Institut Mines-Télécom

Overview of the SPARQL-Generate language and latest developments

Maxime Lefrançois
Maxime.Lefrancois@emse.fr
http://maxime-lefrancois.info/

MINES Saint-Étienne – Institut Henri Fayol
Laboratoire Hubert Curien UMR CNRS 5516

MINES Saint-Étienne
Une école de l'IMT

LABORATOIRE HUBERT CURIEN
UMR • CNRS • 5516 • SAINT-ETIENNE

2005 2008 2009 2010 2014 2017

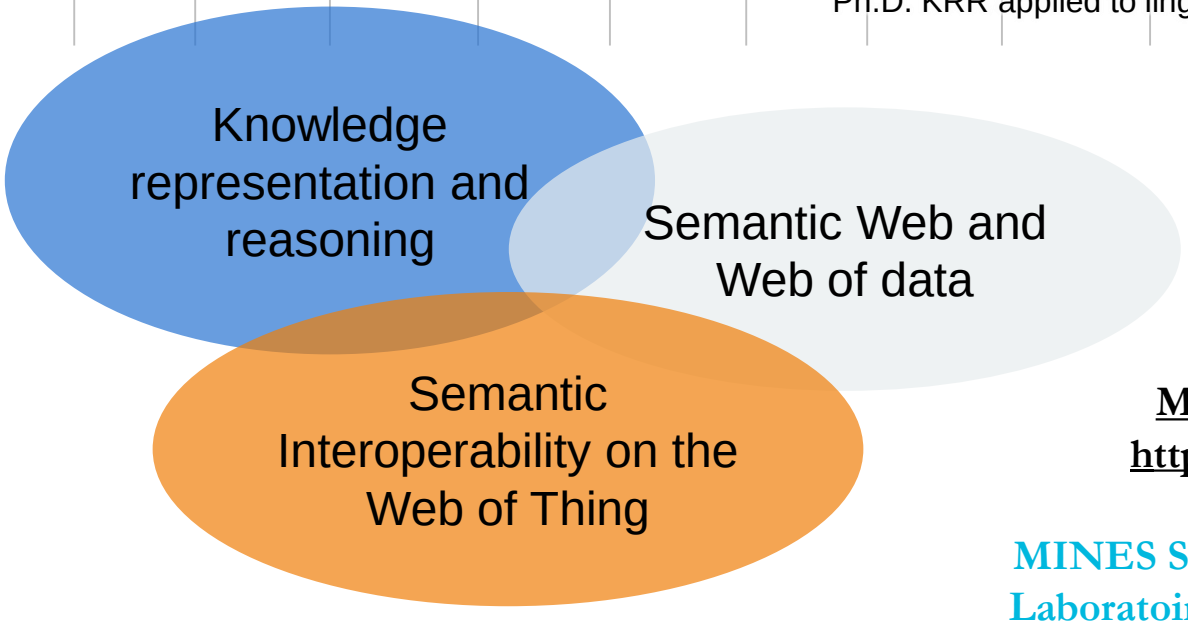
Agrégation in **mechanical engineering**

M2 **Signal processing**

M2 **Computer science**

Ph.D. KRR applied to linguistics

Semantic Web and interoperabiliity



Maxime Lefrançois
Maxime.Lefrancois@emse.fr
http://maxime-lefrancois.info/

MINES Saint-Étienne – Institut Henri Fayol
Laboratoire Hubert Curien UMR CNRS 5516

LABORATOIRE HUBERT CURIEN

UMR • CNRS • 5516 • SAINT-ETIENNE

Connected Intelligence Team
Hubert Curien Laboratory

https://connected-intelligence.univ-st-etienne.fr/

CONNECTED INTELLIGENCE

13 permanent staff members – 11 Ph.D. students (2 open pos. 2020) – 1 postdoc (2 open pos. now)

Contact: Olivier.Boissier@emse.fr

MINES Saint-Étienne
Une école de l'IMT

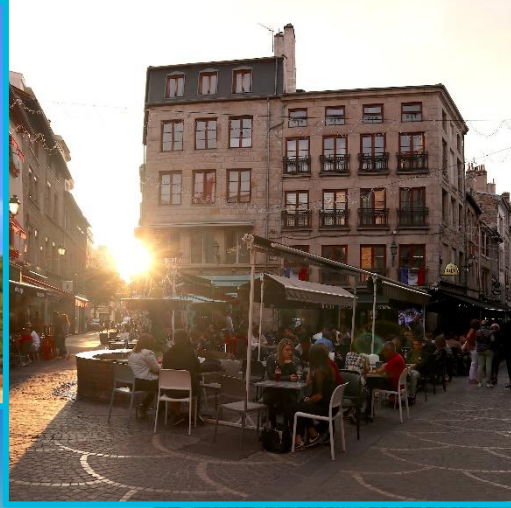
UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

UNIVERSITÉ DE LYON

cnrs

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech







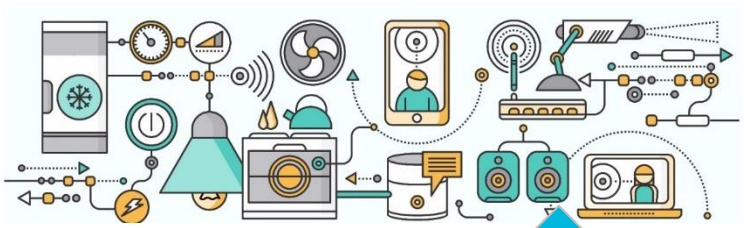





2
T-ETIENNE (LOIRE), 8 DECEMBRE 2018

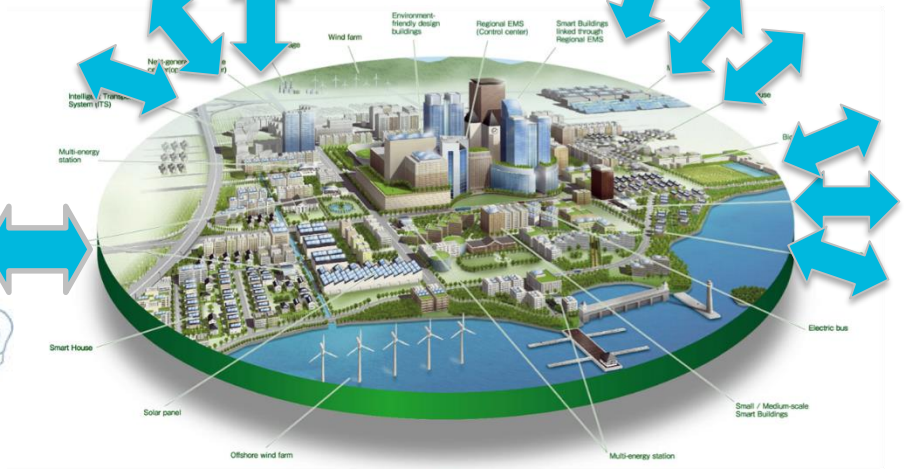
How to reach semantic interoperability at the data level between heterogeneous things and services?

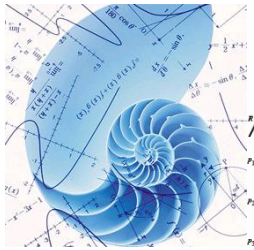
Smart Things



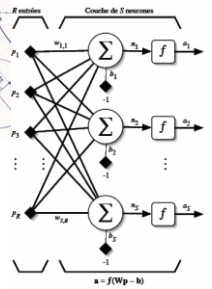
Humans

Data

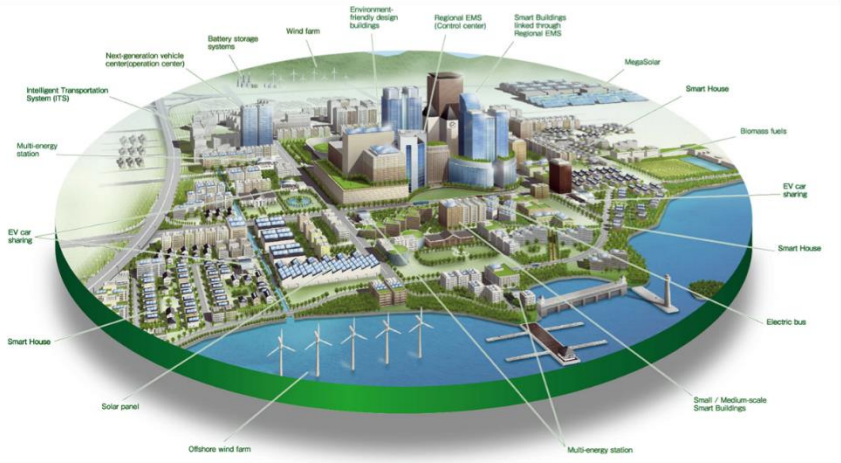


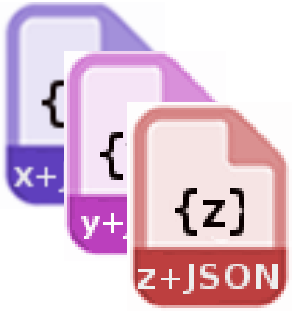


Services

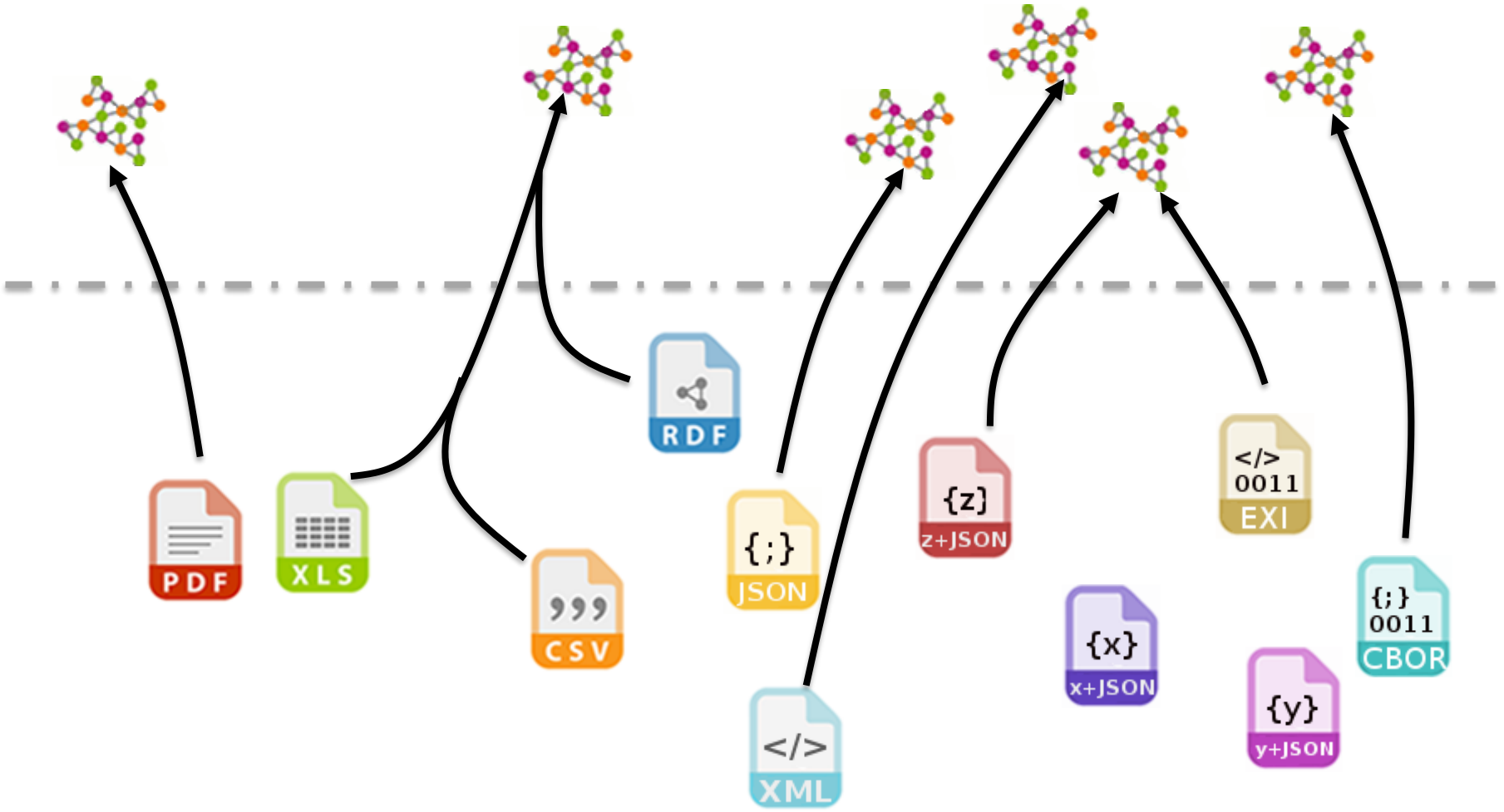


There is a multitude of data formats/models on the Web





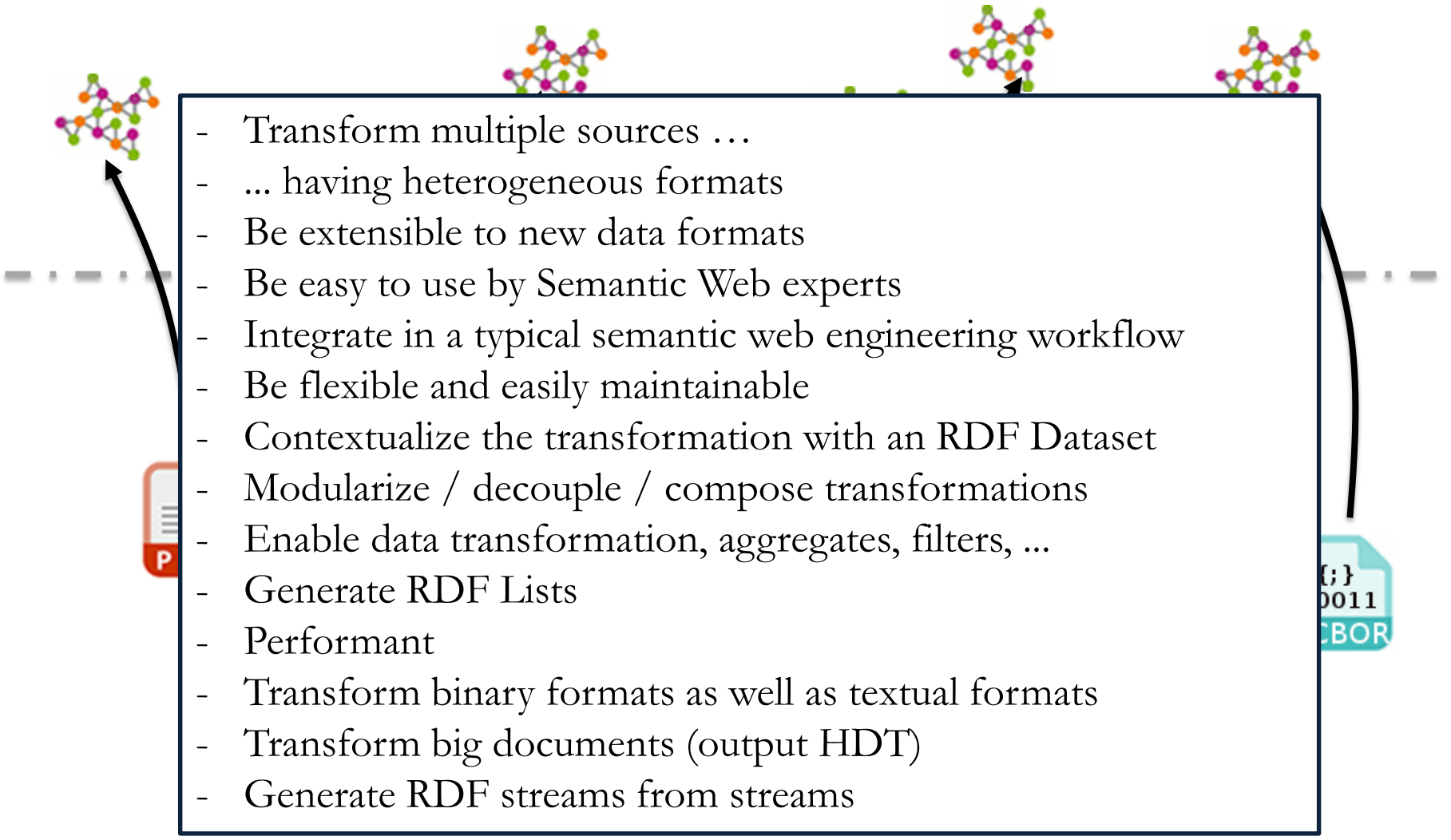
Key step: generate some RDF



M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, In Proc. Extended Semantic Web Conference, 2017

+ EKAW, Hackatons, ESWC Tutorial 2018, PFIA Tutorial 2019, OEG Tutorial ☺

Requirements for RDF generation



- Transform multiple sources …
- ... having heterogeneous formats
- Be extensible to new data formats
- Be easy to use by Semantic Web experts
- Integrate in a typical semantic web engineering workflow
- Be flexible and easily maintainable
- Contextualize the transformation with an RDF Dataset
- Modularize / decouple / compose transformations
- Enable data transformation, aggregates, filters, ...
- Generate RDF Lists
- Performant
- Transform binary formats as well as textual formats
- Transform big documents (output HDT)
- Generate RDF streams from streams

M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, In Proc. Extended Semantic Web Conference, 2017
+ EKAW, Hackatons, ESWC Tutorial 2018, PFIA Tutorial 2019, OEG Tutorial ☺

Existing approaches

RDFizers (see https://www.w3.org/wiki/ConverterToRdf)

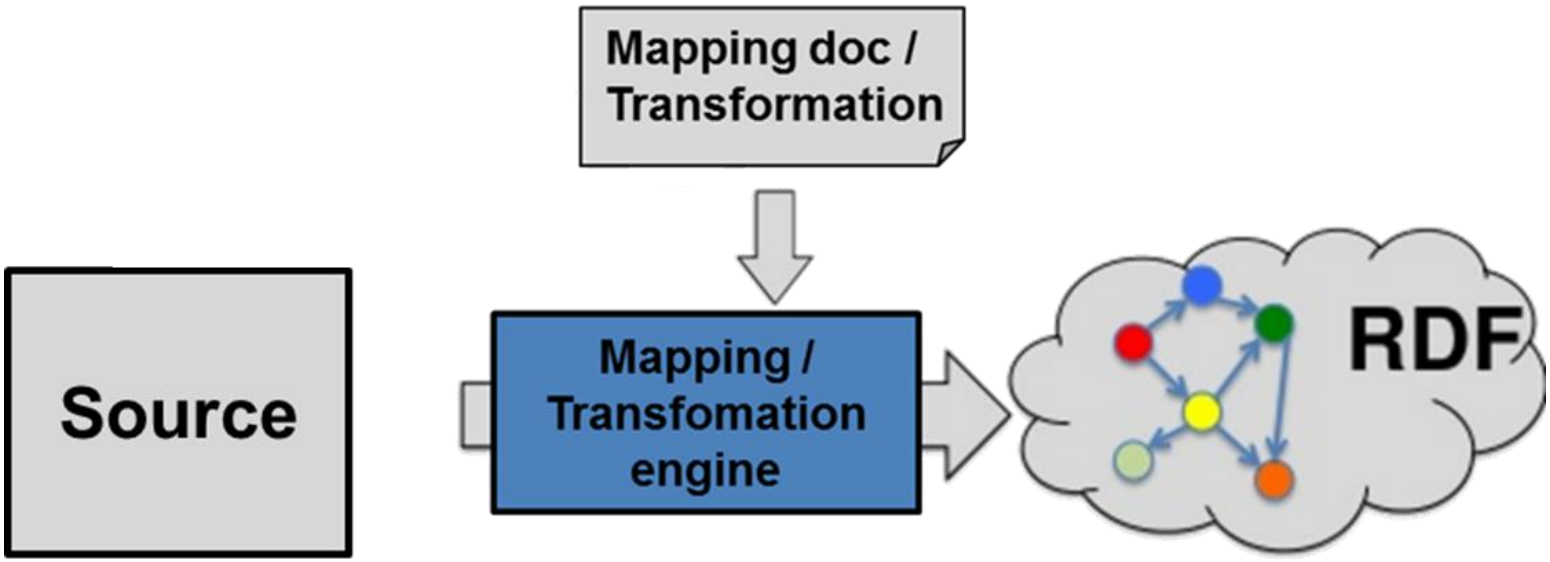
- A lot of tools are specific to one or a few formats (44 referenced formats)
- Some frameworks support several/many formats

