Publishing and Integrating Data on the Semantic Web

Gilles Falquet

Topics

- Publication methodology
- From RDB and others to RDF
  - Direct, R2RML, RML
- SPARQL as a graph transformation language
- Data integration and query rewriting

Publication methodology: Main activities [1]

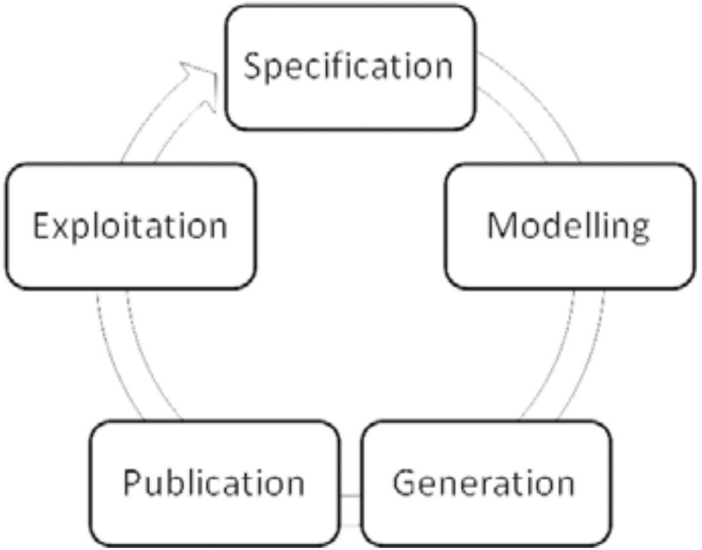


Fig. 1 Main Activities for Publishing Government Linked Data

[1] Villazón-Terrazas B., Vilches-Blázquez L.M., Corcho O., Gómez-Pérez A. (2011) Methodological Guidelines for Publishing Government Linked Data. In: Wood D. (eds) Linking Government Data. Springer, New York, NY. https://doi.org/10.1007/978-1-4614-1767-5_2

SPECIFICATION

- Software/application requirements
- **Precise identification of the government Linked Data requirements.**

  (1) identification and analysis of the government data sources,
  (2) URI design
  (3) definition of license.

URI Design (https://www.w3.org/TR/cooluris/#semweb)

Distinguishing between Representations and Descriptions

- URIs can identify
  - a thing (which may exist outside of the Web) – e.g. Alice
  - a Web document *describing* the thing. – e.g. the homepage of Alice

- The hash technique
  - http://www.example.com/about#alice – Alice
  - http://www.example.com/home/alice -- Alice's homepage

The 303 technique

a) If an "http" resource responds to a GET request with a 2xx response, then the resource identified by that URI is an information resource;

b) If an "http" resource responds to a GET request with a 303 (See Other) response, then the resource identified by that URI could be any resource;

c) c) If an "http" resource responds to a GET request with a 4xx (error) response, then the nature of the resource is unknown.

https://lists.w3.org/Archives/Public/www-tag/2005Jun/0039.html

URI Design Guidelines. Eg. for publishing government data

- Use meaningful URIs, instead of opaque URIs, when possible.
  - to put into the URI as many information as possible.

- Separate the schema from the instances' URIs.
  - http://data.gov.bo/schema#City (the class City)
  - http://data.gov.bo/resource/city#Dublin (a city)
    - use *Patterned URIs* by adding the class name to the schema base URI.

- Use the main official language of the government, when possible.

MODELLING

1. Search for suitable vocabularies to reuse
   - Repositories: LOV, …
   - Well known vocabularies: schema.org, …

2. In case that we did not find any suitable vocabulary
   - create them, trying to reuse as much as possible existing resources, e.g.,
     - government catalogues, vocabularies available at sites, …

3. If we did not find available vocabularies nor resources
   - create the schema/ontology from scratch.
     - ontology design methodologies

GENERATION

- take the data sources selected in the specification activity
- transform them to RDF according to the vocabulary created in the modelling activity
- tasks:
  - (1) transformation,
  - (2) data cleansing, and
  - (3) linking.

