975
ssn3	ssn4	1975

ORM: objectification

The schema:



The relations:

Person

ssn	name
ssn1	liz
ssn2	richard
ssn3	mary
ssn4	paul

married

s1(fk)	s2(fk)	CID(fk)
ssn1	ssn2	c1
ssn3	ssn4	c2

Wedding

CID(fk)	on_date	WID
c1	1964	w1
c1	1975	w2
c2	1975	w3

n-ary relations (n > 2)

Wedding

s1	s2	on_date
ssn1	ssn2	1964
ssn1	ssn2	1975
ssn3	ssn4	1975

Description Logics (DLR)

```
married(s1:ssn1, s2:ssn2, on_date:1964) .

married(s1:ssn1, s2:ssn2, on_date:1975) .

c1 = <s1:ssn1, s2:ssn2, on_date:1964> .

c2 = <s1:ssn1, s2:ssn2, on_date:1975> .

Wedding = ⊙married .

Wedding(c1) .

Wedding(c2) .

best_man(c1, jim_benton) .

location(c2, las_vegas) .
```

More syntactic sugar: the annotation syntax

```
:s :p :o ~ :r {| :a :b; :c :d |}.
```

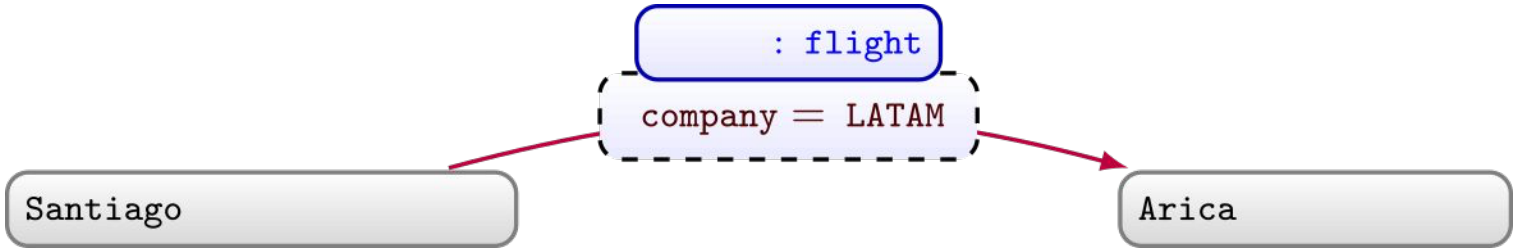
is a shortcut for

```
:s :p :o.
<< :s :p :o ~ :r >> :a :b ; :c :d.
```

which itself is a shortcut for

```
:s :p :o.
:r rdf:reifies <<( :s :p :o )>>.
:r :a :b ; :c :d.
```

Relation to Property Graphs (1)



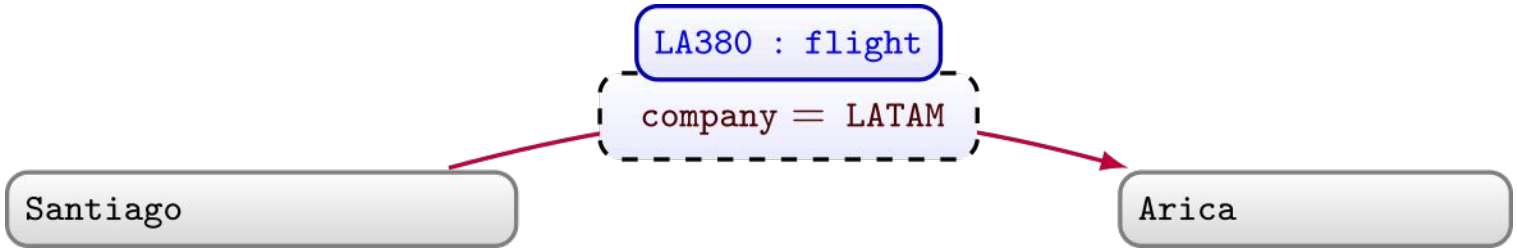
In Cypher:

```
(Santiago) -[:flight {company:"LATAM"}]-> (Arica)
```

In Turtle:

```
:Santiago :flight :Arica {| :company “LATAM” |} .
```

Relation to Property Graphs (2)



In Cypher:

```
(Santiago) -[LA380:flight {company:"LATAM"}]-> (Arica)
```

In Turtle:

```
:Santiago :flight :Arica ~ :LA380 {| :company “LATAM” |} .
```

Relation to Property Graphs (3)

In Cypher:

```
(Santiago) -[:flight {fn: "LA380", company:"LATAM"}]-> (Arica),
(Santiago) -[:flight {fn: "H2300", company:"Sky Airlines"}]-> (Arica),
(Santiago) -[:flight {fn: "LA225", company:"LATAM"}]-> (Temuco)
```

In Turtle:

```
:Santiago :flight :Arica  {| :fn "LA380", :company "LATAM" |}
                          {| :fn "H2300", :company "Sky Airlines" |} ,
                  :Temuco {| :fn "LA225", :company "LATAM" |} .
```

Relation to RDF 1.1 reification

- RDF 1.2 “triple term-based” and “old style” reification are not semantically related (one does not entail the other)
- They can co-exist
- In “old style” reification, instances of rdf:Statement can be thought of as a specific kind of reifier...
- ... but RDF 1.2 reifiers can represent many other things (e.g. flights)

Difference to RDF* / RDF-star

- RDF 1.2's triple terms are almost identical to RDF*'s embedded triples / RDF-star's quoted triples
  - "naming things is one of the two most difficult problems in computer science"
  - RDF 1.2 only allow triple terms in the **object** position

- Turtle 1.2 is significantly different from Turtle* / Turtle 1.2

	`<< :s :p :o >>`	`:s :p :o {| :a :b |}`	`<<( :s :p :o )>>`
Turtle*	triple term	🚫	🚫
Turtle-star		properties on the triple term itself	🚫
Turtle 1.2	reifier	properties on a reifier	triple term

Outline

- 👋 Introduction and overview
- ⏳ History and motivation
- 🪆 Foundations of triple terms and reification
- ❓ Discussion and questions
- ☕ Break
- ⚙️ Triple terms and reification in SPARQL
- 🍇 Overview of smaller additions and changes
- ❓ Discussion and questions