⇨ ad hoc methods,
little or no control on the structure of the output
=> may require an additional transformation

M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, In Proc. Extended Semantic Web Conference, 2017
+ EKAW, Hackatons, ESWC Tutorial 2018, PFIA Tutorial 2019, OEG Tutorial ☺

Existing approaches

Approaches based on mapping/transformation languages



M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, In Proc. Extended Semantic Web Conference, 2017
+ EKAW, Hackatons, ESWC Tutorial 2018, PFIA Tutorial 2019, OEG Tutorial ☺

```
<html xmlns=http://www.w3.org/1999/xhtml
   xmlns:grddl='http://www.w3.org/2003/g/data-view#'
   grddl:transformation="http://example.com/getAuthor.xsl" >
  <head>
    <title>Are You Experienced?</title>
    [...]
</html>

<album xmlns:grddl='http://www.w3.org/2003/g/data-view#'
   grddl:transformation="http://example.org/getAlbum.xsl" >
  <artist mbid="">The Jimi Hendrix Experience</artist>
  <name>Are You Experienced?</name>
    ...
</album>
```

GRDDL

(W3C REC 2007)

```
declare namespace foaf="http://xmlns.com/foaf/0.1/";
declare namespace rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#";
let $persons := //*[@name or ../knows]
return
<rdf:RDF>
 {
 for $p in $persons
 let $n := if( $p[@name] )
           then $p/@name else $p
 let $id := count($p/preceding::*)
           +count($p/ancestor::*)
 where
  not(exists($p/following::*[
      @name=$n or data(.)=$n]))
 return
 <foaf:Person rdf:nodeId="b{$id}">
  <foaf:name>{data($n)}</foaf:name>
  {
  for $k in $persons
  let $kn := if( $k[@name] )
             then $k/@name else $k
  let $kid := count($k/preceding::*)
             +count($k/ancestor::*)
  where
   $kn = data(//*[@name=$n]/knows) and
   not(exists($kn/../following::*[
       @name=$kn or data(.)=$kn]))
  return
  <foaf:knows>
   <foaf:Person rdf:nodeID="b{$kid}"/>
  </foaf:knows>
  }
 </foaf:Person>
 }
</rdf:RDF>
```

XSPARQL

(W3C member submission 2009)

R2RML

(W3C REC 2012)

```
<#TriplesMap2>
  rr:logicalTable <#DeptTableView>;
  rr:subjectMap [
    rr:template "http://data.example.com/department/{DEPTNO}";
    rr:class ex:Department;
   ];
  rr:predicateObjectMap [
    rr:predicate ex:name;
    rr:objectMap [ rr:column "DNAME" ];
   ];
  rr:predicateObjectMap [
    rr:predicate ex:location;
    rr:objectMap [ rr:column "LOC" ];
   ];
  rr:predicateObjectMap [
    rr:predicate ex:staff;
    rr:objectMap [ rr:column "STAFF" ];
  ].
```

```
{
  "@context": ["http://www.w3.org/ns/csvw", {"@language": "en"}],
  "url": "tree-ops.csv",
  "dc:title": "Tree Operations",
  "dcat:keyword": ["tree", "street", "maintenance"],
  "dc:publisher": {
    "schema:name": "Example Municipality",
    "schema:url": {"@id": "http://example.org"}
  },
  "dc:license": {"@id": "http://opendefinition.org/licenses/cc-by/"},
  "dc:modified": {"@value": "2010-12-31", "@type": "xsd:date"},
  "tableSchema": {
    "columns": [{
      "name": "GID",
      "titles": ["GID", "Generic Identifier"],
      "dc:description": "An identifier for the operation on a tree.",
      "datatype": "string",
      "required": true
    }, {
      "name": "on_street",
      "titles": "On Street",
      "dc:description": "The street that the tree is on.",
      "datatype": "string"
    }, {
      "name": "species",
      "titles": "Species",
      "dc:descr
      "datatype
    }, {
      "name": "
      "titles":
      "dc:descr
      "datatype
    }, {
      "name": "
      "titles":
      "dc:description": "The date of the operation that was performed.",
      "datatype": {"base": "date", "format": "M/d/yyyy"}
    }],
```

CSVW

(W3C REC 2015)

RML + Extensions

(Dimou et al., 2013)

```
<#PerformancesMapping>
  rml:logicalSource [
    rml:source "http://ex.com/performances.json";
    rml:referenceFormulation ql:JSONPath;
    rml:iterator "$.Performance.[*]" ];

  rr:subjectMap [
     rr:template “http://ex.com/{Perf_ID}”];

 rr:predicateObjectMap [
   rr:predicate ex:location;
   rr:objectMap [
     rr:parentTriplesMap <#LocationMapping> ] ].
```

```
<#EventsMapping>
  rml:logicalSource [
    rml:source "http://ex.com/events.xml";
    rml:referenceFormulation ql:XPath;
    rml:iterator "/Events/Exhibition" ];

  rr:subjectMap [
    rr:template "http://ex.com/{@id}" ];

rr:predicateObjectMap [
    rr:predicate ex:location;
    rr:objectMap [
      rr:parentTriplesMap <#LocationMapping> ] ].
```

RML + Extensions – covers the REQs?

(Dimou et al., 2013)

- Transform multiple sources …
- ... having heterogeneous formats
- Be extensible to new data formats
- Be easy to use by Semantic Web experts
- Integrate in a typical semantic web engineering workflow
- Be flexible and easily maintainable
- Contextualize the transformation with an RDF Dataset
- Modularize / decouple / compose transformations
- Enable data transformation, aggregates, filters, ...
- Generate RDF Lists
- Performant
- Transform binary formats as well as textual formats
- Transform big documents (output HDT)
- Generate RDF streams from streams

- Subject-centric?
- Include parts from different sources in one RDF Term?

Main research questions

How to design a mapping language that…

…is expressive enough to cover all of our use cases?

…is still rather simple to use?

…can be easily extended to any source format?

RDF generation process


```
{"DirectorCompensation":
   [
      {
         "DirectorName":"Jane Doe",
         "Salary":"1,000",
         "Bonus":"1,000",
         "DirectorFees":"1,000",
         "FairValueOfOptionsGranted":"1,000"
      },
      {
         "DirectorName":"John Doe",
         "Salary":"1,000",
         "Bonus":"1,000",
         "DirectorFees":"1,000",
         "FairValueOfOptionsGranted":"1,000"
      },
      {
         "DirectorName":"All Directors",
         "Salary":"2,000",
         "Bonus":"2,000",
         "DirectorFees":"2,000",
         "FairValueOfOptionsGranted":"2,000"
      }
   ]
}
```

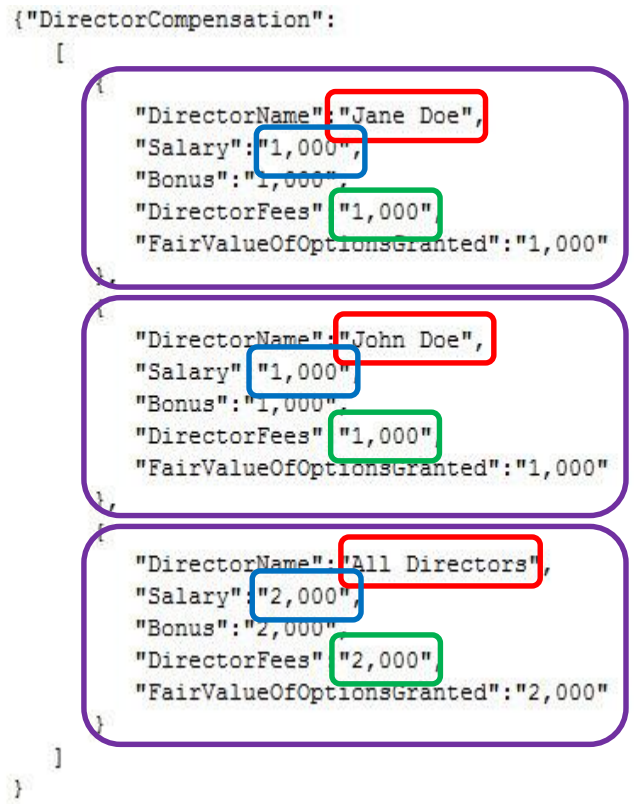
RDF generation process


```
{"DirectorCompensation":
    [
        {
            "DirectorName":"Jane Doe",
            "Salary":"1,000",
            "Bonus":"1,000",
            "DirectorFees":"1,000",
            "FairValueOfOptionsGranted":"1,000"
        },
        {
            "DirectorName":"John Doe",
            "Salary":"1,000",
            "Bonus":"1,000",
            "DirectorFees":"1,000",
            "FairValueOfOptionsGranted":"1,000"
        },
        {
            "DirectorName":"All Directors",
            "Salary":"2,000",
            "Bonus":"2,000",
            "DirectorFees":"2,000",
            "FairValueOfOptionsGranted":"2,000"
        }
    ]
}
```


Selection patterns

Xpath, JSONpath, CSS selectors, regex, etc.

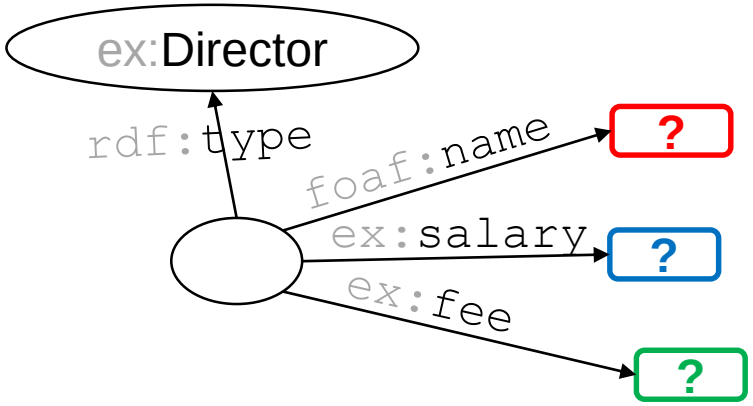
RDF generation process



Selection patterns

Xpath, JSONpath, CSS selectors, regex, etc.

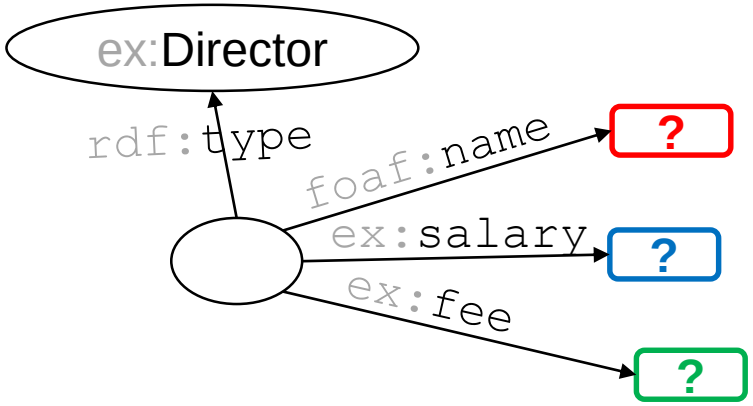
RDF generation process



Selection patterns
Xpath, JSONpath, CSS selectors, regex, etc.

Graph pattern definition

RDF generation process

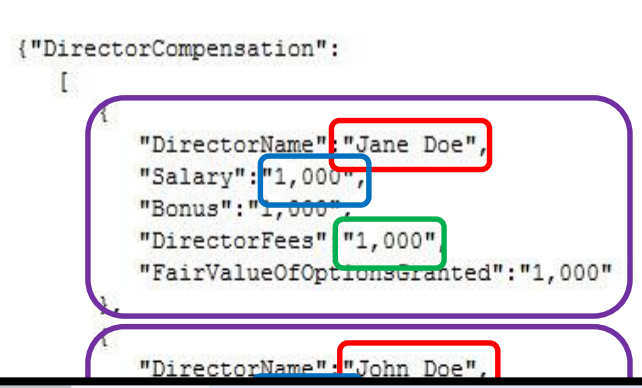


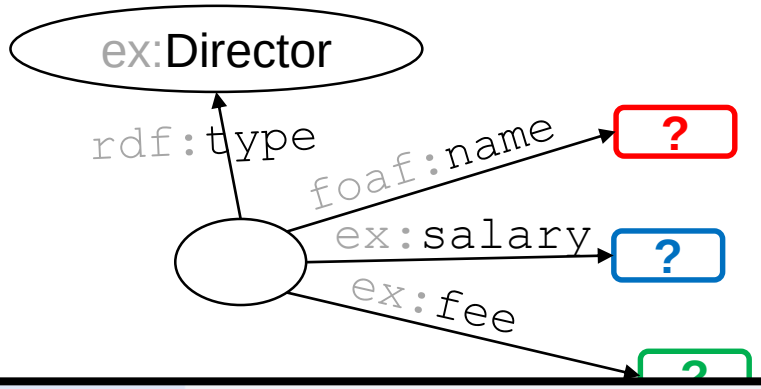
Selection patterns
Xpath, JSONpath, CSS selectors, regex, etc.

Graph pattern definition

+ Select ontologies

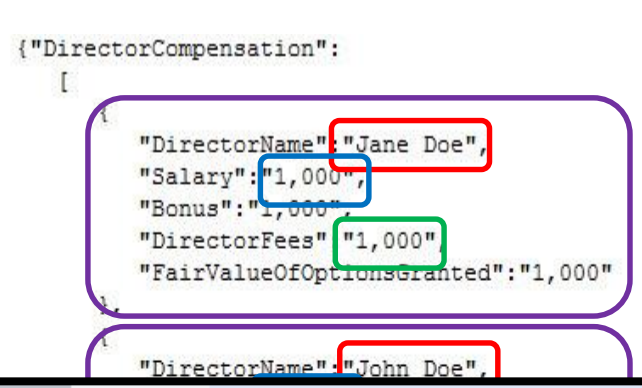
First SPARQL-Generate query

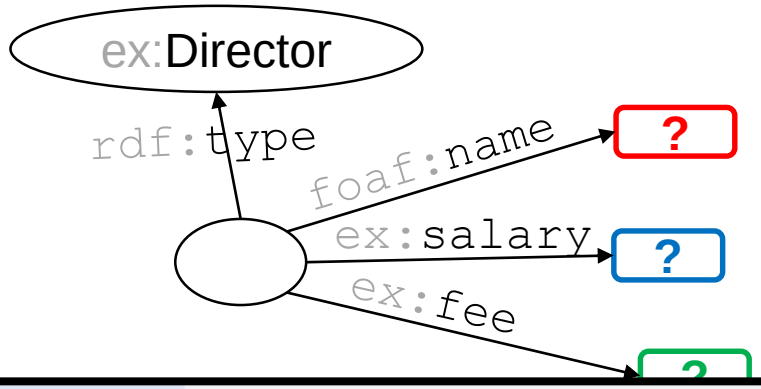





```
PREFIX fun:  <http://w3id.org/sparql-generate/fn/>
PREFIX iter: <http://w3id.org/sparql-generate/iter/>
GENERATE {
  [] a ex:Director ;
    foaf:name ?name ;
    ex:salary ?salary ;
    ex:fee ?fee .
}
SOURCE <myDocument> AS ?myDocument
ITERATOR iter:JSONPath( ?myDocument , "$.DirectorCompensation[*]") AS ?director
WHERE {
  BIND( fun:JSONPath( ?director , "$.DirectorName") AS ?name )
  BIND( fun:JSONPath( ?director , "$.Salary") AS ?salary )
  BIND( fun:JSONPath( ?director , "$.DirectorFee") AS ?fee )
}
```

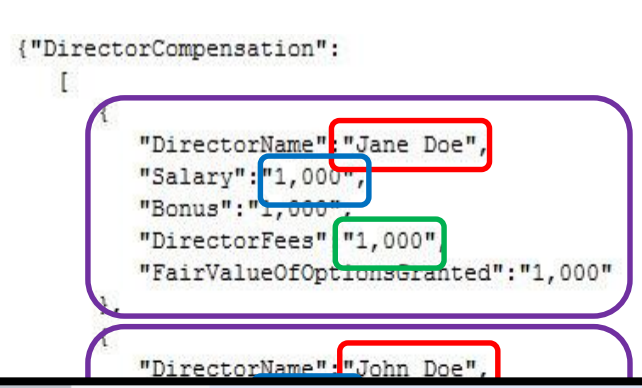
The Graph Pattern Definition is here

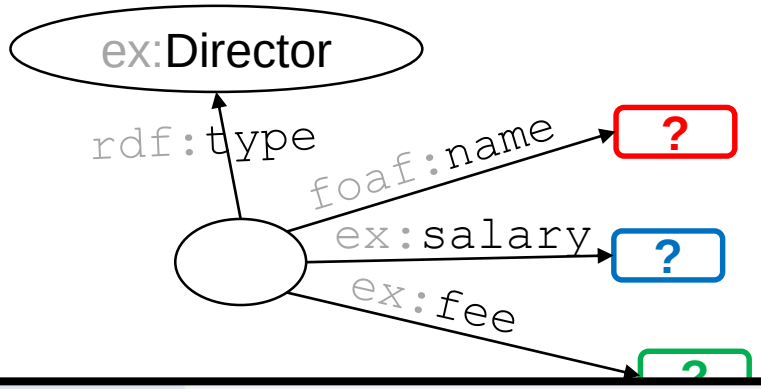





```
PREFIX fun:  <http://w3id.org/sparql-generate/fn/>
PREFIX iter: <http://w3id.org/sparql-generate/iter/>
GENERATE {
  [] a ex:Director ;
    foaf:name ?name ;
    ex:salary ?salary ;
    ex:fee ?fee .
}
SOURCE <myDocument> AS ?myDocument
ITERATOR iter:JSONPath( ?myDocument , "$.DirectorCompensation[*]") AS ?director
WHERE {
  BIND( fun:JSONPath( ?director , "$.DirectorName") AS ?name )
  BIND( fun:JSONPath( ?director , "$.Salary") AS ?salary )
  BIND( fun:JSONPath( ?director , "$.DirectorFee") AS ?fee )
}
```

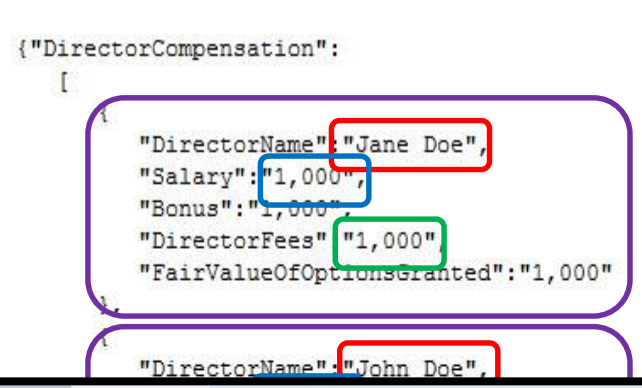
Query RDF Dataset + « Documentset »

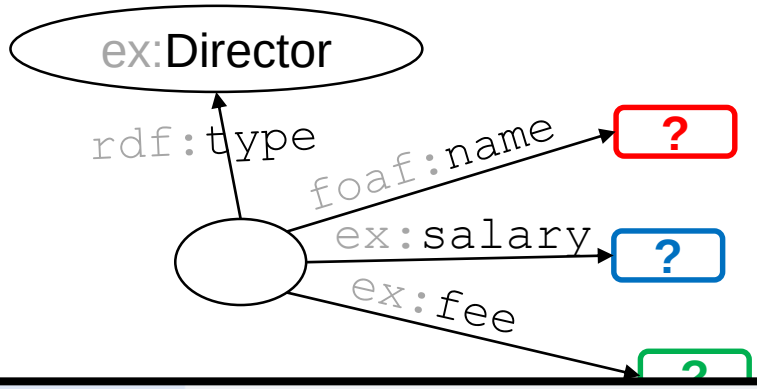





```
PREFIX fun:  <http://w3id.org/sparql-generate/fn/>
PREFIX iter: <http://w3id.org/sparql-generate/iter/>
GENERATE {
  [] a ex:Director ;
    foaf:name ?name ;
    ex:salary ?salary ;
    ex:fee ?fee .
}
SOURCE <myDocument> AS ?myDocument
ITERATOR iter:JSONPath( ?myDocument , "$.DirectorCompensation[*]") AS ?director
WHERE {
  BIND( fun:JSONPath( ?director , "$.DirectorName") AS ?name )
  BIND( fun:JSONPath( ?director , "$.Salary") AS ?salary )
  BIND( fun:JSONPath( ?director , "$.DirectorFee") AS ?fee )
}
```