GENERATION: Transformation

CSV to RDF

- e.g. GraphDB upload

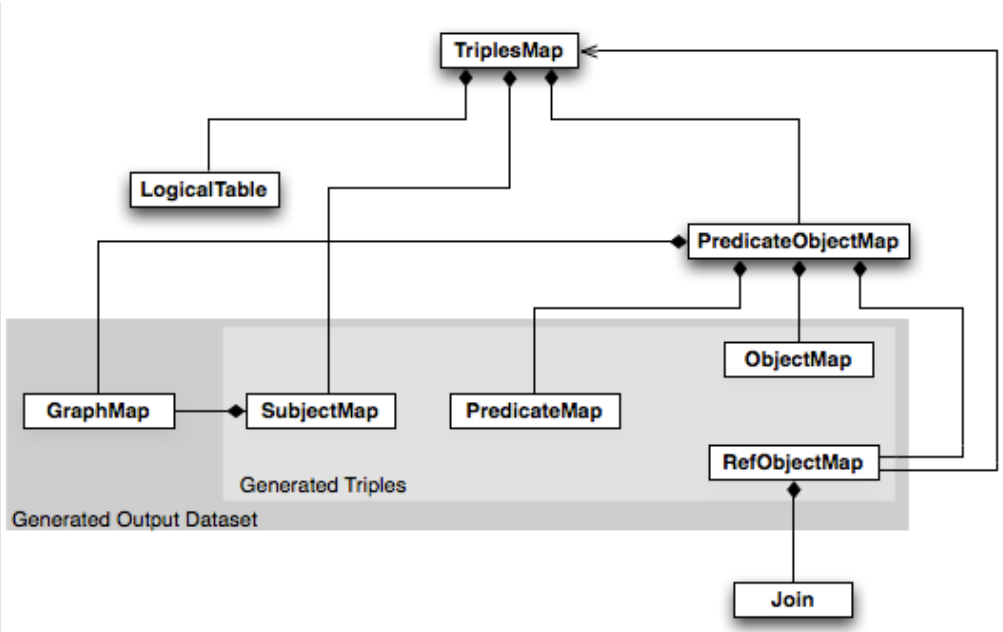
Relational databases to RDF

- direct mapping https://www.w3.org/TR/rdb-direct-mapping/
- R2RML https://www.w3.org/TR/r2rml/
  - language to specify mappings

Anything to RDF

- RML : extension of R2RML

R2RML


TriplesMap
LogicalTable
PredicateObjectMap
ObjectMap
GraphMap
SubjectMap
PredicateMap
RefObjectMap
Generated Triples
Generated Output Dataset
Join

EMP			
EMPNO INTEGER PRIMARY KEY	ENAME VARCHAR(100)	JOB VARCHAR(20)	DEPTNO INTEGER REFERENCES DEPT (DEPTNO)
7369	SMITH	CLERK	10

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://example.com/ns#>.
<#TriplesMap1>
 rr:logicalTable [ rr:tableName "EMP" ];
 rr:subjectMap [
   rr:template "http://data.example.com/employee/{EMPNO}";
   rr:class ex:Employee;
 ];
 rr:predicateObjectMap [
   rr:predicate ex:name;
   rr:objectMap [ rr:column "ENAME" ];
 ].
```

<http://data.example.com/employee/7369> rdf:type ex:Employee.
<http://data.example.com/employee/7369> ex:name "SMITH".

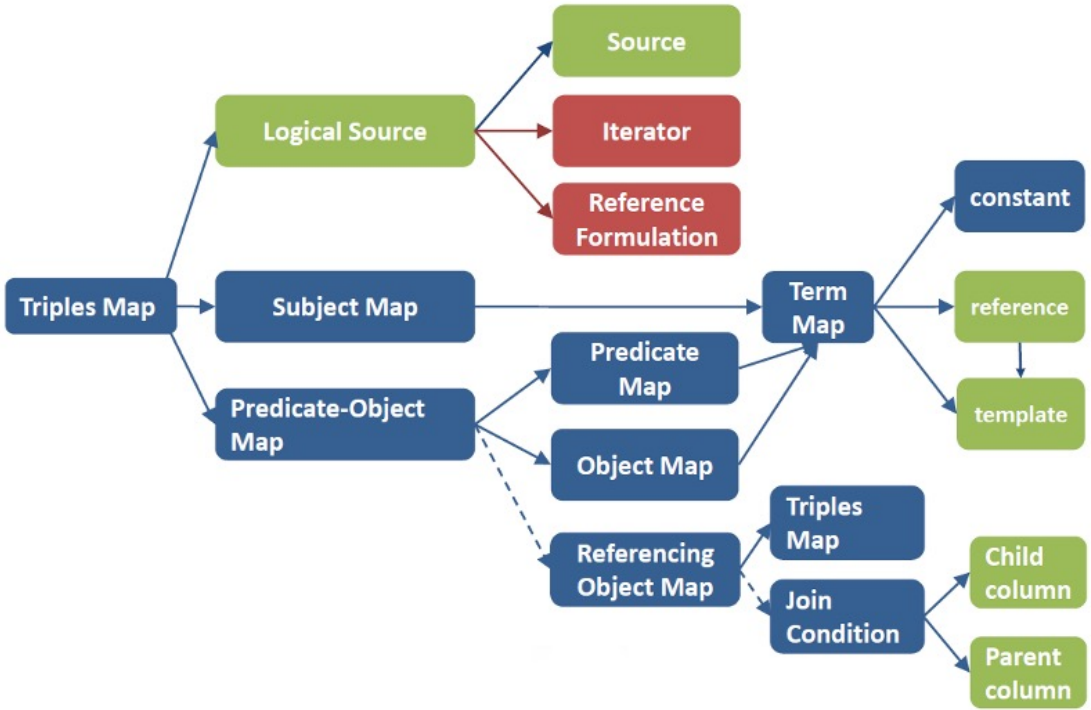
EMP			
EMPNO INTEGER PRIMARY KEY	ENAME VARCHAR(100)	JOB VARCHAR(20)	DEPTNO INTEGER REFERENCES DEPT (DEPTNO)
7369	SMITH	CLERK	10