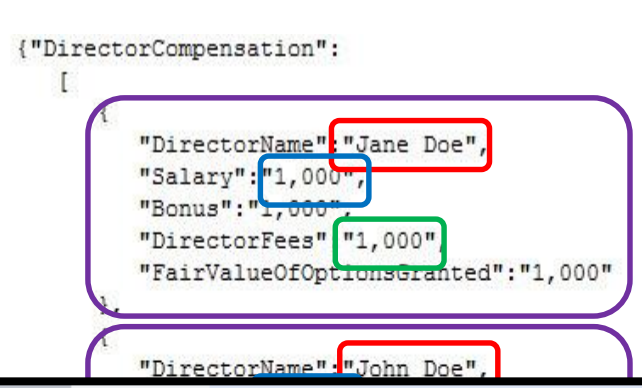
Iterator Functions

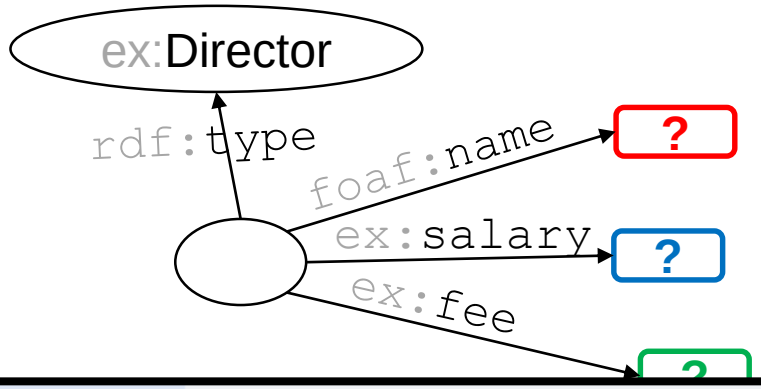





```
1   PREFIX fun:  <http://w3id.org/sparql-generate/fn/>
2   PREFIX iter: <http://w3id.org/sparql-generate/iter/>
3   GENERATE {
4     [] a ex:Director ;
5       foaf:name ?name ;
6       ex:salary ?salary ;
7       ex:fee ?fee .
8   }
9   SOURCE <myDocument> AS ?myDocument
10  ITERATOR iter:JSONPath( ?myDocument , "$.DirectorCompensation[*]") AS ?director
11  WHERE {
12    BIND( fun:JSONPath( ?director , "$.DirectorName") AS ?name )
13    BIND( fun:JSONPath( ?director , "$.Salary") AS ?salary )
14    BIND( fun:JSONPath( ?director , "$.DirectorFee") AS ?fee )
15  }
```

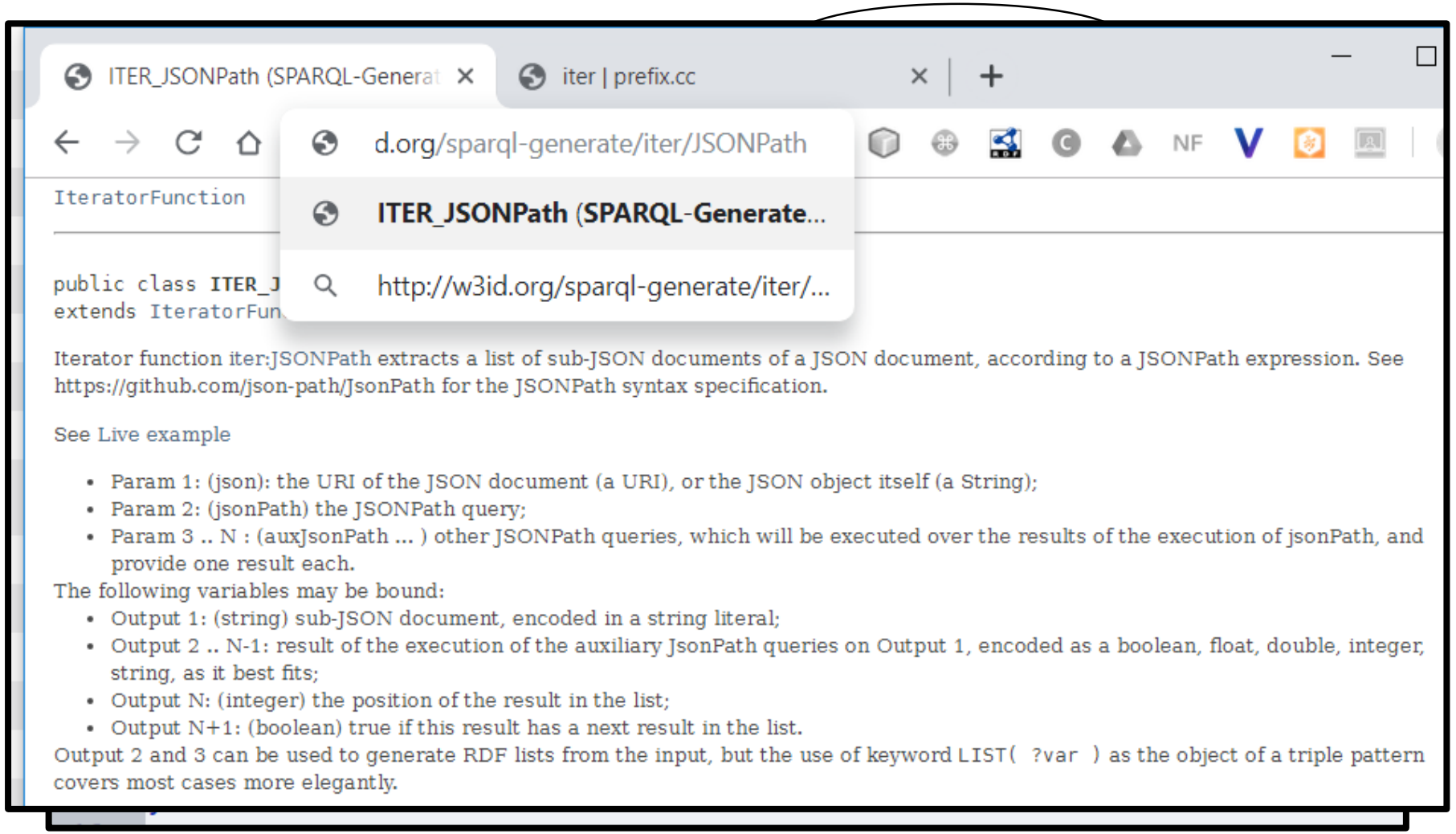
Binding Functions (that's standard SPARQL)






```
PREFIX fun:  <http://w3id.org/sparql-generate/fn/>
PREFIX iter: <http://w3id.org/sparql-generate/iter/>
GENERATE {
  [] a ex:Director ;
    foaf:name ?name ;
    ex:salary ?salary ;
    ex:fee ?fee .
}
SOURCE <myDocument> AS ?myDocument
ITERATOR iter:JSONPath( ?myDocument , "$.DirectorCompensation[*]") AS ?director
WHERE {
  BIND( fun:JSONPath( ?director , "$.DirectorName") AS ?name )
  BIND( fun:JSONPath( ?director , "$.Salary") AS ?salary )
  BIND( fun:JSONPath( ?director , "$.DirectorFee") AS ?fee )
}
```

iter&fun functions have dereferenceable URIs



It extends SPARQL, so we got for free …

- Implementable on top of existing SPARQL engines
- Easy to learn when you know SPARQL
- You get to know SPARQL better when you use SPARQL-Generate
- Output template (Basic Graph Pattern) **is not subject-centric**

- SPARQL 1.1 `FILTER BIND OPTIONAL`,…

- SPARQL 1.1 Standard binding functions and operators

```
STR LANG LANGMATCHES DATATYPE BOUND IRI URI BNODE RAND ABS CEIL
    FLOOR ROUND CONCAT SUBSTR  STRLEN REPLACE UCASE LCASE
ENCODE_FOR_URI CONTAINS STRSTARTS STRENDS STRBEFORE STRAFTER YEAR
 MONTH DAY HOURS MINUTES SECONDS TIMEZONE TZ NOW UUID STRUUID MD5
SHA1 SHA256 SHA384 SHA512 COALESCE IF STRLANG STRDT sameTerm isIRI
    isURI isBLANK isLITERAL isNUMERIC REGEX  + * / = > < || &&
```

- SPARQL 1.1 Binding functions extension mechanism

- SPARQL 1.1 `ORDER LIMIT OFFSET`

- SPARQL 1.1 `GROUP BY, HAVING`, Aggregate functions

```
COUNT SUM MIN MAX AVG SAMPLE GROUP_CONCAT
```

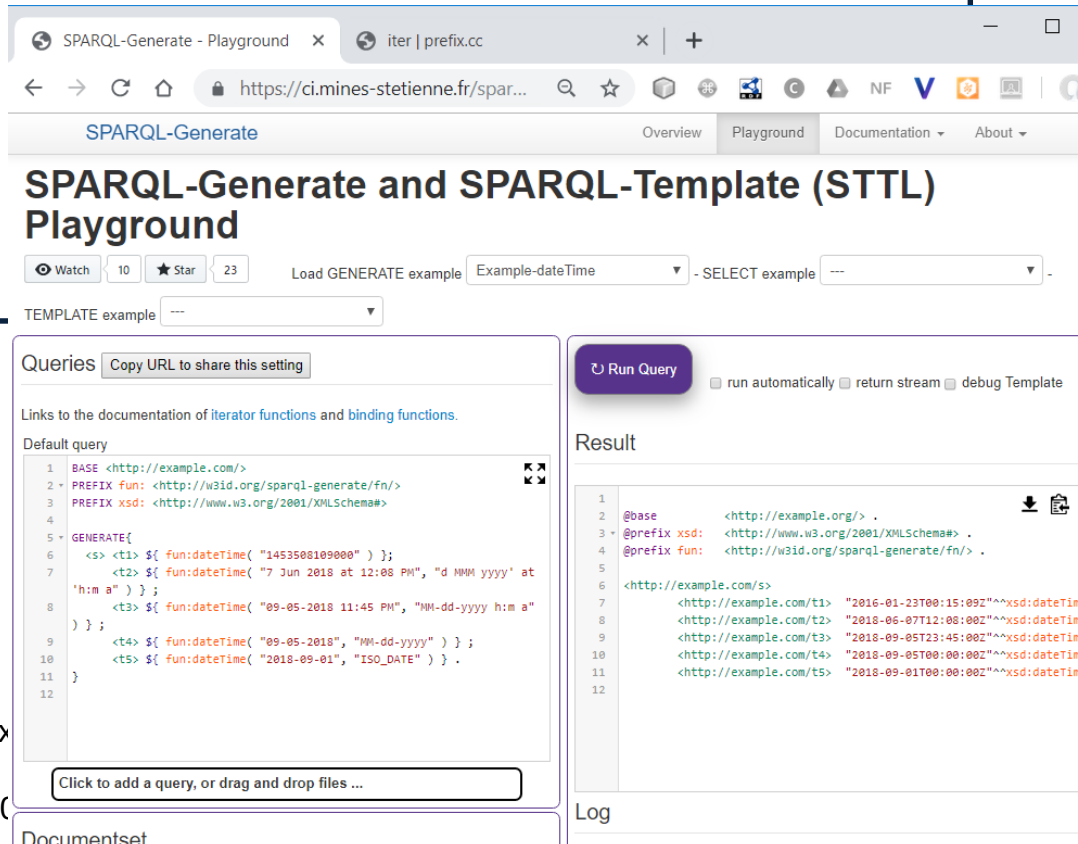
SPARQL-Generate is not just « theoretically cool »

https://w3id.org/sparql-generate/

Open-source Licence Apache 2.0 implementation based on Apache Jena financed by ITEA, ANR, ENGIE

Expressive, Perfomant, extensible to other formats

API (available on Maven Central)
JAR
Web playground + tutorial



M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL ex
In Proc. Extended Semantic Web Conference, 2017
+ EKAW, Hackatons, ESWC Tutorial 2018, PFIA Tutorial 20

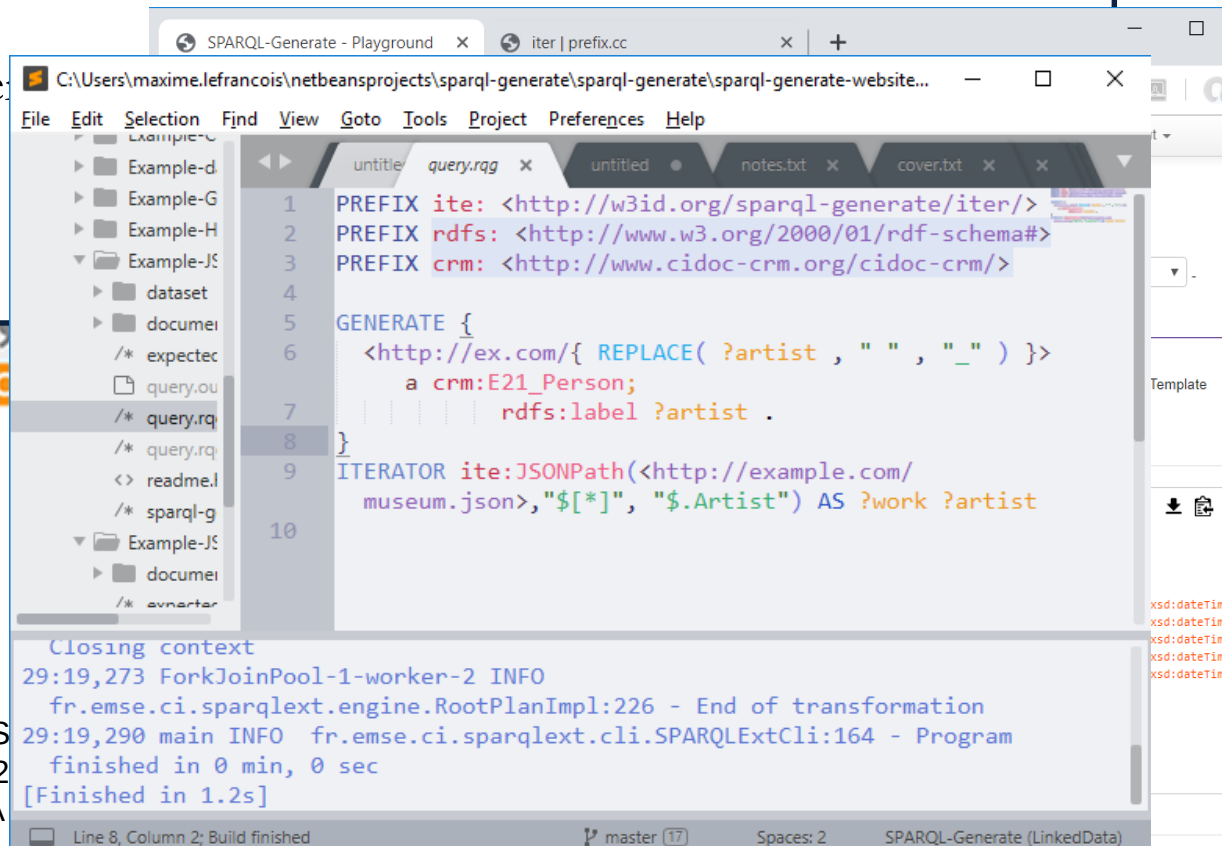
SPARQL-Generate is not just « theoretically cool »

https://w3id.org/sparql-generate/

Open-source Licence Apache 2.0 implementation based on Apache Jena financed by ITEA, ANR, ENGIE

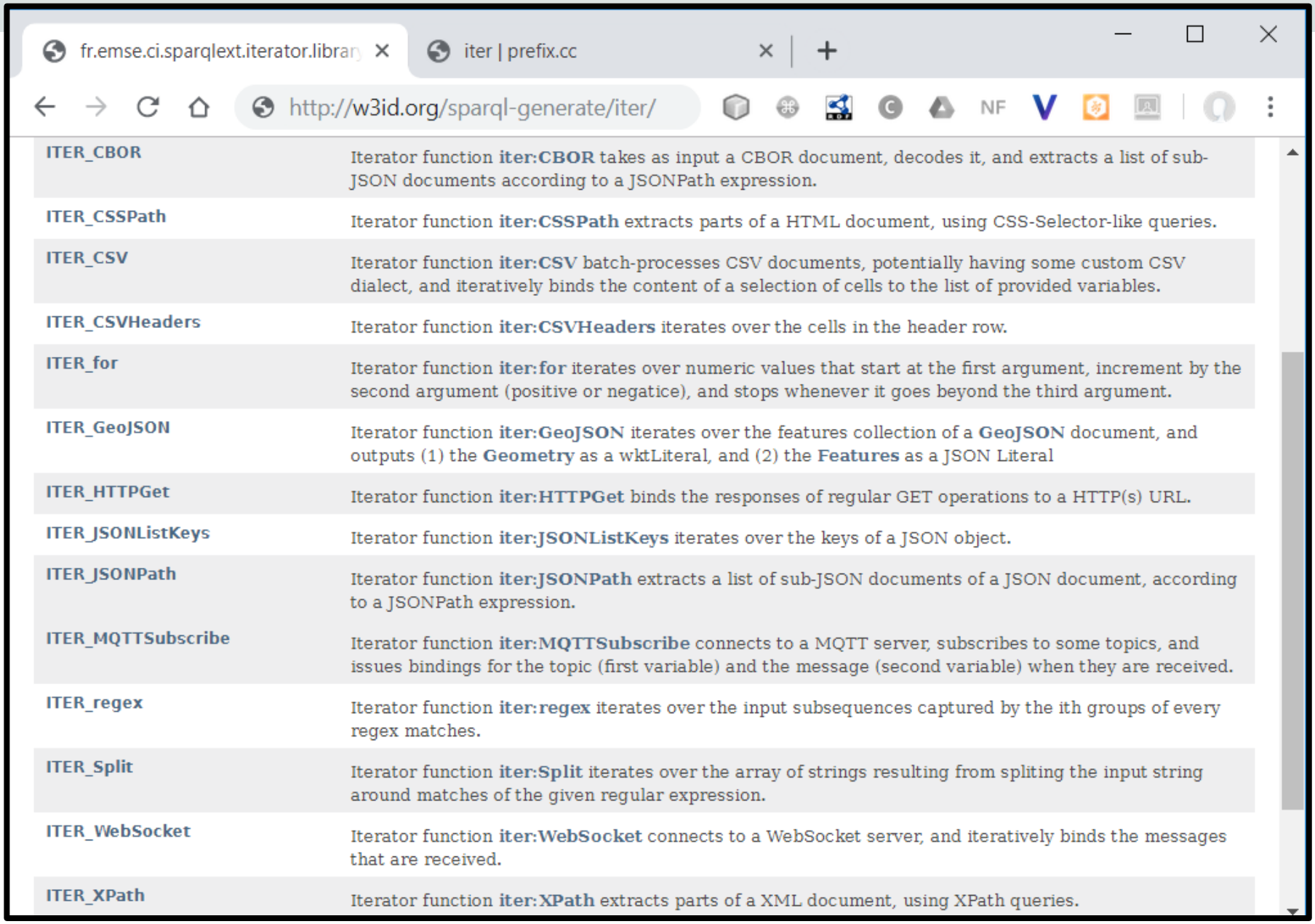
Expressive, Perfomant, extensible to other formats

API (available on Maven Ce
JAR
Web playground + tutorial
Sublime Text package



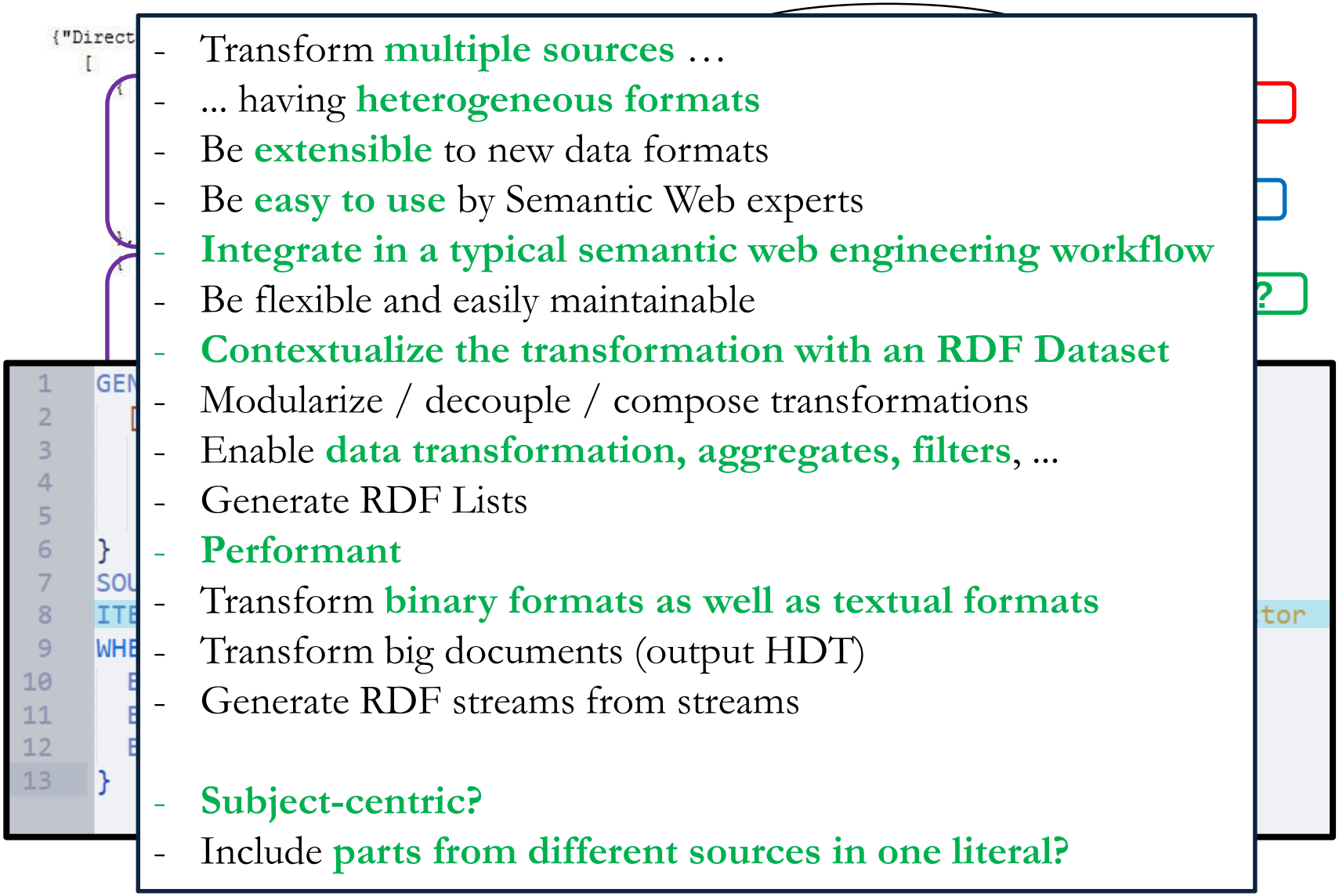
M. Lefrançois, A. Zimmermann, N. Bakerally, A S
In Proc. Extended Semantic Web Conference, 2
+ EKAW, Hackatons, ESWC Tutorial 2018, PFIA

Functions have dereferenceable URIs



ITER_CBOR	Iterator function **iter:CBOR** takes as input a CBOR document, decodes it, and extracts a list of sub-JSON documents according to a JSONPath expression.
ITER_CSSPath	Iterator function **iter:CSSPath** extracts parts of a HTML document, using CSS-Selector-like queries.
ITER_CSV	Iterator function **iter:CSV** batch-processes CSV documents, potentially having some custom CSV dialect, and iteratively binds the content of a selection of cells to the list of provided variables.
ITER_CSVHeaders	Iterator function **iter:CSVHeaders** iterates over the cells in the header row.
ITER_for	Iterator function **iter:for** iterates over numeric values that start at the first argument, increment by the second argument (positive or negatice), and stops whenever it goes beyond the third argument.
ITER_GeoJSON	Iterator function **iter:GeoJSON** iterates over the features collection of a **GeoJSON** document, and outputs (1) the **Geometry** as a wktLiteral, and (2) the **Features** as a JSON Literal
ITER_HTTPGet	Iterator function **iter:HTTPGet** binds the responses of regular GET operations to a HTTP(s) URL.
ITER_JSONListKeys	Iterator function **iter:JSONListKeys** iterates over the keys of a JSON object.
ITER_JSONPath	Iterator function **iter:JSONPath** extracts a list of sub-JSON documents of a JSON document, according to a JSONPath expression.
ITER_MQTTSubscribe	Iterator function **iter:MQTTSubscribe** connects to a MQTT server, subscribes to some topics, and issues bindings for the topic (first variable) and the message (second variable) when they are received.
ITER_regex	Iterator function **iter:regex** iterates over the input subsequences captured by the ith groups of every regex matches.
ITER_Split	Iterator function **iter:Split** iterates over the array of strings resulting from spliting the input string around matches of the given regular expression.
ITER_WebSocket	Iterator function **iter:WebSocket** connects to a WebSocket server, and iteratively binds the messages that are received.
ITER_XPath	Iterator function **iter:XPath** extracts parts of a XML document, using XPath queries.

How are we going so far?



- Transform **multiple sources** …
- … having **heterogeneous formats**
- Be **extensible** to new data formats
- Be **easy to use** by Semantic Web experts
- **Integrate in a typical semantic web engineering workflow**
- Be flexible and easily maintainable
- **Contextualize the transformation with an RDF Dataset**
- Modularize / decouple / compose transformations
- Enable **data transformation, aggregates, filters**, …
- Generate RDF Lists
- **Performant**
- Transform **binary formats as well as textual formats**
- Transform big documents (output HDT)
- Generate RDF streams from streams

- **Subject-centric?**
- Include **parts from different sources in one literal?**

What I didn't tell yet...

There's more than
SOURCE and ITERATOR ...

Iterators can bind several variables

```
#date;store;amount;consumer
#2019-05-01T00:50:09-07:00;95;352.33;Raja
#2019-05-01T23:52:06-07:00;69;354.95;Tyrone

GENERATE {
  ?storeIRI a ex:Store ;
    ex:sell [
      a ex:Purchase ;
      ex:customer [ a foal:Person ; foaf:name ?consumer ] ;
      ex:amount ?amountDecimal ;
      ex:date ?dateTime
    ] .
}
ITERATOR iter:CSV(<http://example.com/consumption.csv>,
    "date",       "store", "amount", "consumer" )
 AS ?dateTimeStr ?store   ?amount   ?consumer
WHERE {
  BIND(STRDT(?dateTimeStr, xsd:dateTime) AS ?dateTime)
  BIND(IRI(?store) AS ?storeIRI)
  BIND(xsd:decimal(?amount) as ?amountDecimal)
}
```

Iterators can be more powerful than that

```
#date;store;amount;consumer
#2019-05-01T00:50:09-07:00;95;352.33;Raja
#2019-05-01T23:52:06-07:00;69;354.95;Tyrone

GENERATE {
  ?storeIRI a ex:Store ;
    ex:sell [
      a ex:Purchase ;
      ex:customer [ a foal:Person ; foaf:name ?consumer ] ;
      ex:amount ?amountDecimal ;
      ex:date ?dateTime
    ] .
}
ITERATOR iter:CSV(<http://example.com/consumption.csv>, true , "\"", ";", "\n",
    "date",      "store", "amount", "consumer" )
 AS ?dateTimeStr ?store   ?amount   ?consumer
WHERE {
  BIND(STRDT(?dateTimeStr, xsd:dateTime) AS ?dateTime)
  BIND(IRI(?store) AS ?storeIRI)
  BIND(xsd:decimal(?amount) as ?amountDecimal)
}
```

Syntactic sugar helps to compact queries

```
#date;store;amount;consumer
#2019-05-01T00:50:09-07:00;95;352.33;Raja
#2019-05-01T23:52:06-07:00;69;354.95;Tyrone

GENERATE {
  <{?storeIRI}> a ex:Store ;
    ex:sell [
      a ex:Purchase ;
      ex:customer [ a foal:Person ; foaf:name ?consumer ] ;
      ex:amount ?{ xsd:decimal(?amount) } ;
      ex:date "{?dateTimeStr}"^^xsd:dateTime
    ] .
}
ITERATOR iter:CSV(<http://example.com/consumption.csv>, true , "\"", ";", "\n",
    "date",      "store", "amount", "consumer" )
 AS ?dateTimeStr ?store   ?amount   ?consumer
WHERE {}
```

Iterators work in batches

```
#date;store;amount;consumer
#2019-05-01T00:50:09-07:00;95;352.33;Raja
#2019-05-01T23:52:06-07:00;69;354.95;Tyrone

GENERATE {
  <{?storeIRI}> a ex:Store ;
    ex:sell [
      a ex:Purchase ;
      ex:customer [ a foal:Person ; foaf:name ?consumer ] ;
      ex:amount ?{ xsd:decimal(?amount) } ;
      ex:date "{?dateTimeStr}"^^xsd:dateTime
    ] .
}
ITERATOR iter:CSV(<http://example.com/consumption.csv>, 1000, true ,"\"",";","\n
    "date",      "store", "amount", "consumer" )
 AS ?dateTimeStr ?store   ?amount   ?consumer
WHERE {}
```

Iterators work in batches

```
7   GENERATE {
8       <events/{?eventId}> a ex:Event;
9           ex:type ?type;
10          ex:happenedAt ?datetime;
11  }
12  ITERATOR iter:WebSocket("wss://api.gemini.com/v1/marketdata/BTCUSD") AS ?events
13  WHERE {
14          BIND(fun:JSONPath(?events, "$.type" ) AS ?type)
15          BIND(xsd:string(fun:JSONPath(?events, "$.eventId" )) AS ?eventId)
16          BIND(fun:dateTime(xsd:string(fun:JSONPath(?events, "$.timestampms" ))) AS ?
              datetime)
17  }
```

Generate clauses can be nested

```
GENERATE {
  <{?idx}/observations/{?t}#aqi> a aqio:AirQualityIndexObservation;
    sosa:resultTime "{?dateTime}{?tz}"^^xsd:dateTime;
    sosa:hasSimpleResult ?aqi.

  GENERATE {
    <pollution/observations#aqi-avg> a seas:MinimumEvaluation ;
      seas:hasSimpleValue ?{ avg(?aqi) } ;
      seas:evaluationOf <pollution#aqi>  .
  } .
}
ITERATOR iter:for(0,1,100) AS ?index
SOURCE <https://ci.mines-stetienne.fr/aqi/static/station/{STR(?index)}> AS ?source
WHERE {
  BIND(STR(xsd:integer(fun:JSONPath(?source,"$.data.idx"))) AS ?idx)
  BIND(STR(xsd:integer(fun:JSONPath(?source,"$.data.time.v"))) AS ?t)
  BIND(REPLACE(fun:JSONPath(?source,"$.data.time.s")," ","T") AS ?dateTime)
  BIND(fun:JSONPath(?source,"$.data.time.tz") AS ?tz)
  BIND(xsd:integer(fun:JSONPath(?source,"$.data.aqi")) AS ?aqi)
}
```

Generate queries can be modularized

```
GENERATE {

<loc/{?idx}> a sosa:FeatureOfInterest;
  rdfs:label ?name;
  geo:lat ?lat;
  geo:long ?long ;
  ex:pollution ?aqi .

  GENERATE <findEvents>( <http://example.org/loc/{?idx}>, ?lat, ?long ) .

}
ITE
SOU
WHE

} O
LIM
```

```
GENERATE <findEvents>(?near, ?lat, ?long) {
  <event/{?eid}> a event:Event;
    foaf:basedNear ?near ;
    dc:date ?date ;
    rdfs:label ?title;
    rdfs:comment "{?desc}"^^rdf:HTML .
}
ITERATOR iter:XPath(<http://api.eventful.com/rest/events/
  search?app_key=9p3bTRHL2NTzVgxG&location={STR(?lat)},{STR(?long)}&within=20>,"/
  search/events/event", "/event/@id", "/event/title/text()", "/event/description/
  text()", "/event/start_time/text()") AS ?event ?eid ?title ?desc ?eventDate
BIND("{ REPLACE( ?eventDate, " ", "T") }"^^xsd:dateTime AS  ?date)
WHERE {}
ORDER BY ?eventDate
LIMIT 3
```

One can generate RDF lists easily*

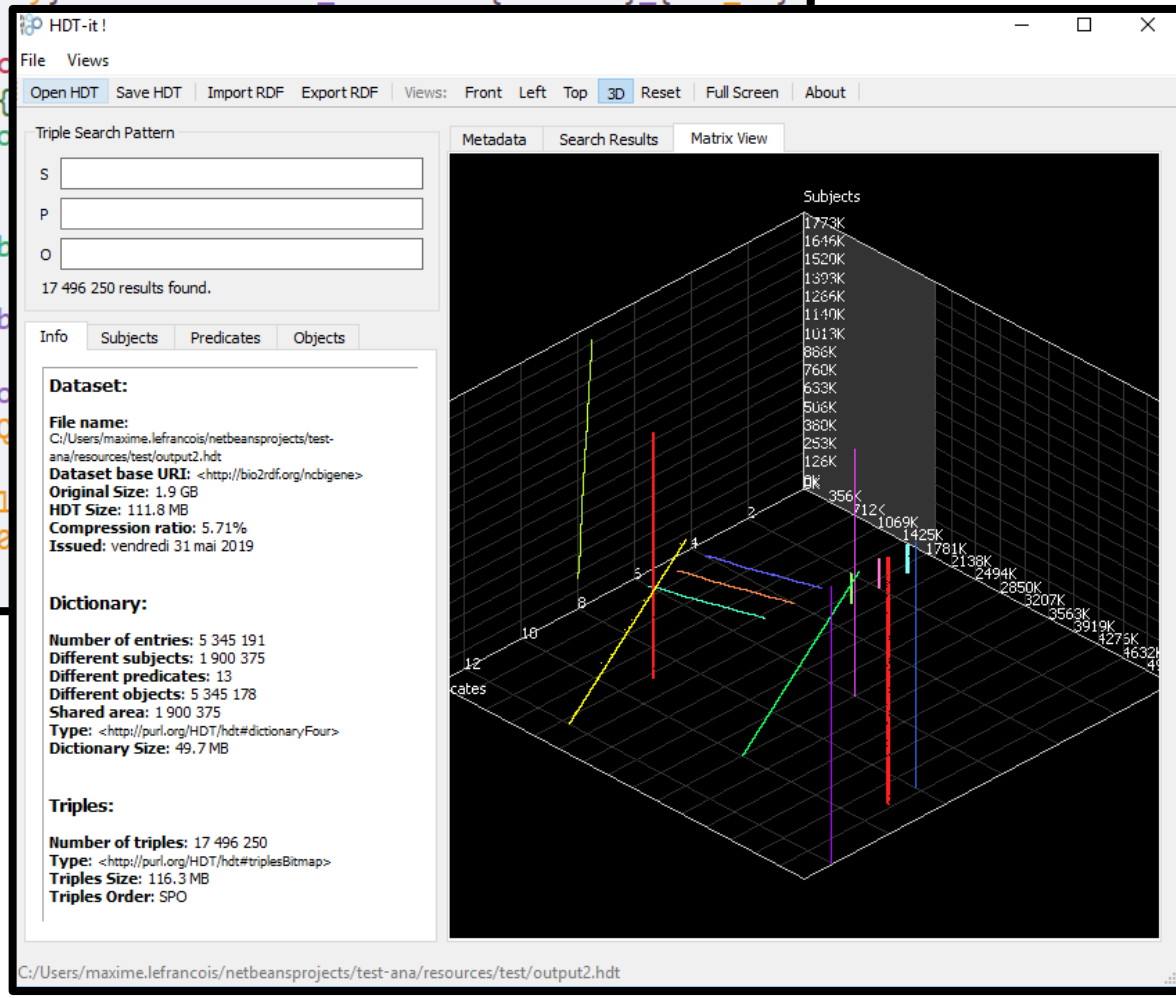
* NEW!

```
GENERATE {
<loc/{?idx}> a sosa:FeatureOfInterest;
    rdfs:label ?name .

  ?observation a aqio:AirQualityIndexObservation;
    sosa:hasSimpleResult ?aqi ;
    sosa:hasFeatureOfInterest <loc/{?idx}>.

  <loc/world> ex:observationList LIST( ?observation ) .

}
ITERATOR iter:for(0,1,9) AS ?index
SOURCE <https://ci.mines-stetienne.fr/aqi/static/station/{STR(?index)}> AS ?source
WHERE {
  BIND(STR(xsd:integer(fun:JSONPath(?source,"$.data.idx"))) AS ?idx)
  BIND(fun:JSONPath(?source,"$.data.city.name") AS ?name)
  BIND(xsd:integer(fun:JSONPath(?source,"$.data.aqi")) AS ?aqi)
  BIND( BNODE()) AS ?observation )
  FILTER( BOUND( ?aqi ) )
} ORDER BY DESC (?aqi)
```

One can generate HDT directly*

* NEW!

```
GENERATE {
  <:{ ?GeneID }> rdf:type ncbiv:Resource ;
     <_vocabulary:{ ?Category }> ?GO_ID ;
     <_vocabulary:ncbiv:gene-{?Category}-association> <_resource:{?GeneID}_{?GO_ID}>
        .
  <_resource:{?GeneID}_{?GO_ID}> a nc
    rdfs:label "association between {
    dcterms:identifier "ncbigene_resc
    ncbiv:evidence "eco:{?Evidence}"
    ncbiv:gene  <:{ ?GeneID }> ;
    ncbiv:go-category "ncbigene_vocab
    ncbiv:go-term ?GO_ID ;
    ncbiv:go-term <http://purl.obolib
}
ITERATOR iter:CSV( <http://example.co
  tax_id0 ?GeneID ?GO_ID ?Evidence ?Q
WHERE {
  BIND(IF(?Qualifier0="-", ?sd, ?Qual
  BIND(IF(?PubMed0="-", ?sd, ?PubMed0
}
```



20 seconds for a 20 MB gene2go sample, output is 17 MB of HDT

9 min, 20 sec for a 145 MB gene2go sample, output is "only" 114 MB HDT

What I didn't tell yet...

There's more than GENERATE ...

FUNCTION

Functions on RDF terms, RDF graphs or SPARQL results
Subset of LDScript: a Linked Data Script Language
http://ns.inria.fr/sparql-extension/

```
FUNCTION <default>( ?value , ?default ) {
  IF( BOUND( ?value ), ?value, ?default )
}
```

```
FUNCTION <printLabel> ( ?uri , ?value ) {
  IF( ISURI(?uri),
    "\n<a href='{?uri}'>{ <default>( ?value, ?uri) }</a>",
    IF( ISLITERAL(?uri) ,
      "<span>{?uri}</span>",
      "<span>{?value}</span>"
    )
  )
}
```

```
FUNCTION <factorial>( ?x ) {
  IF( ?x=1, 1, ?x * <factorial>(?x-1) )
}
```

Olivier Corby, Catherine Faron-Zucker and Fabien Gandon, LDScript: a Linked Data Script Language, ISWC 2017, spotlight paper

TEMPLATE

Generate Text from RDF

Extended subset of STTL: the SPARQL Template Transformation Language

https://ns.inria.fr/sparql-template/

```
4  TEMPLATE <printFoafName> ( ?dt , ?uri , ?sep ) {
5    before = "\n<dt>{ ?dt }:</dt><dd>" ;
6
7    <printLabel>(?uri, ?name)
8
9    ; separator = ?sep
10   ; after = "</dd>"
11
12 } WHERE {
13   OPTIONAL {
14     ?uri foaf:name ?name
15   }
16 } ORDER BY ?uri
```

- Elements `FORMAT GROUP BOX`
- Binding function `st:call-template(templateURI, param1, param2, ...)`

Olivier Corby and Catherine Faron-Zucker. A Transformation Language for RDF based on SPARQL. Web Information Systems and Technologies - Selected Extended Papers from WEBIST 2015. Springer-Verlag, 2015. Best paper nominee.

TEMPLATE

Generate Text from RDF
Extended subset of STTL: the SPARQL Template Transformation Language
https://ns.inria.fr/sparql-template/

```
TEMPLATE <printFoafName> ( ?uri ) {

  fun:JSONPath(?document, "$.name")  "has child" <printLabel>(?uri, ?name)

}
SOURCE ?uri AS ?document
ITERATOR fun:JSONPath(?document, "$.children[*]") AS ?child
WHERE {

} ORDER BY ?name
```

- Elements `FORMAT GROUP BOX` also SPARQL-Generate clauses `ITERATOR SOURCE`
- Binding function `st:call-template(templateURI, param1, param2, ...)`

Olivier Corby and Catherine Faron-Zucker. A Transformation Language for RDF based on SPARQL. Web Information Systems and Technologies - Selected Extended Papers from WEBIST 2015. Springer-Verlag, 2015. Best paper nominee.