```
<#TriplesMap1>
  rr:predicateObjectMap [
     rr:predicate ex:department;
     rr:objectMap [
         rr:parentTriplesMap <#TriplesMap2>;
         rr:joinCondition [
            rr:child "DEPTNO";
            rr:parent "DEPTNO";
         ];
    ];
 ].
```

<http://data.example.com/employee/7369> ex:department <http://data.example.com/department/10>.

RML

- Sources can be anything (relational, JSON, XML, …)



RML Logical Source

```
<#PersonMapping>
    rml:logicalSource [
        rml:source "People.json";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.[*].Person" ].
```

Subject Map

```
<#PersonMapping>
   rr:subjectMap [
     rr:template "http://ex.com/Person/{name}_{surname}";
     rr:class ex:Person ].
```

Predicate object map

```
<#PersonMapping>
   rr:predicateObjectMap [
     rr:predicate ex:name;
     rr:objectMap [ rml:reference "name" ]
   ].
```

GENERATION: Data Cleansing

Detect and correct problems

Problems at the URI/HTTP level

- *Dereferencability issues*
- *No structured data available*

Syntax errors in published RDF (Turtle, XML, …)

Errors that hamper reasoning

- *Atypical use of RDFS/OWL constructs (e.g. cyclic rdf:type relation)*
- *Undefined (misspeled) classes and properties*
- *Malformed datatype literals*

Ontology hijacking

- *Changing the meaning of re-used vocabularies through (re)definitions*

Hogan, A. et al. Weaving the Pedantic Web, LDOW2010, April 27, 2010, Raleigh, USA.

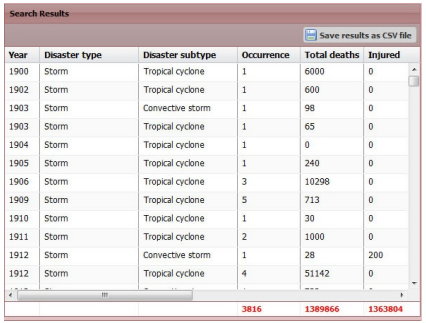
GENERATION: Transformation and integration of sources

- the publication of geodata
- specification of the generation process
- SPARQL as a graph transformation language

Falquet G, Métral, C., Ozaine, S., Giuliani, G. An Abstract Specification Technique for the Publication of Linked Geospatial Data. In Proc. 3Dgeoinfo Conf., Dubai, 2014.