SELECT

SPARQL Select:
- with SPARQL-Generate clauses `ITERATOR SOURCE`
- with syntactic sugar, etc.

```
SELECT <aqiStats>(?start, ?stop) (min(?aqi) AS ?avg) (max(?aqi) AS ?avg)
ITERATOR iter:for(?start, 1, ?stop) AS ?index
SOURCE <https://ci.mines-stetienne.fr/aqi/static/station/{STR(?index)}>
                    AS ?source
WHERE {
  BIND(xsd:decimal(fun:JSONPath(?source,"$.data.aqi")) AS ?aqi)
  FILTER(BOUND(?aqi))
}
```

SELECT

SPARQL Select:
- with SPARQL-Generate clauses `ITERATOR SOURCE`
- with syntactic sugar, etc.

```
SELECT <aqiStats>(?start, ?stop) (min(?aqi) AS ?avg) (max(?aqi) AS ?avg)
ITERATOR iter:for(?start, 1, ?stop) AS ?index
SOURCE <https://ci.mines-stetienne.fr/aqi/static/station/{STR(?index)}>
                AS ?source
WHERE {
  BIND(xsd:decimal(fun:JSONPath(?source,"$.data.aqi")) AS ?aqi)
  FILTER(BOUND(?aqi))
}
```

- SELECT queries output a list of solution bindings …
- … Like `ITERATOR` clauses

```
GENERATE {   }
ITERATOR <aqiStats>(0,1000) AS ?min ?max
WHERE {    }
```

```
TEMPLATE {   }
ITERATOR <aqiStats>(0,1000) AS ?min ?max
WHERE {    }
```

Use ITERATORS in sequence?

Let's give you an idea of how an unoptimized query looks like

```
GENERATE {

  <http://udaptor-apple.com/tracks/{?trackId}> rdf:type apple:Song;
     appleProp:writtenBy <http://udaptor-apple.com/artist/{?artistName}> ;
     appleProp:inAlbum <http://udaptor-apple.com/artist/{?albumName}> .

  <http://udaptor-apple.com/artist/{?albumName}> rdf:type apple:Album;
              appleProp:albumName ?albumName.

  <http://udaptor-apple.com/artist/{?artistName}> rdf:type apple:Artist ;
              appleProp:artistName ?artistName.

  <http://udaptor-apple.com/playlist/{?playlistId}> rdf:type apple:Playlist ;

              appleProp:playlistType ?playlistType.

  <http://udaptor-apple.com/tracks/{?playlistTrackIdStr}>
      appleProp:inPlaylist <http://udaptor-apple.com/playlist/{?playlistId}>.

}

SOURCE </Users/ankushsharma/Desktop/code/spark-generate-example/resources/playlists.json> AS ?playlistFile
ITERATOR iter:JSONPath( ?playlistFile , "$[*]") as ?playlistData
ITERATOR iter:JSONPath( ?playlistData, "$.['Playlist Item Identifiers'][*]" ) as ?playlistTrackId

SOURCE </Users/ankushsharma/Desktop/code/spark-generate-example/resources/tracks.json> AS ?tracksFile
ITERATOR iter:JSONPath( ?tracksFile , "$[*]", "$.Title", "$.Artist", "$.Album", "$.['Track Identifier']") AS ?songsKey ?
  songTitle ?artistName ?albumName ?songId

WHERE {
  BIND(STR(?songId) as ?trackId)
  BIND(STR(?playlistTrackId) as ?playlistTrackIdStr)
  BIND(STR(fun:JSONPath(?playlistData, "$.['Container Identifier']")) as ?playlistId)
  BIND(fun:JSONPath(?playlistData, "$.Title") as ?playlistTitle)
  BIND(fun:JSONPath(?playlistData, "$.['Container Type']") as ?playlistType)
}
```

Use ITERATORS in sequence?

Let's give you an idea of how an unoptimized query looks like

- GENERATE queries output RDF Graph …
- …

```
GENERATE {

  <http://udaptor-apple.com/artist/{?artistName}> rdf:type apple:Artist ;
        appleProp:albumName ?albumName.

  <http://udaptor-apple.com/artist/{?artistName}> rdf:type apple:Artist ;
        appleProp:artistName ?artistName.

  <http://udaptor-apple.com/playlist/{?playlistId}> rdf:type apple:Playlist ;

        appleProp:playlistType ?playlistType.

  <http://udaptor-apple.com/tracks/{?playlistTrackIdStr}>
     appleProp:inPlaylist <http://udaptor-apple.com/playlist/{?playlistId}>.

}

SOURCE </Users/ankushsharma/Desktop/code/spark-generate-example/resources/playlists.json> AS ?playlistFile
ITERATOR iter:JSONPath( ?playlistFile , "$[*]") as ?playlistData
ITERATOR iter:JSONPath( ?playlistData, "$.['Playlist Item Identifiers'][*]" ) as ?playlistTrackId

SOURCE </Users/ankushsharma/Desktop/code/spark-generate-example/resources/tracks.json> AS ?tracksFile
ITERATOR iter:JSONPath( ?tracksFile , "$[*]", "$.Title", "$.Artist", "$.Album", "$.['Track Identifier']") AS ?songsKey ?
  songTitle ?artistName ?albumName ?songId

WHERE {
  BIND(STR(?songId) as ?trackId)
  BIND(STR(?playlistTrackId) as ?playlistTrackIdStr)
  BIND(STR(fun:JSONPath(?playlistData, "$.['Container Identifier']")) as ?playlistId)
  BIND(fun:JSONPath(?playlistData, "$.Title") as ?playlistTitle)
  BIND(fun:JSONPath(?playlistData, "$.['Container Type']") as ?playlistType)
}
```

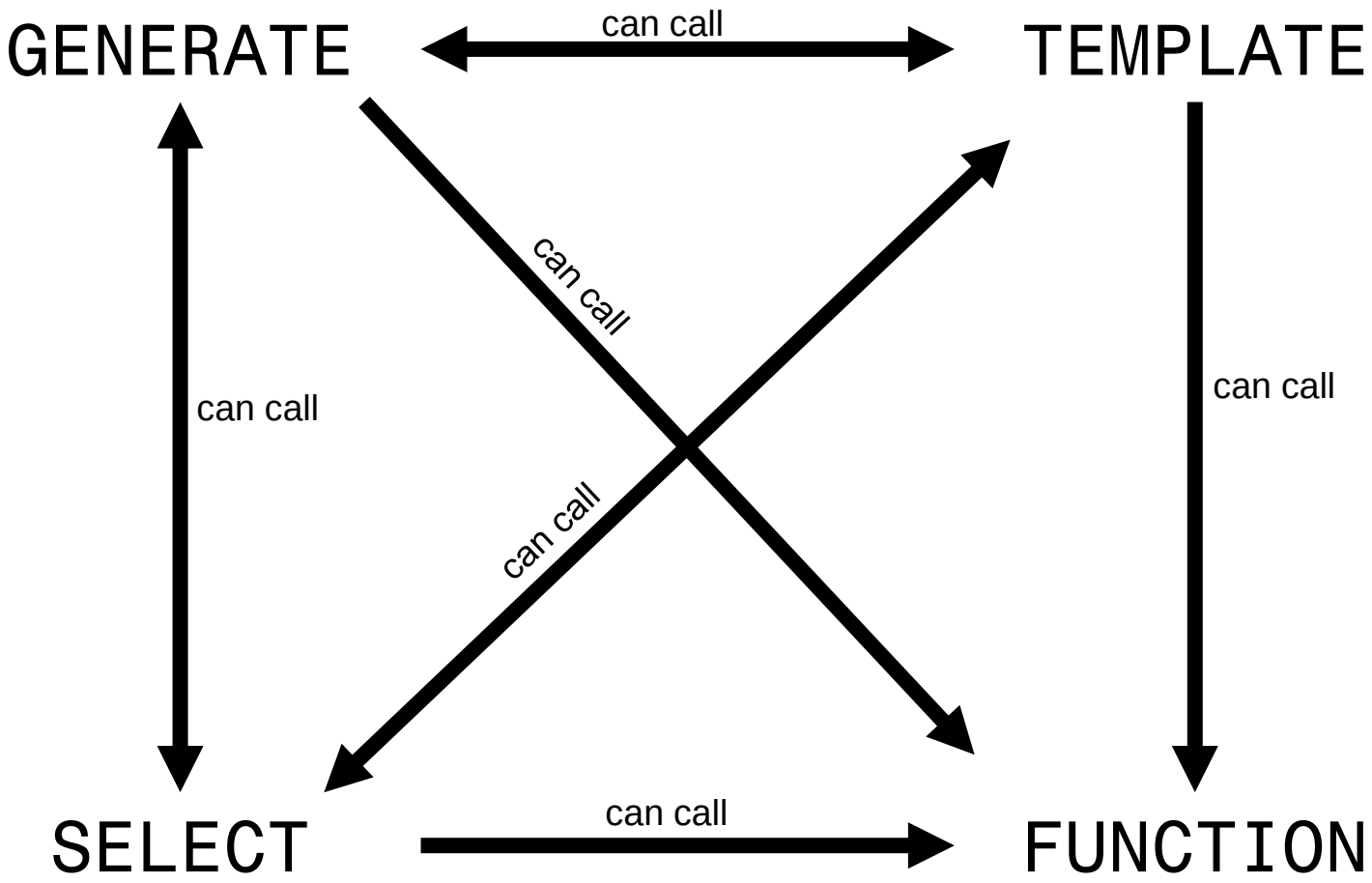
Calling a GENERATE query

`FROM GENERATE {}` and `FROM GENERATE <>(param1, param2, …)`

- GENERATE queries output RDF Graph …
- … like FROM Clauses

```
GENERATE {
  <http://udaptor-apple.com/tracks/{str(?songId)}> rdf:type apple:Song;
     appleProp:writtenBy <http://udaptor-apple.com/artist/{?artistName}> ;
     appleProp:inAlbum <http://udaptor-apple.com/artist/{?albumName}> .
  <http://udaptor-apple.com/artist/{?albumName}> rdf:type apple:Album;
     appleProp:albumName ?albumName.
  <http://udaptor-apple.com/artist/{?artistName}> rdf:type apple:Artist ;
     appleProp:artistName ?artistName.
}
FROM GENERATE {
      <http://udaptor-apple.com/playlist/{str(?playlistId)}> rdf:type apple:Playlist ;
         appleProp:playlistType ?playlistType.
      <http://udaptor-apple.com/tracks/{str(?playlistTrackId)}> rdf:type apple:Song ;
         appleProp:inPlaylist <http://udaptor-apple.com/playlist/{str(?playlistId)}>.
    }
    SOURCE <http://example.org/resources/playlists.json> AS ?playlistFile
    ITERATOR iter:JSONPath( ?playlistFile , "$[*]", "$.['Container Identifier']", "$.Title", "$.['Container Type']") as ?
      playlistData ?playlistId ?playlistTitle ?playlistType
    ITERATOR iter:JSONPath( ?playlistData, "$.['Playlist Item Identifiers'][*]" ) as ?playlistTrackId .

SOURCE <http://example.org/resources/tracks.json> AS ?tracksFile
ITERATOR iter:JSONPath( ?tracksFile , "$[*]", "$.Title", "$.Artist", "$.Album", "$.['Track Identifier']") AS ?songsKey ?
  songTitle ?artistName ?albumName ?songId
WHERE {
  ?trackId rdf:type apple:Song;
  FILTER( <http://udaptor-apple.com/tracks/{str(?songId)}> = ?trackId )
}
```

Modularize, Decouple, Compose queries



What I didn't tell yet...

There's much more to do than « just use it »

There's much more to do than « just use it »

Two use cases that drove recent development

- Generation of documentation from an ontology
- Generation of an ontology from configuration (RDF or files)

Use it

- On the web, as JAR, in Sublime Text, as API
- Extend with your own functions and iterators (e.g., NER)
- Make students play with it
- Ask for help, contribute, report issues, propose enhancements
  https://github.com/sparql-generate/sparql-generate/issues

There's much more to do than « just use it »

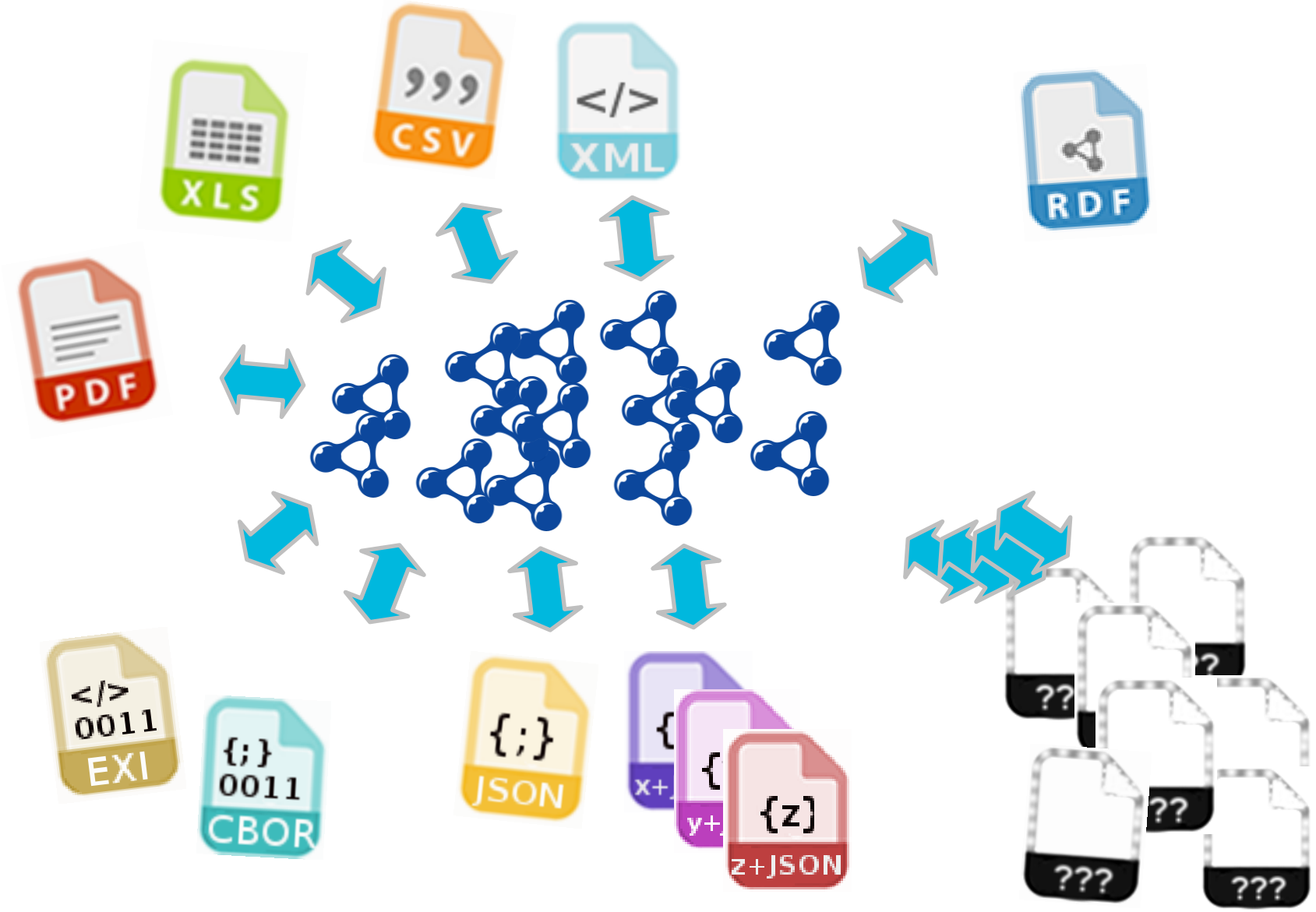
Ideas

- How can SPARQL-Generate be compared to / used with RML?
- How can SPARQL-Generate be used by Helio?
- What query optimization techniques for SPARQL-Generate?
- Can we use (a subset of) SPARQL-Generate for OBDA?
- Can we qualify transformations w.r.t the input/output space?
- Can we identify/compute operators on these objects?
  e.g., If input conforms to a schema, then does the output conforms to a SHACL shape?

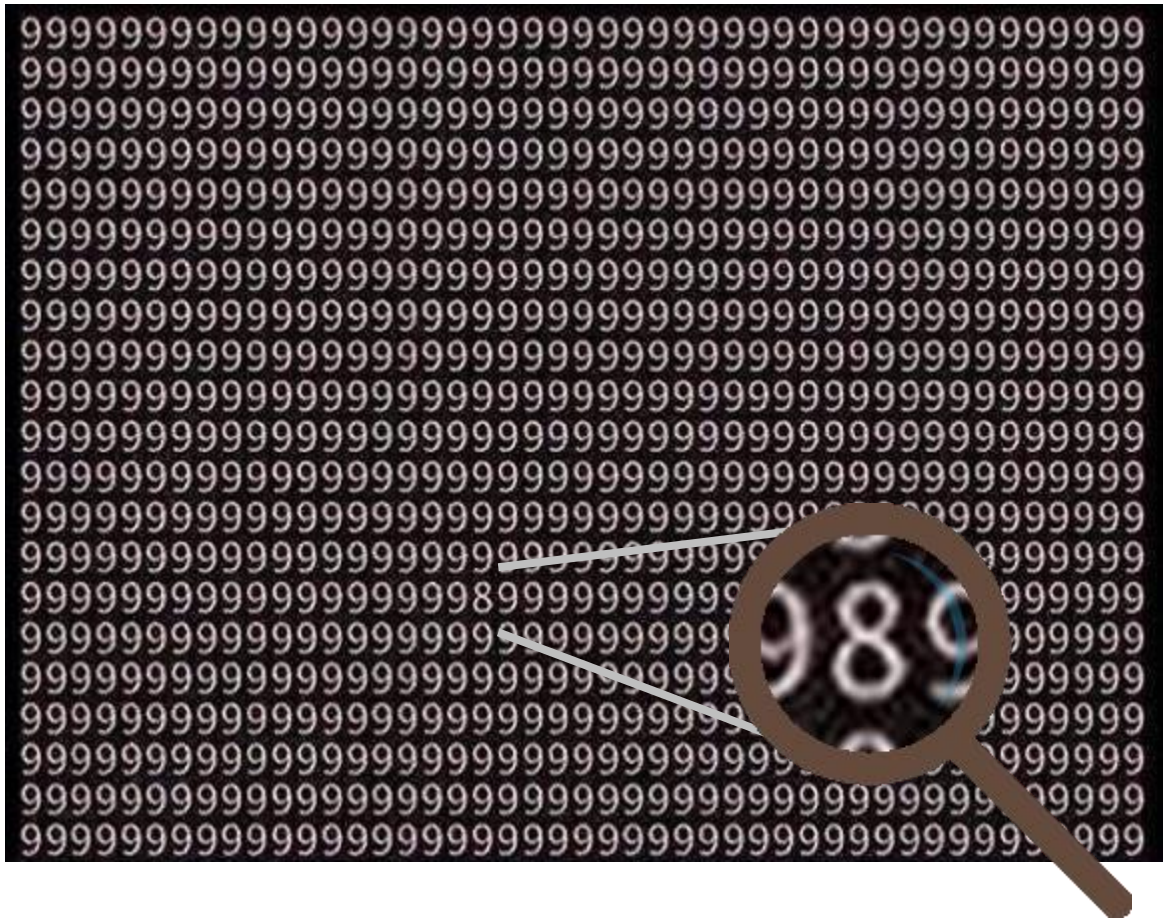
What I didn't tell yet...

What was the main purpose ...

Can we use RDF as a lingua franca?



Idea: just send RDF!



application/rdf+xml
text/turtle
application/ld+json
…
ERI
HDTQ

RDF data *formats* will never be the only ones on the web
Developers prefer JSON, …

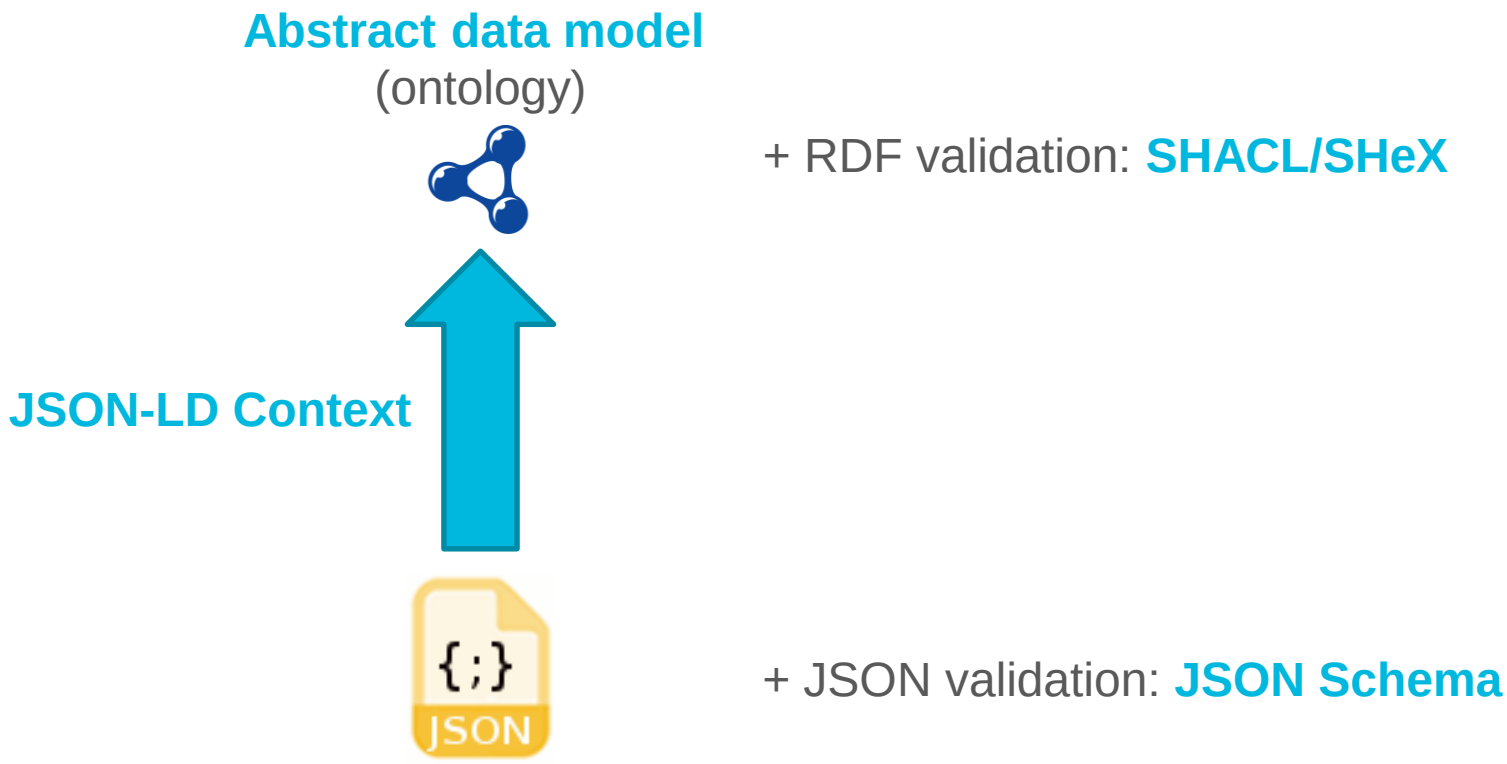
Idea: just send ~~RDF!~~ JSON-LD!

Solution:

- to rely on *one* RDF syntax like JSON-LD

Requires:

- Global adoption of JSON-LD Utopian
- Maintaining during the development or the evolution phase



Approx. modeling of the communication between heterogeneous agents on the Web.

sender

wants to send some content

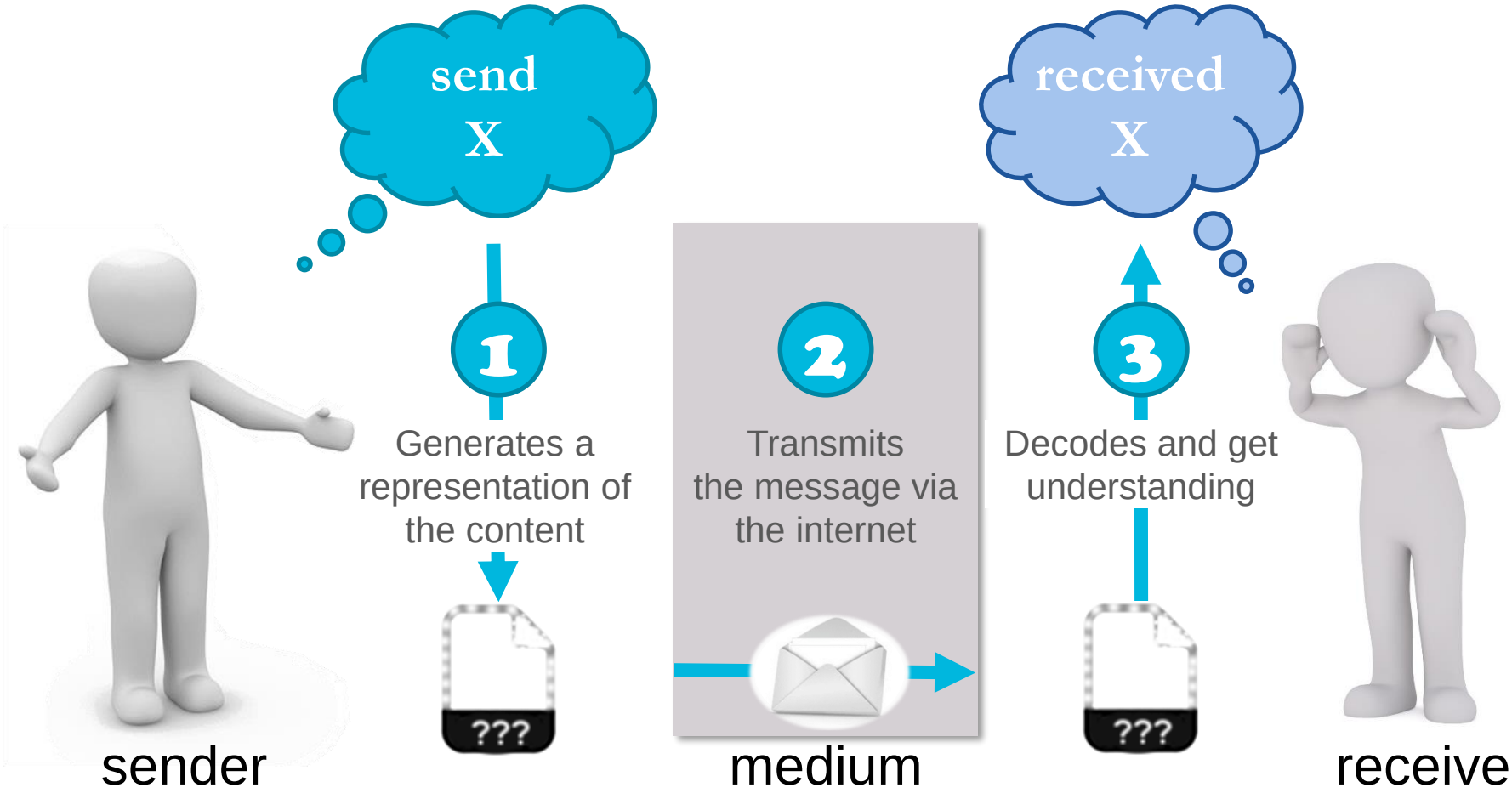
e.g., state of a resource

medium

receiver

M. Lefrançois, RDF presentation and correct content conveyance for legacy services and the web of things. In Proc. IOT'18

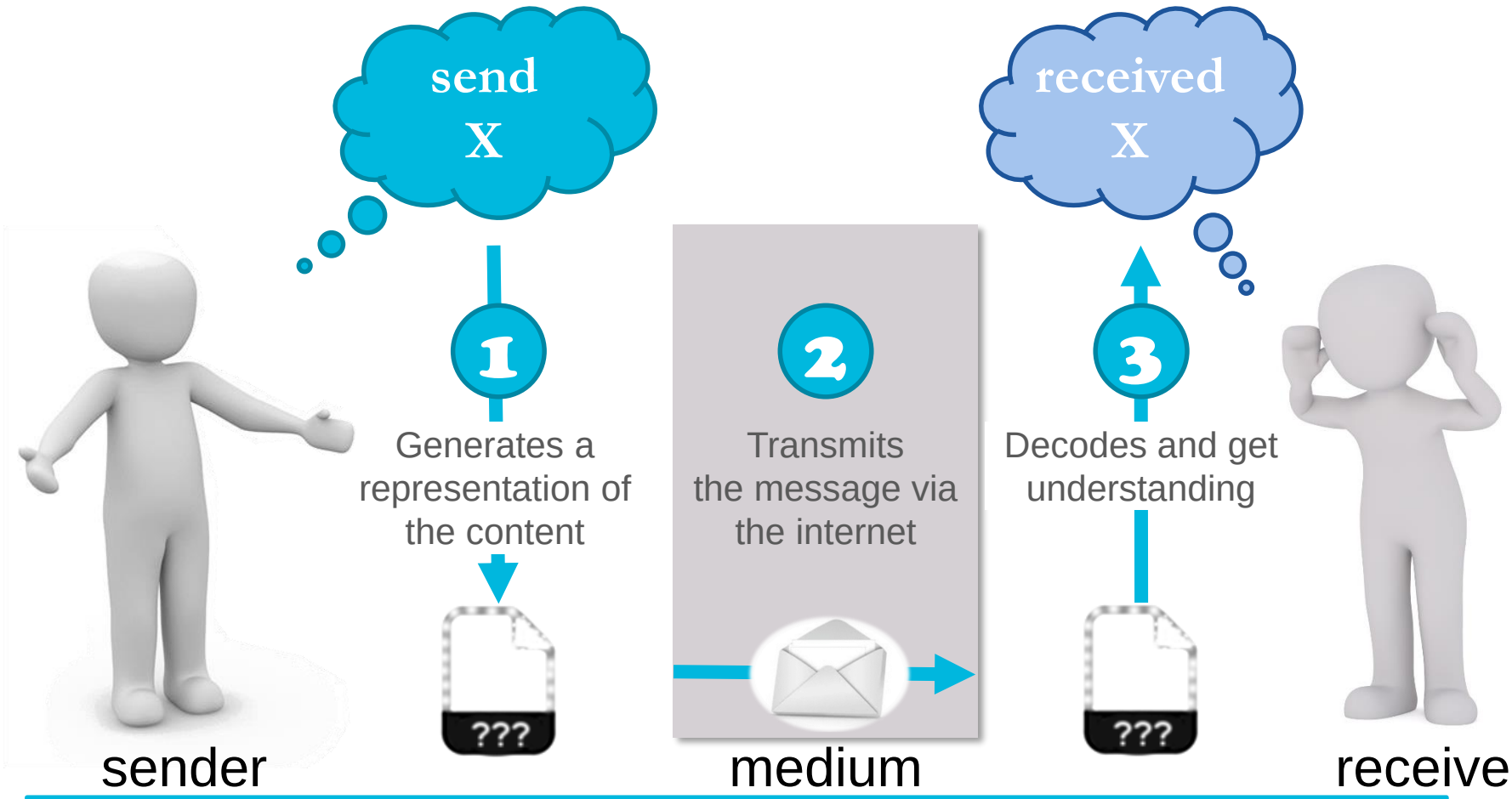
Approx. modeling of the communication between heterogeneous agents on the Web.



wants to send some content
e.g., state of a resource

M. Lefrançois, RDF presentation and correct content conveyance for legacy services and the web of things. In Proc. IOT'18

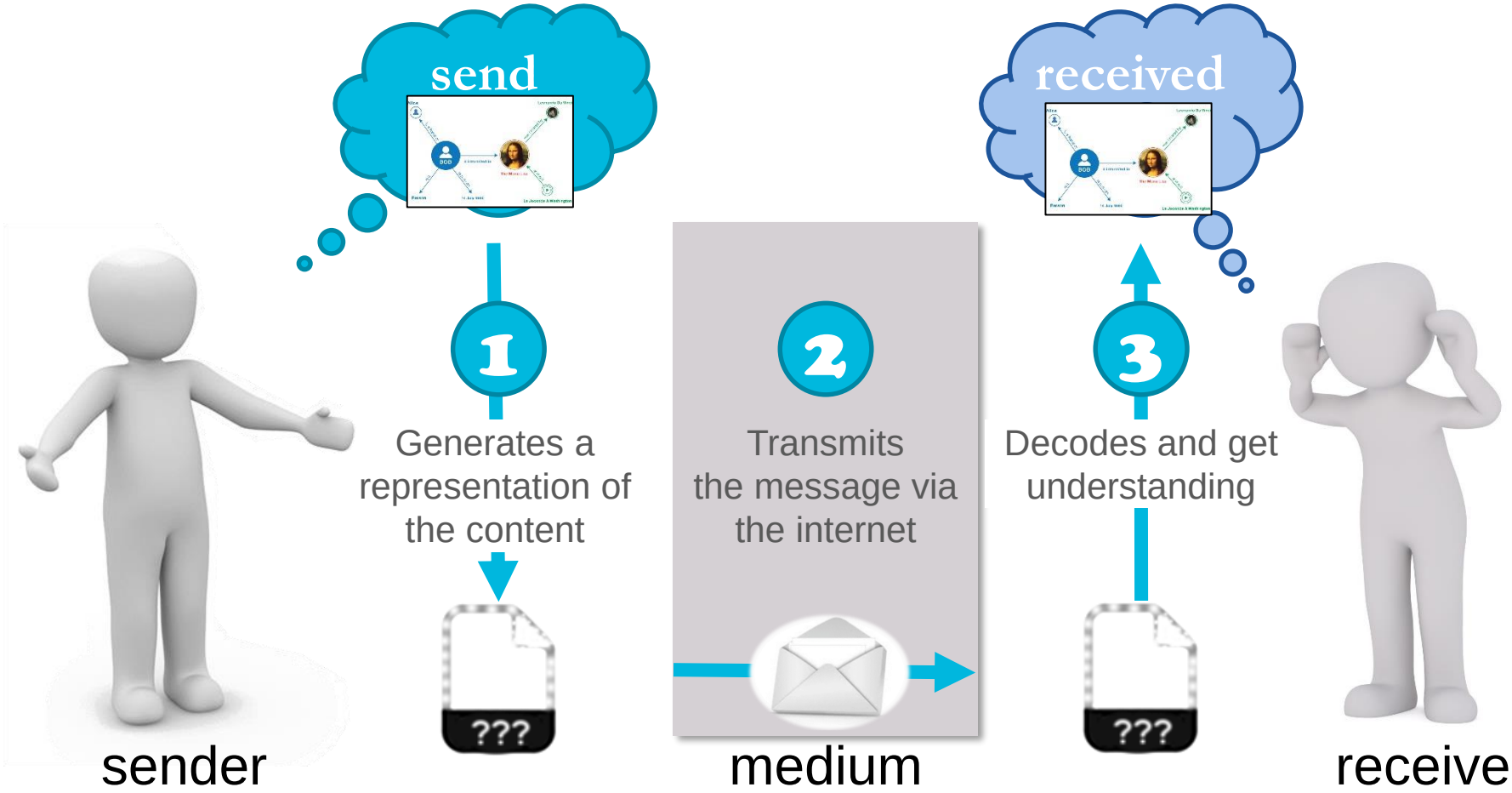
Approx. modeling of the communication between heterogeneous agents on the Web.



Correct Content Conveyance **iif:**

1. all of the essential characteristics of the content is encoded in the message
2. the encoding and the decoding phase are symmetric
3. the message is not altered in the transmission medium

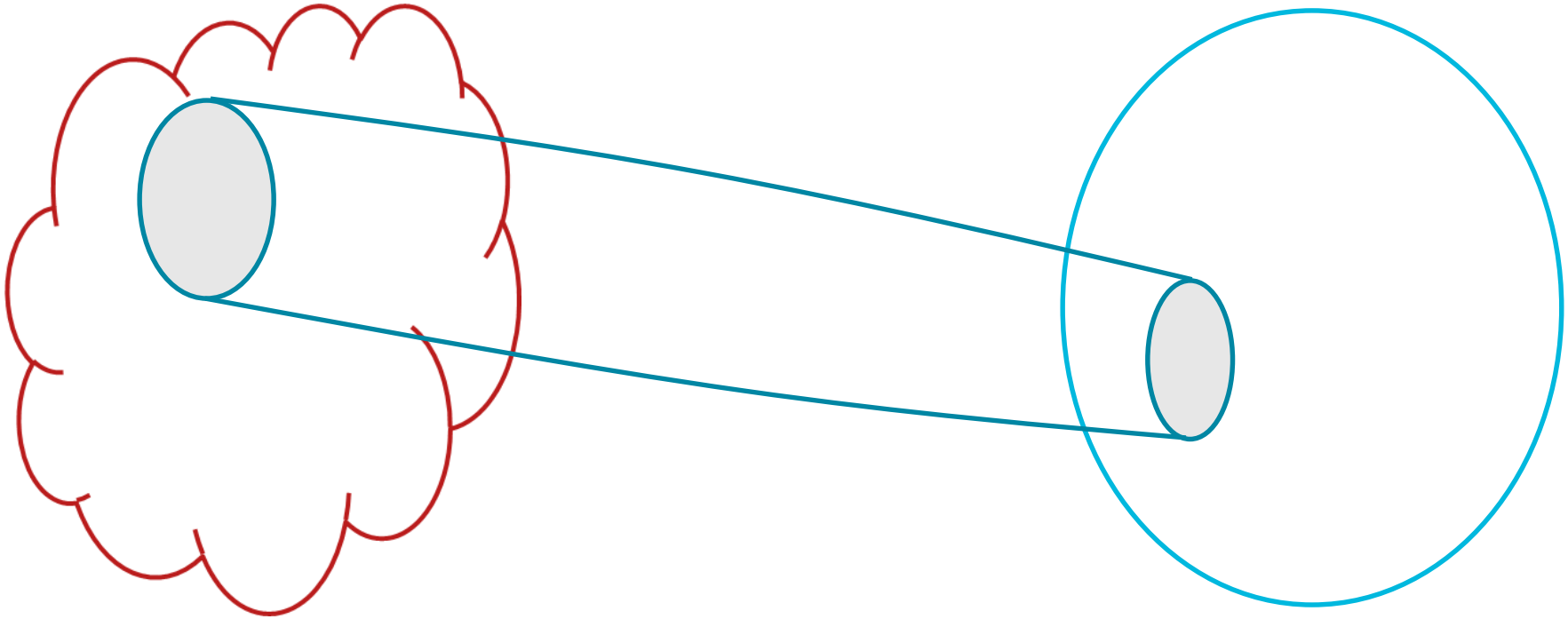
Assumption:
content is always a RDF graph



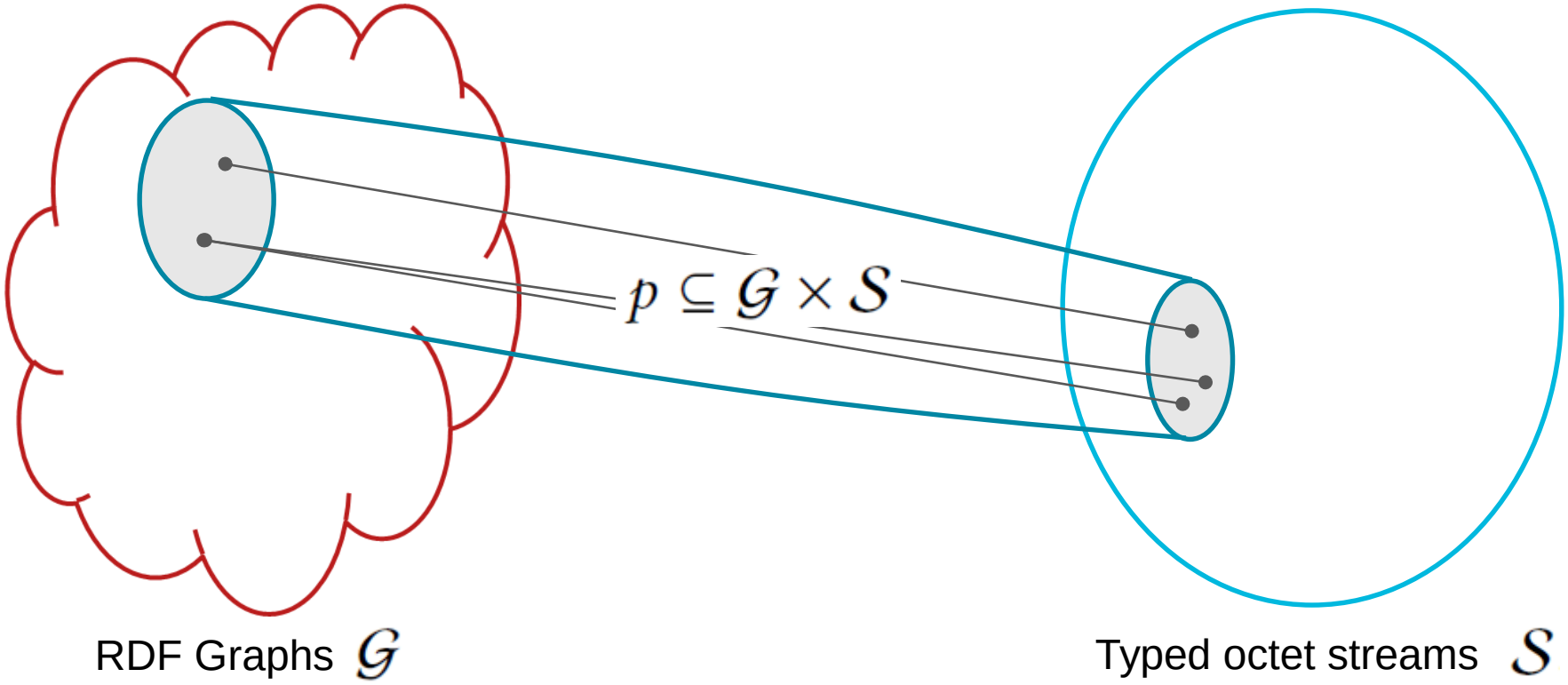
Correct Content Conveyance **iif:**

The RDF graph the sender encodes is equivalent to the RDF graph the receiver obtained after decoding the message