Publishing geodata as linked data

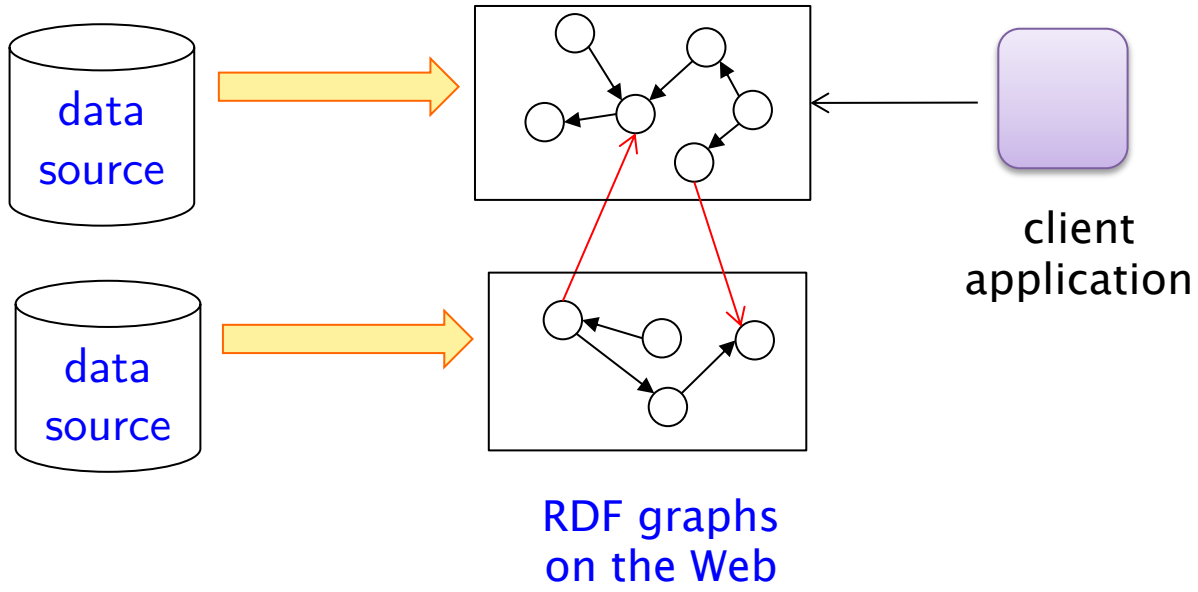
Objective: transform geodata sources into linked geodata



Search Results

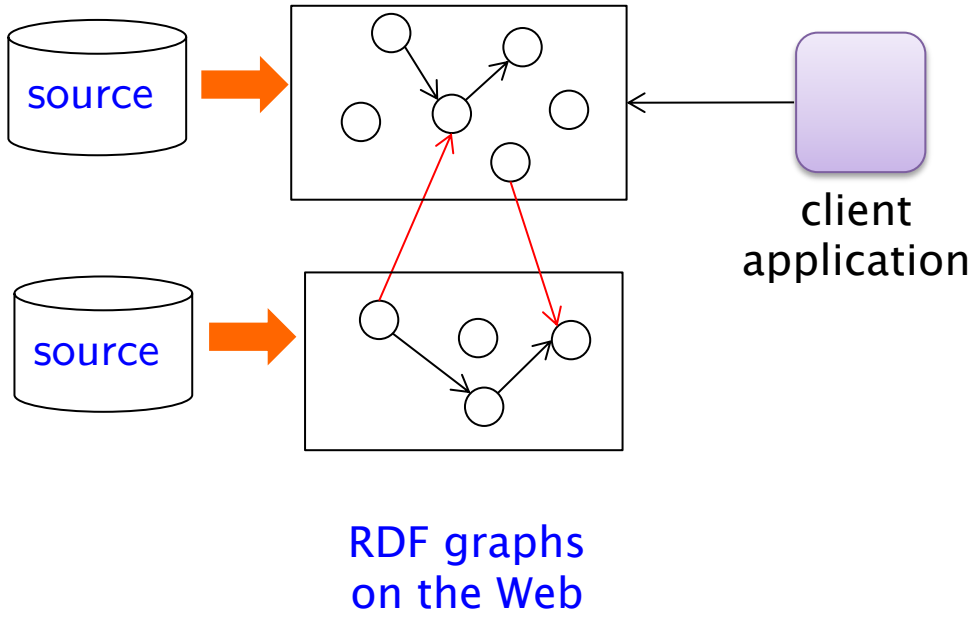
Save results as CSV file

Year	Disaster type	Disaster subtype	Occurrence	Total deaths	Injured
1900	Storm	Tropical cyclone	1	6000	0
1902	Storm	Tropical cyclone	1	600	0
1903	Storm	Convective storm	1	98	0
1903	Storm	Tropical cyclone	1	65	0
1904	Storm	Tropical cyclone	1	0	0
1905	Storm	Tropical cyclone	1	240	0
1906	Storm	Tropical cyclone	3	10298	0
1909	Storm	Tropical cyclone	5	713	0
1910	Storm	Tropical cyclone	1	30	0
1911	Storm	Tropical cyclone	2	1000	0
1912	Storm	Convective storm	1	28	200
1912	Storm	Tropical cyclone	4	51142	0
			3816	1389866	1363804



Publication methodology (Vilaches-Blázquez et al., 2014)

1. specification of requirements
   - data source identification
   - URI design
2. ontology modelling
   (conceptual & data schema design)
3. RDF graph generation
4. link generation
5. data and metadata publication and exploitation



Publishing as Linked Data ≠ Translating to RDF

Published data == a view on the source data

Adapted to the intended use

=> non-trivial operations

- simplifications
- attribute pre-computations
- selection of relevant data
- etc.

Example: Publishing the UNEP PREVIEW+EMDAT

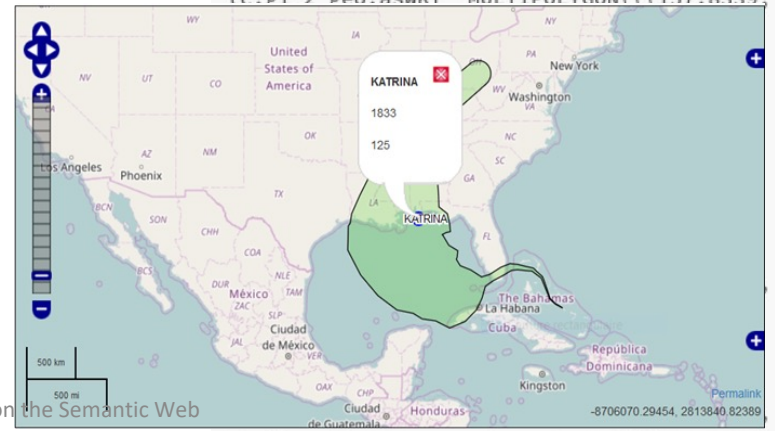
Requirements

can be used to generate integrated cyclone coverage maps, with impact data

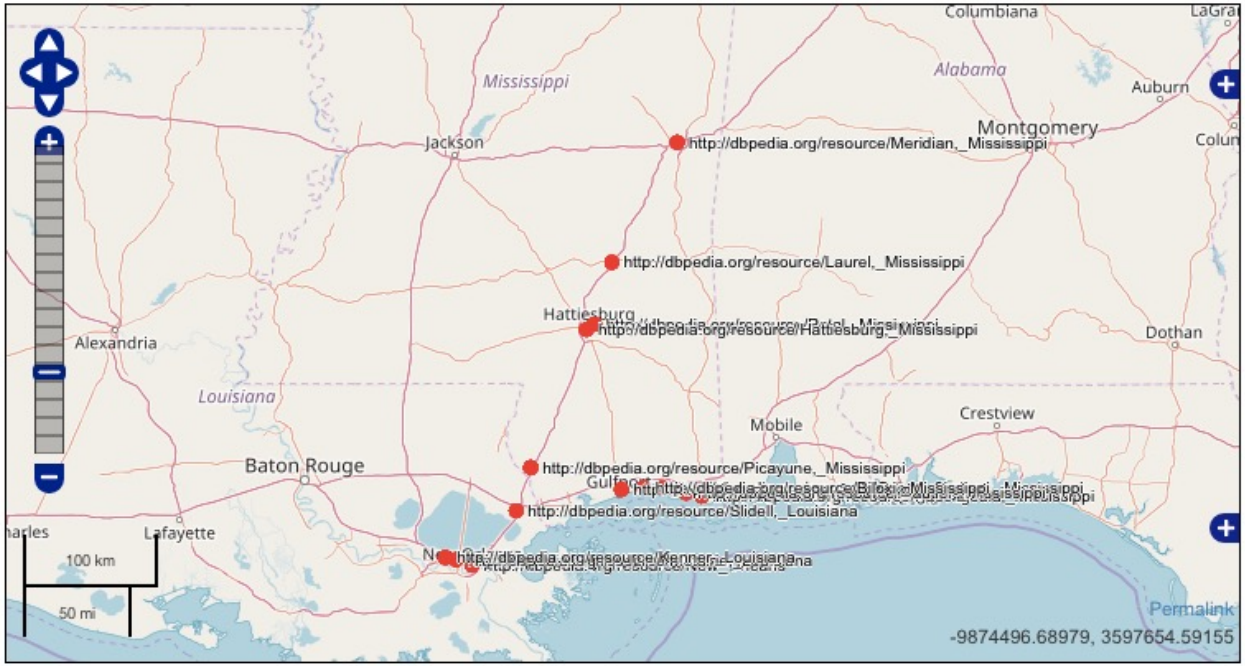
Sources

- PREVIEW: evolution of the spatial extent of tropical cyclones events
- EMDAT: disaster data (affected people, damages, ...)
- World Borders thematicmapping.org
- dbpedia

```
@prefix tc: <http://geolink.grid.unep.ch/tropCyc#> .
@prefix em: <http://geolink.grid.unep.ch/emdatCyc#> .

tc:event1 rdf:type sweet:TropicalCyclone ;
    rdfs:label "ADA" ;
    geo:hasBuffer tc:b1-1 ;
    geo:hasBuffer tc:b1-2 .

tc:b1-1 tc:hasGeometry tc:g1-1 ;
  sweet:SSScale "0"

tc:g1-1 geo:asWKT "MULTIPOLYGON(((162.8339, ...)))"

tc:b1-2 tc:hasGeometry tc:g1-2 ;
  sweet:SSScale "1"

tc:g1-2 geo:asWKT "MULTIPOLYGON(((157.8339, ...)))"
```



A geospatial query using the GEOSPARQL standard, against a DBpedia dataset of the cities with more than 10'000 inhabitants. The query returns every city affected by the Hurricane Katrina in 2005, when reaching the Saffir-Simpson category 2. The popups show the websites of affected cities and their population




```
SELECT ?citywkt ?name ?web ?pop WHERE
{ ?city geo:hasGeometry ?geom;
city:cityName ?name;
city:website ?web;
city:pop ?pop.
?geom geo:asWKT ?citywkt .
?cyc rdfs:label 'KATRINA';
tropcyc:hasBuffer ?buffer.
?buffer sweet:SaffirSimpsonScale '2';
geo:hasGeometry ?cycGeom.
?cycGeom geo:asWKT ?cycwkt.
FILTER (geof:sfWithin(?citywkt, ?cycwkt))
}
```

The publication process

How to structure and formalize this kind of description ?