Content Representation



RDF Presentations

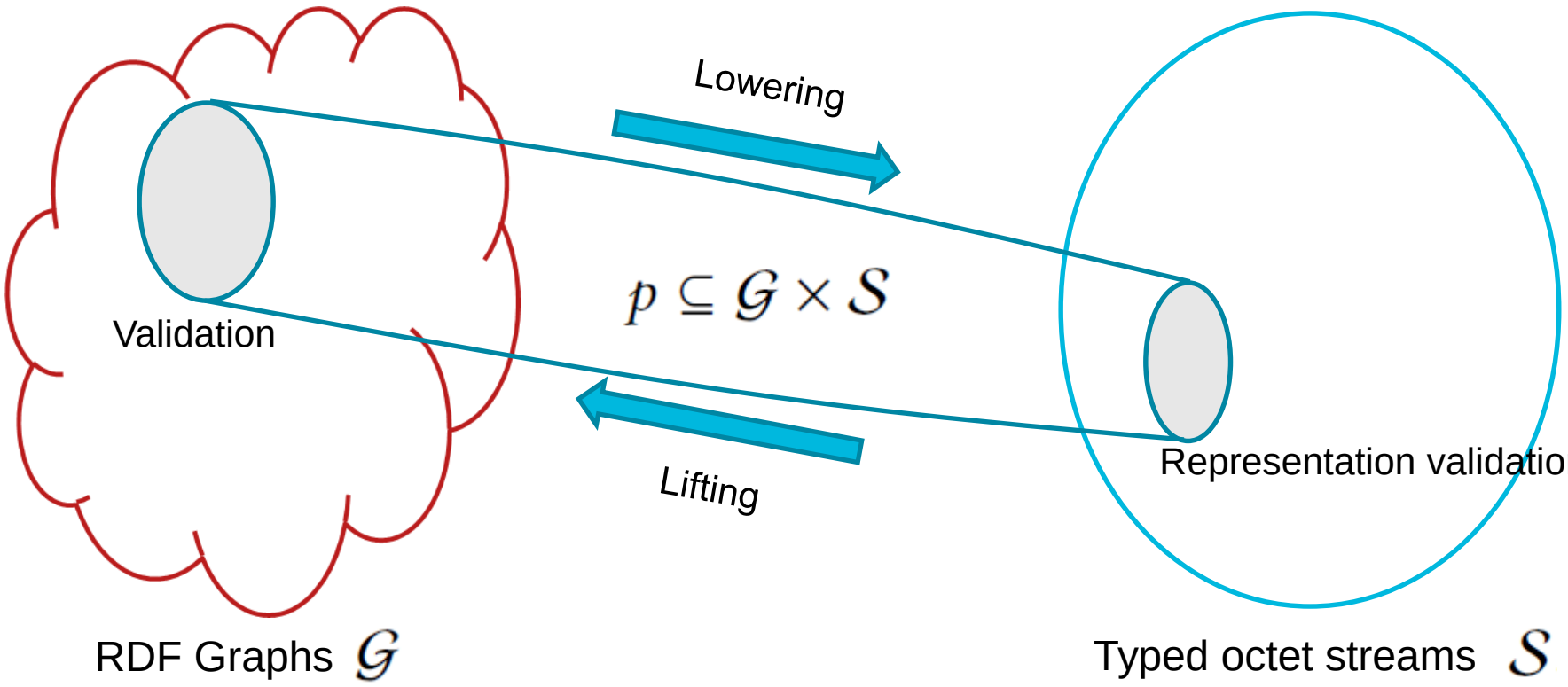


RDF Presentation

$$\forall \langle g, s \rangle, \langle g', s' \rangle \in p,\ \text{type}(s) = \text{type}(s') = t$$

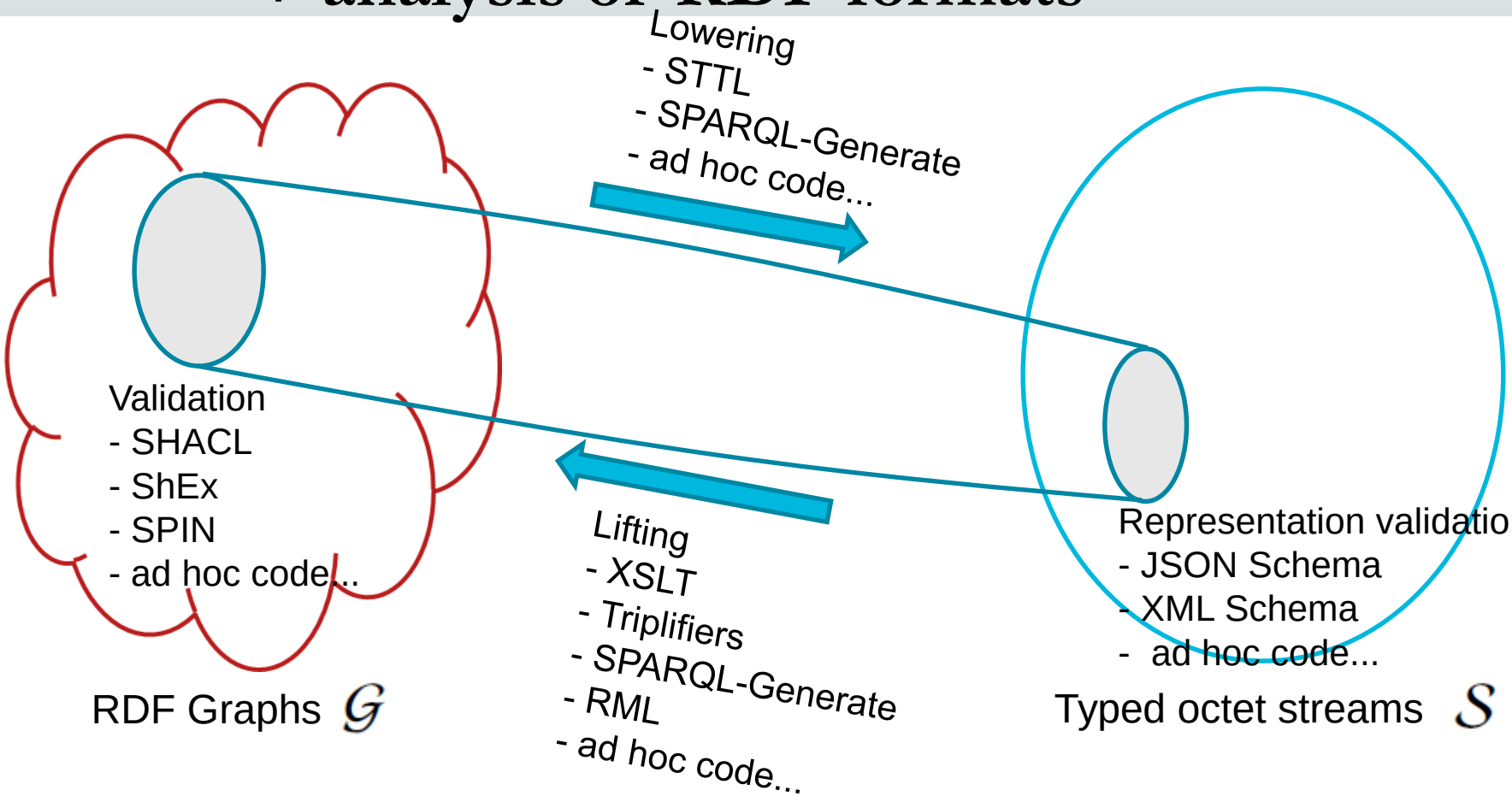
$$\forall \langle g, s \rangle, \langle g', s' \rangle \in p,\ s = s' \Rightarrow g = g'$$

Lifting, lowering, validating

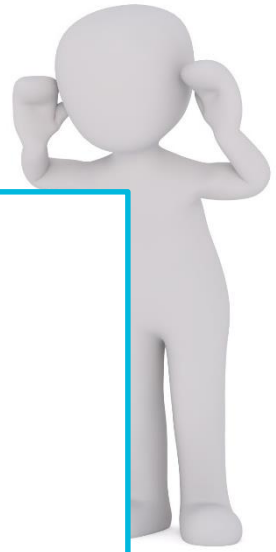


Lifting rule maps all the *tos* in p to their corresponding graph (unique)
Lowering rule maps all the graphs in p to a *chosen* corresponding typed stream
Validation rule checks if there is a pair with that graph
Representation validation rule checks if there is a pair with that *tos*

+ categorisation of tools
+ analysis of RDF formats



Usage scenarios on the Web



send

ceiver

The sender can be…

- Req/Res: client sends some content to the server
- Req/Res: server responds to the client
- Pub/Sub: client broadcasts some content to its subscribers

Both agents can also…

- be constrained in some ways (battery, storage, comput,…)
- implement some of the principles we devise (be *semantically flexible*)
- rely on a third party to operate lifting/lowering/…

Scenario 1: server/publisher sends its message to a client/subscriber

initiates interaction

I lowered this way, just lift accordingly

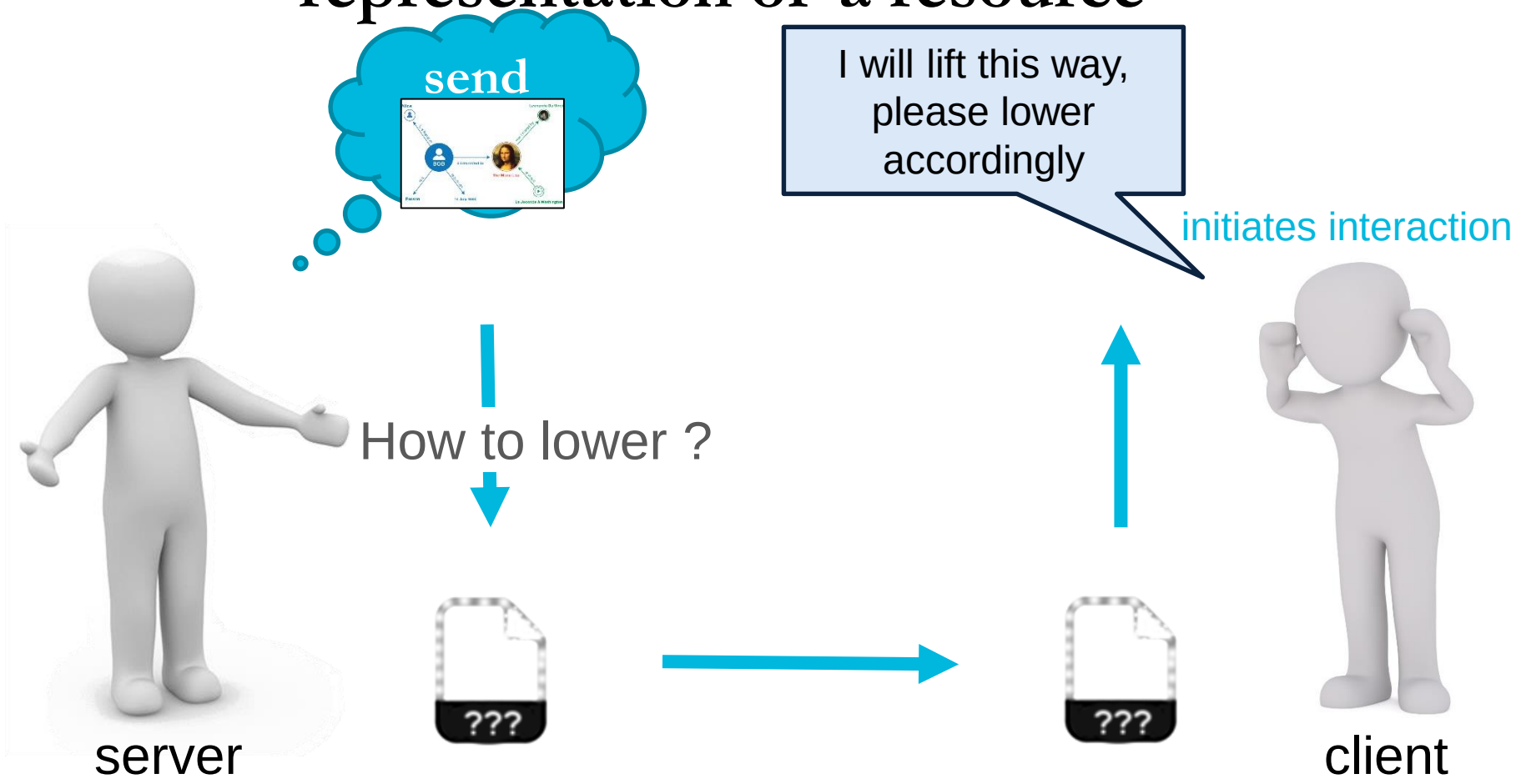
server/ publisher

How to lift ?

client/ subscriber

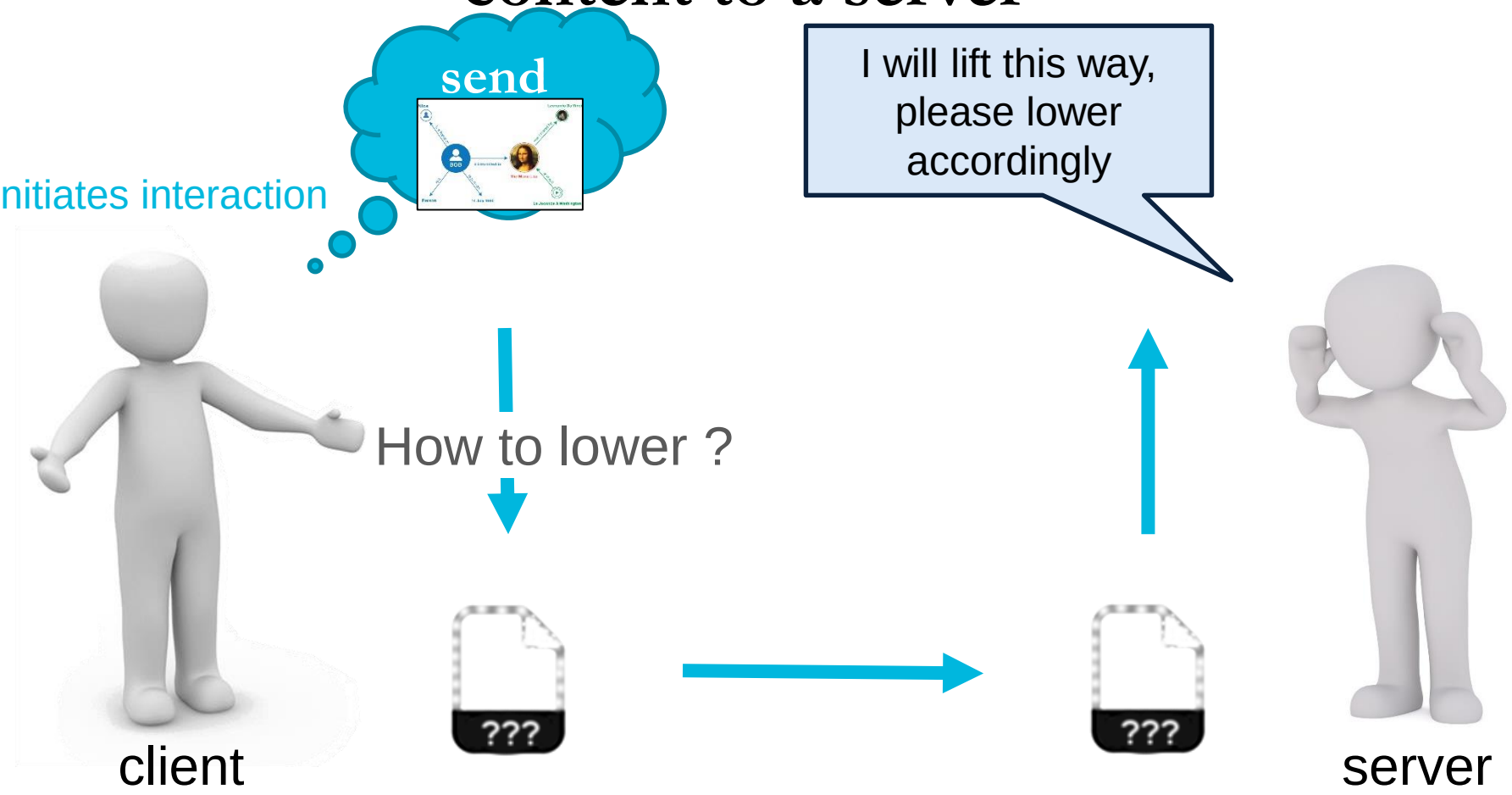
receiver discovers how to lift (from the sender, or elsewhere)
sender can be constrained… / receiver can be constrained…

Scenario 2: A client asks a server for the representation of a resource



server discovers a presentation suitable for the client
client can be constrained… / server can be constrained…

Scenario 3: A client sends some encoded content to a server



client discovers a presentation suitable for the server
client can be constrained… / server can be constrained…

Institut Mines-Télécom

Take away message

1. RDF Transformation Language
2. Use it and contribute
3. Not just a tool

Maxime Lefrançois
Maxime.Lefrancois@emse.fr
http://maxime-lefrancois.info/

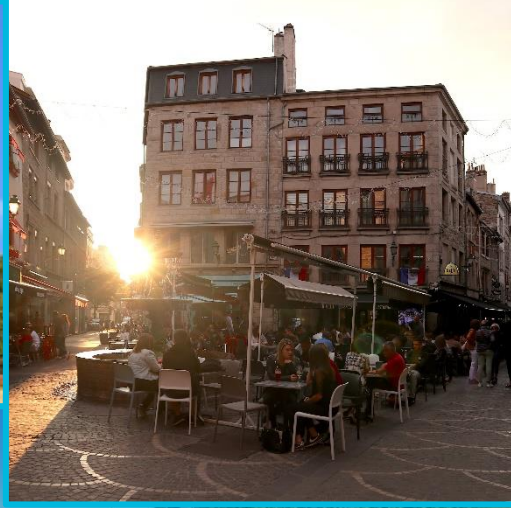
MINES Saint-Étienne – Institut Henri Fayol
Laboratoire Hubert Curien UMR CNRS 5516