"... We decided thus to produce a simplified dataset including the whole extent of all events for each Saffir-Simpson category by merging the corresponding country-clipped polygons by means of GIS geoprocessing tools, using ArcGIS 10.2.2. and QGIS 2.10. The number of vertices of the resulting polygons was then reduced using the “Simplify geometries” tool of QGIS, with a 0.001 tolerance. The name of affected countries was eventually reattributed to each resulting polygon from the “TM World Borders” shapefile retrieved from the thematicmapping.org website, using the ArcGIS Spatial join tool and its Join-One-To-Many option, as some cyclones hit several countries during their lifetime.

...

... "

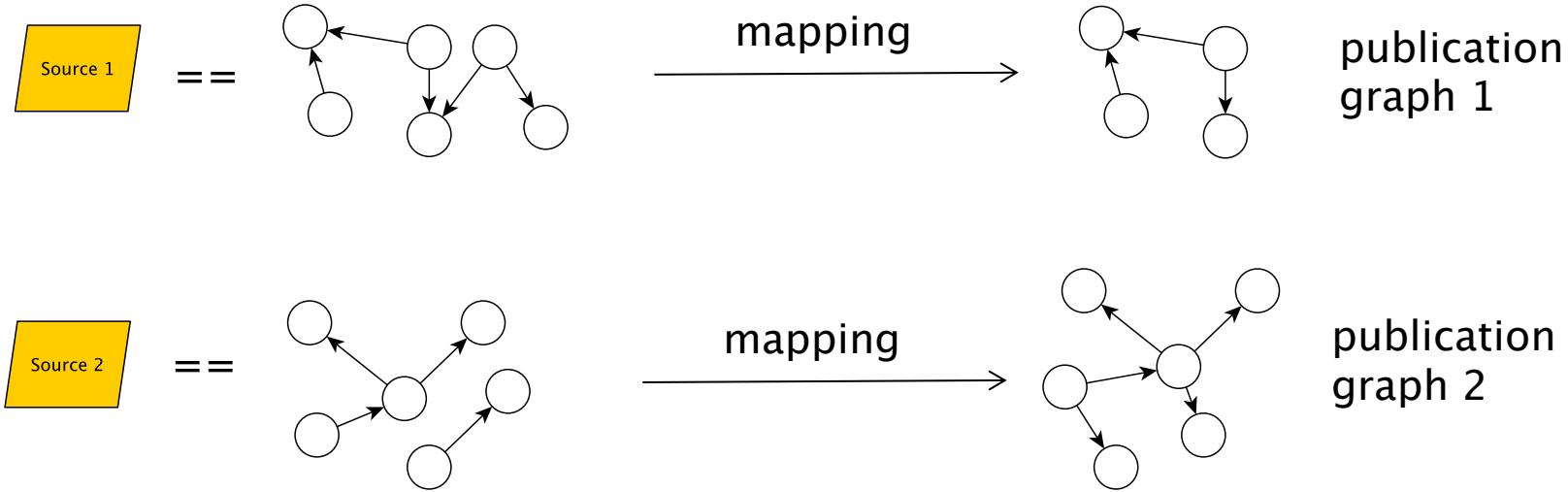
Research question

Can we obtain a specification of the publication process that is

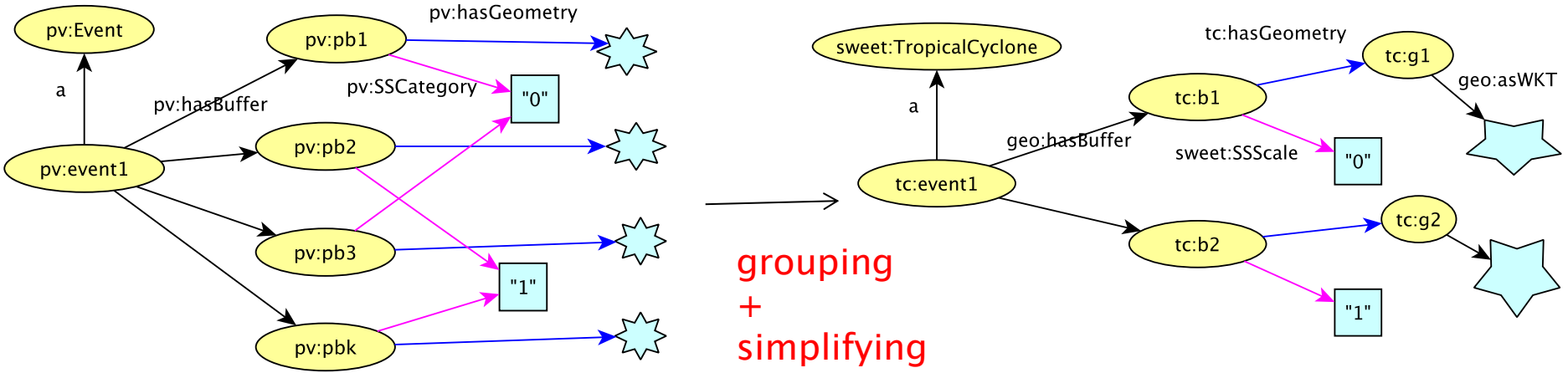
- **abstract**
  - independent of specific systems and implementations
- **declarative**
  - => easier to write/read than a procedural specification
- **formal**
  - can be checked and reasoned upon
  - can be stored as metadata an queried with a query language
- **“compatible”** with the Semantic web

Principe: Linked data publication as graph transformation

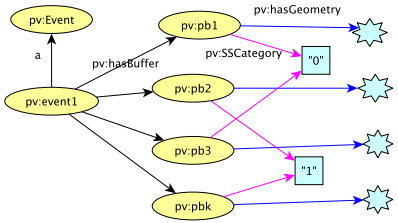
Specify the publication process as a (RDF) graph mapping



Example: Publishing EMDAT events



A mapping specification language