MINES Saint-Étienne
Une école de l'IMT

LABORATOIRE HUBERT CURIEN
UMR • CNRS • 5516 • SAINT-ETIENNE

Ontologies, langages, tools, to ease the usage of RDF as a lingua franca





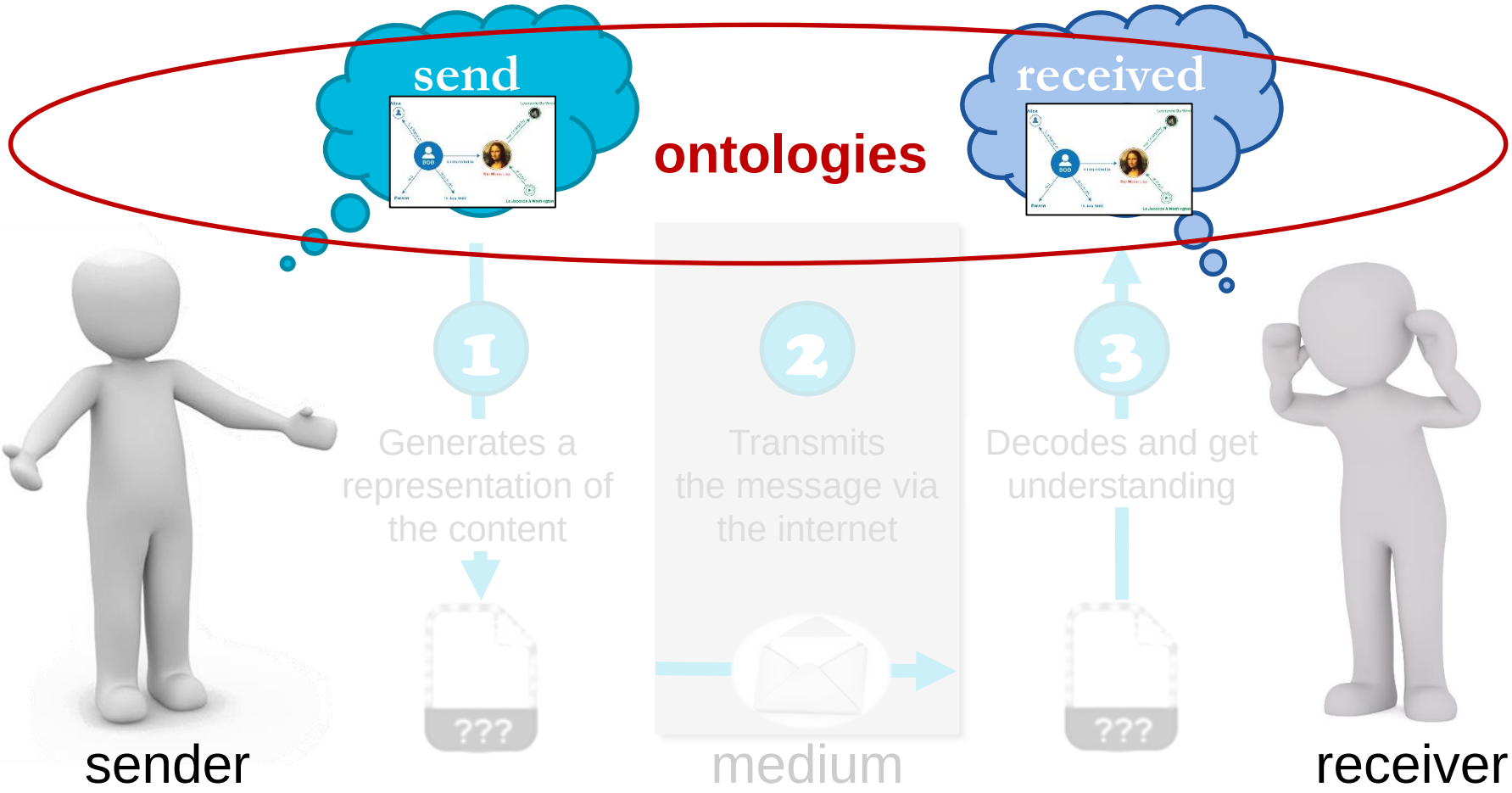







2
T-ETIENNE (LOIRE), 8 DECEMBRE 2018

Assumption:
content is always a RDF graph



Ontologies, contribution to standardization
SOSA/SSN ontology

Standard OGC et W3C

W3C Recommendation

Semantic Sensor Network Ontology

OGC® Making location count. W3C

W3C Recommendation 19 October 2017 (Link errors corrected 08 December 2017)

This version:
https://www.w3.org/TR/2017/REC-vocab-ssn-20171019/

Latest published version:
https://www.w3.org/TR/vocab-ssn/

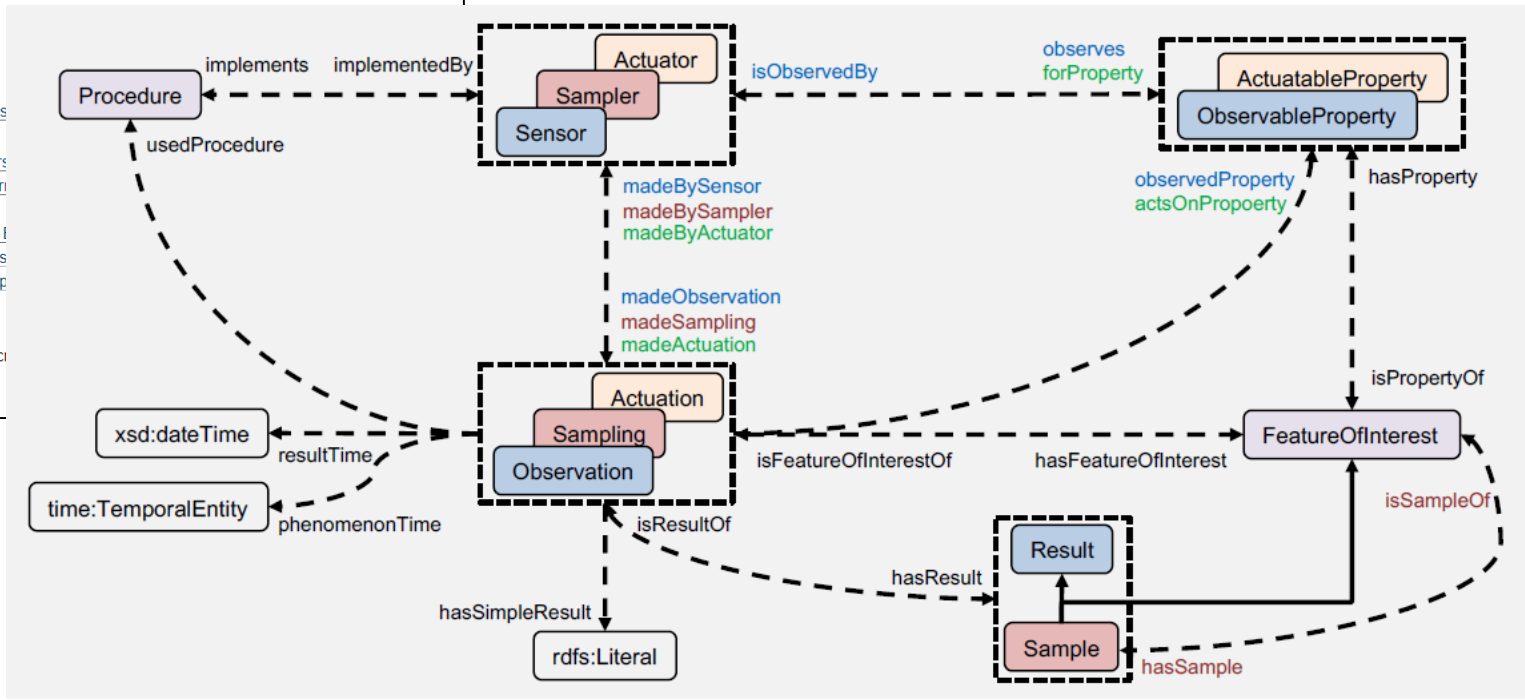
Latest editor's draft:
https://w3c.github.io/sdw/ssn/

Implementation report:
https://w3c.github.io/sdw/ssn-usage/

Previous version:
https://www.w3.org/TR/2017/PR-vocab-s

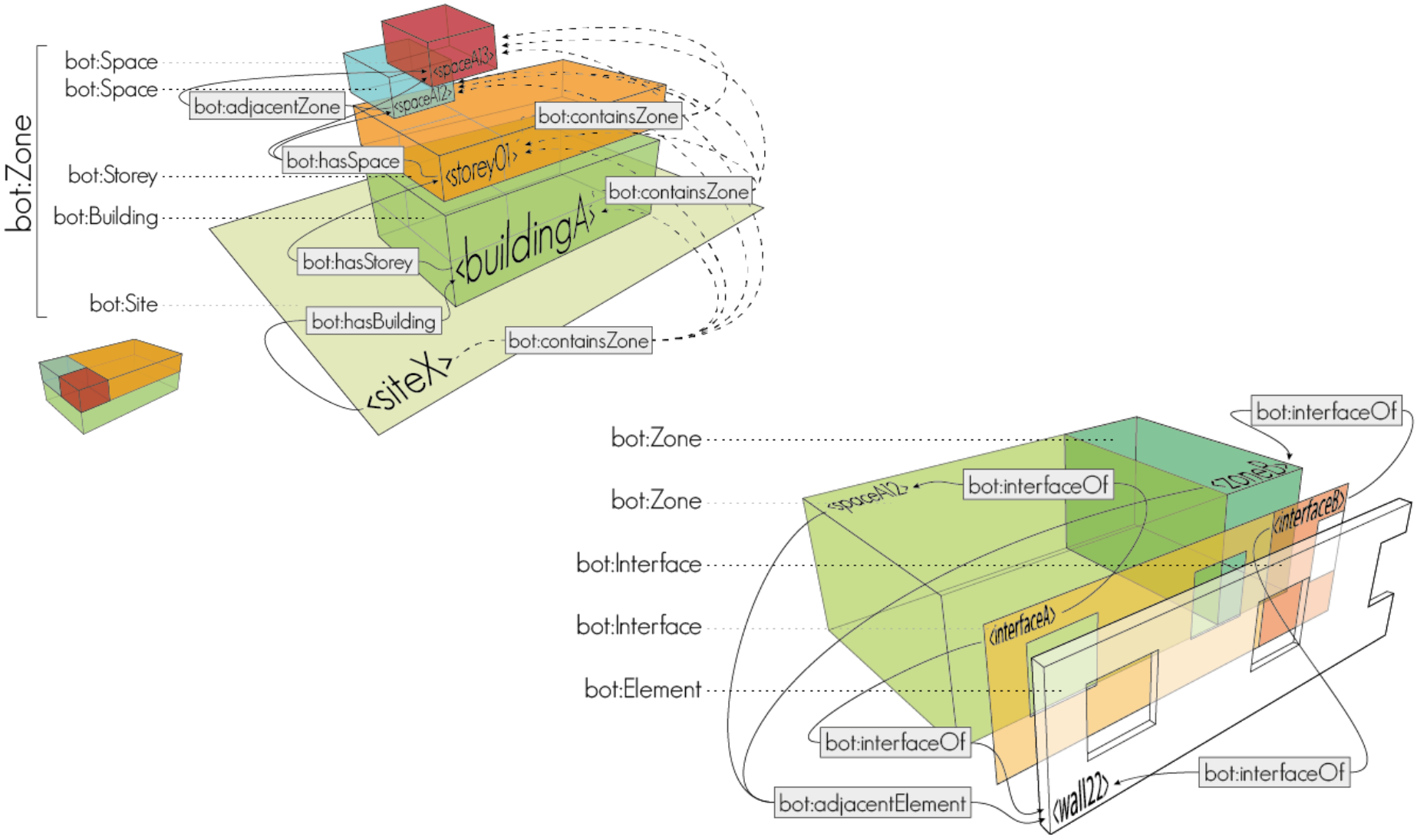
Editors:
Armin Haller, Australian National Univers
Krzysztof Janowicz, University of Califor
Simon Cox, CSIRO
Danh Le Phuoc, Technical University of
Kerry Taylor, Australian National Univers
Maxime Lefrançois, École Nationale Sup

Contributors (ordered alphabetically):
Rob Atkinson, Metalinkage
Raúl García-Castro, Universidad Politécn
Joshua Lieberman, Tumbling Walls
Claus Stadler, Universität Leipzig



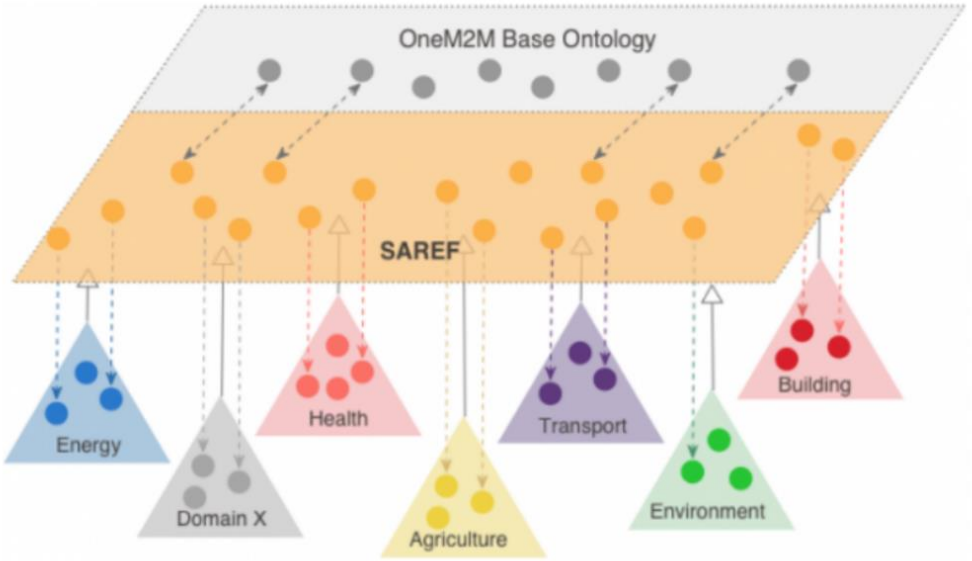
A. Haller, K. Janowicz, S. Cox, D. Le Phuoc, K. Taylor, M. Lefrançois, Semantic Sensor Network Ontology, W3C Recommendation, W3C, 19 October 2017

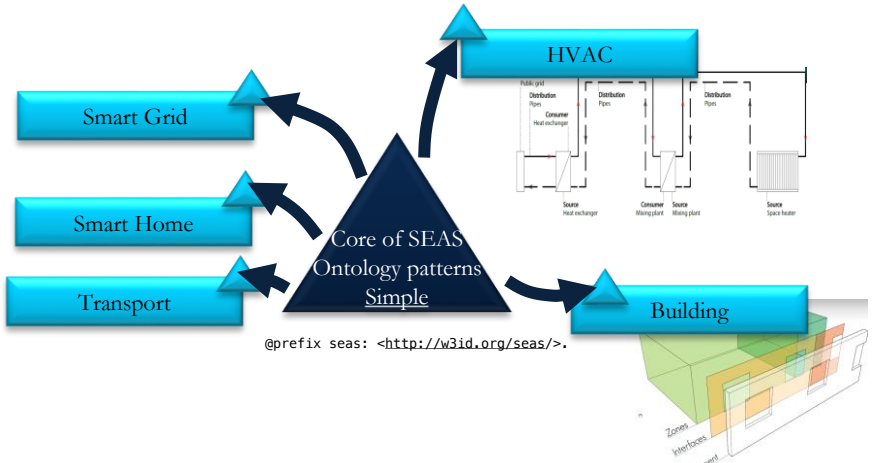
Ontologies, contribution to standardization
BOT ontology

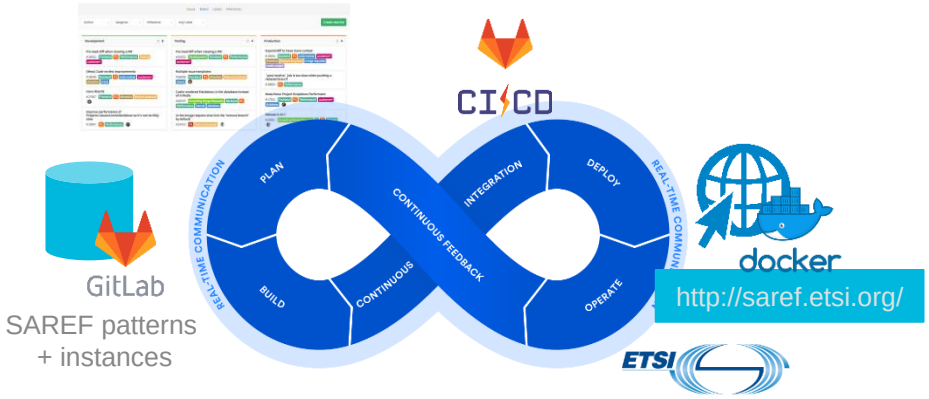


M. H. Rasmussen, M. Lefrançois, G. F. Schneider, P. Pauwels, BOT: the Building Topology Ontology of the W3C Linked Building Data Group (submitted May 2019 to Semantic Web Journal)

Ontologies, contribution to standardization
SAREF + influences



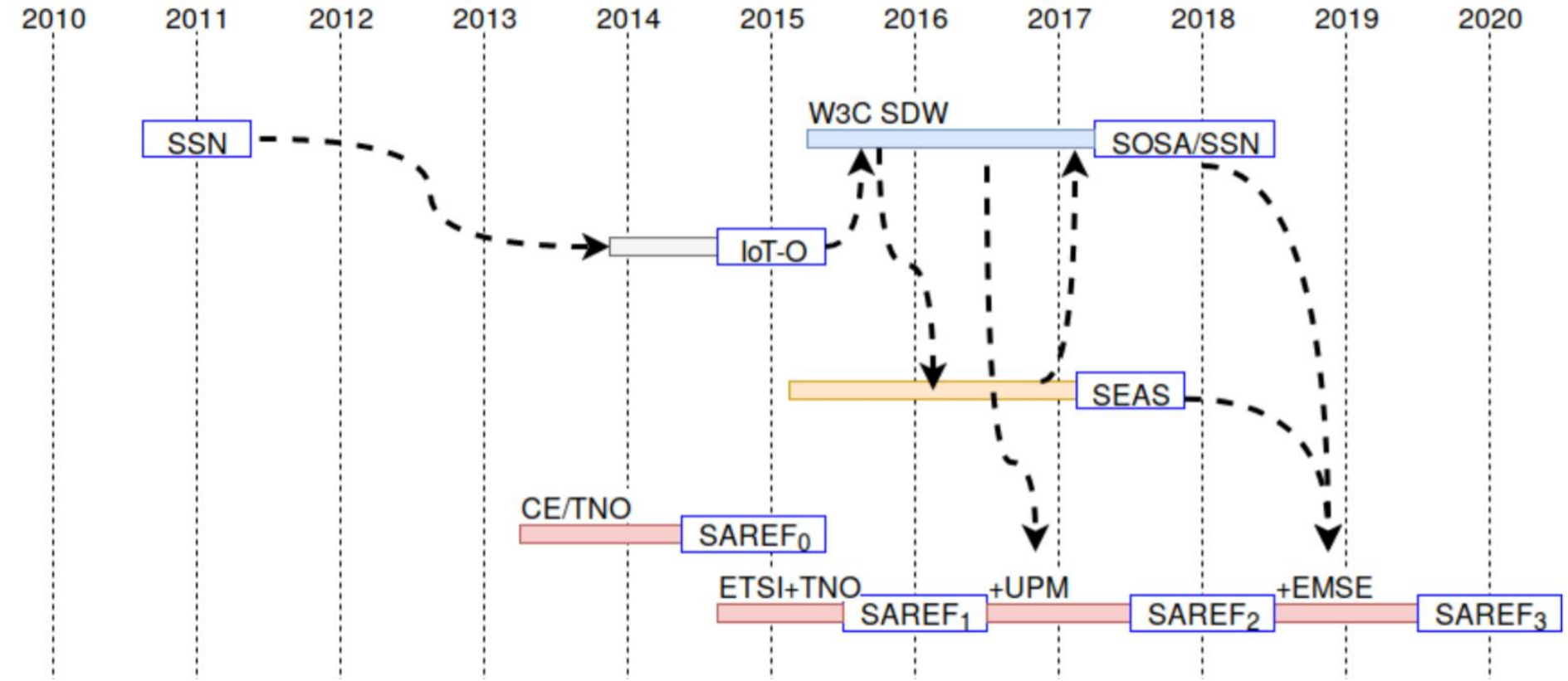




ETSI STF 556: Consolidation of SAREF and its community of industrial users, based on the experience of the EUREKA ITEA 12004 SEAS

ETSI STF DP: Specification of the SAREF development framework and workflow, and development of the Community SAREF Portal for user engagement

Ontologies, contribution to standardization
A work-in progress at international level



N. Seydoux, M. Lefrançois, L. Médini, . F. Schneider, P. Pauwels, Positionnement sur le Web Sémantique des Objets, Ingénierie des Connaissances 2019 (best paper award)