source:

graph pattern (s_var, ...)

group: var, ...

publication

IRI:

p_var = IRI(s_var, ...), ...

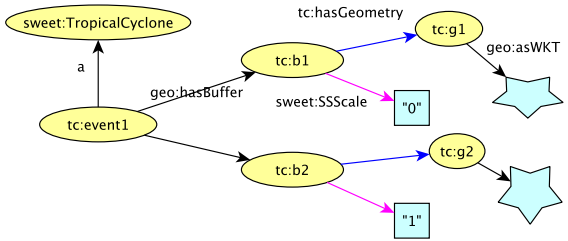
value:

p_var = Expr(s_var, ...), ...

triples:

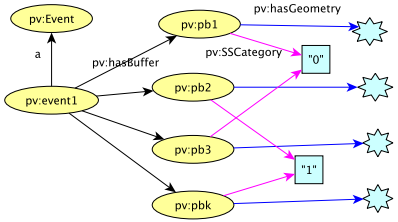
(s_var, ..., p_var, ...)

generation of IRIs for the new nodes
(injective functions)



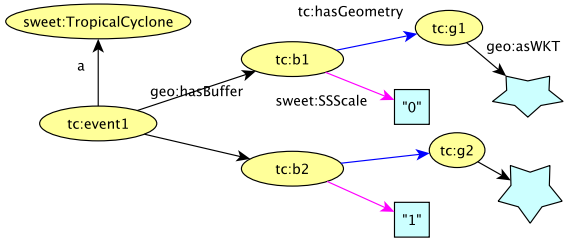
(geo) computation of new literal values

Example



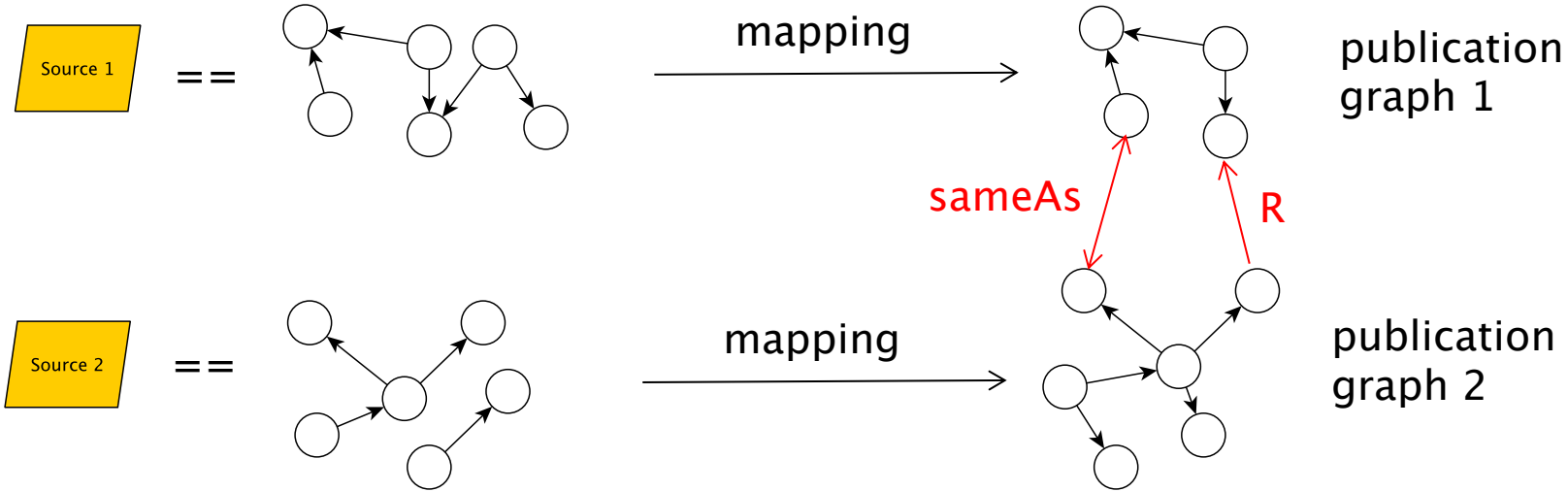
source: {
e a pv:Event; pv:hasBuffer pb.
pb pv:SSCategory cat; pv:hasGeometry pg.
}
group: e, cat

publication
IRI:
c = genCycIRI(e),
b = genBufIRI(e, cat),
g = genGeomIRI(e, cat)
value:
w = toWKT(Simplify(Merge(pg)))
triples: {
c a sweet:TropicalCyclone; tc:hasBuffer b.
b sweet:SSScale cat; geo:hasGeometry g.
g geo:asWKT w. }



Linking data

specify the linking process as edge generation

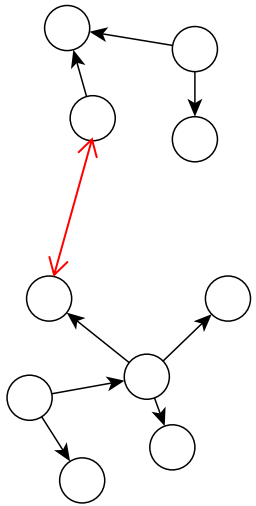


condition:

graph pattern (v_1, …, v_n)

Expr(v_1, …, v_n, x, y)

relation: x R y



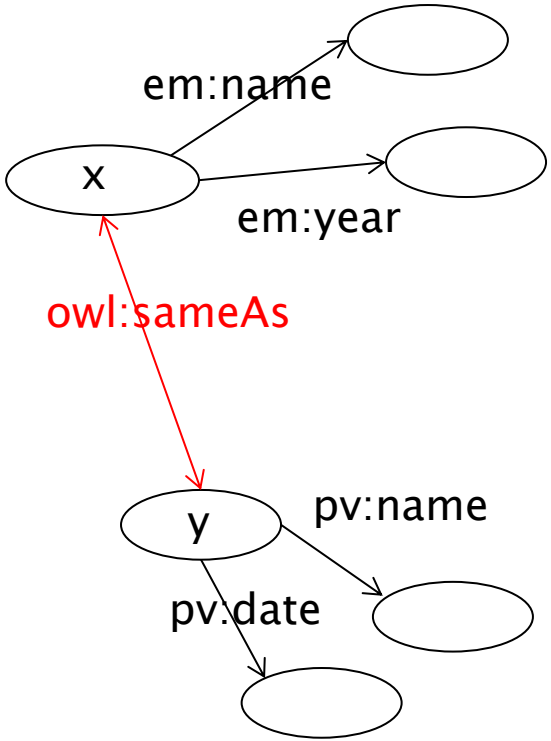
publication graph 1

publication graph 2

Example

condition
{ x a em:Cyclone; em:name xName
y a pv:Cyclone; pv:name yName
},
Similarity(xName, yName) > 0.9

relation {x owl:sameAs prevCyc}



Semantics

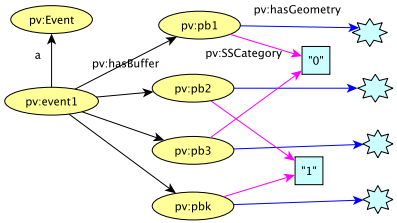
By translation to (abstract) geoSPARQL

source:
graph pattern (s_var, …)
group: var, ...

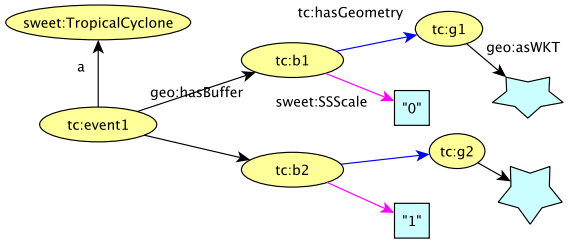
publication
IRI:
p_var = IRI(s_var, ...), ...
value:
p_var = Expr(s_var, ...), ...

triples: {s_var, ..., p_var, ...}

construct
triple patterns(s_var, ..., p_var, ...)
where
graph pattern (s_var, …)
bind(IRI(s_var, ...) as p_var), ...
bind(Expr(s_var, ...) as p_var), ...
group by var, ...

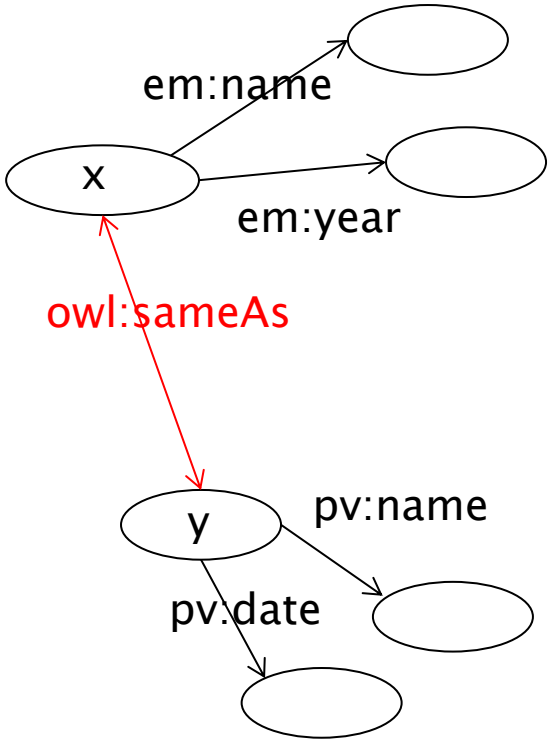



```
construct {
  ?c a sweet:TropicalCyclone; tc:hasBuffer ?b.
  ?b sweet:SSScale ?cat; geo:hasGeometry ?g.
  ?g geo:asWKT ?w.
}
where {
   ?e a pv:Event;  pv:hasBuffer ?pb.
    ?pb pv:SSCategory ?cat; pv:hasGeometry ?pg.

    bind(genCycIRI(?e) as ?c)
    bind(genBufIRI(?e, ?cat) as ?b)
    bind(genGeoIRI(?e, ?cat) as ?g)
    bind(toWKT(Simplify(Merge(?pg))) as ?w)
}
group by ?cat
```



linking with geoSPARQL constructs

```
construct {?x owl:sameAs ?prevCyc}

where {?x a em:Cyclone; em:name ?xName.
   ?y a pv:Cyclone; pv:name ?yName.
  filter(Similarity(?xName, ?yName) > 0.9)
}
```



GENERATION: Summary

- from sources to RDF – with mapping tools/languages
- from generated RDF to published RDF
  - graph transformation can be implemented (mostly) with SPARQL CONSTRUCT
- apply error detection and correction techniques on published RDF