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

Session 3. Validating RDF data

Jose Emilio Labra Gayo
WESO Research group
University of Oviedo, Spain

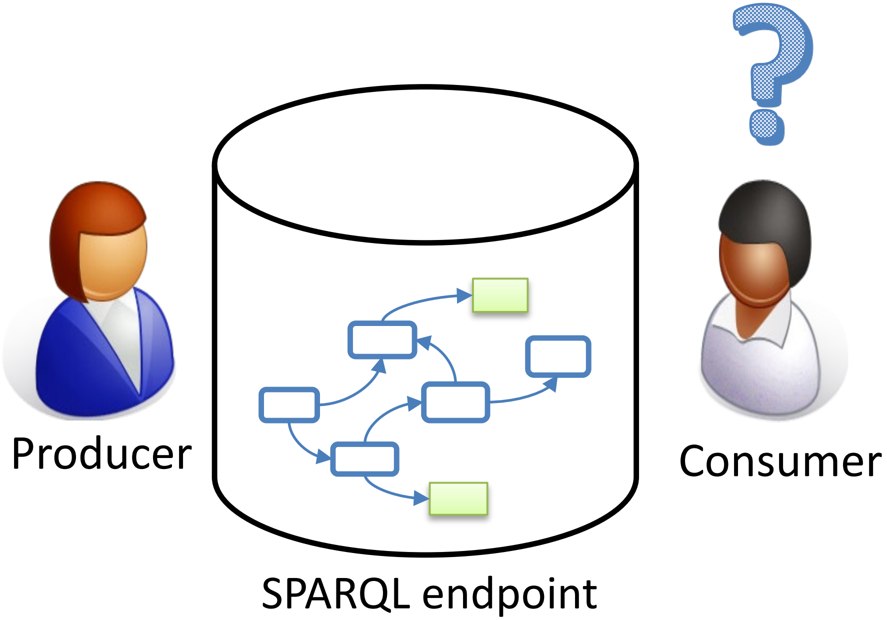
RDF, the good parts...

- RDF as an integration language
  - RDF as a *lingua franca* for semantic web and linked data
  - Basis for knowledge representation
- RDF flexibility
  - Data can be adapted to multiple environments
  - Reusable data by default
- RDF tools
  - RDF data stores & SPARQL
  - Several serializations: Turtle, JSON-LD, RDF/XML...
  - Can be embedded in HTML (Microdata/RDFa)

RDF, the other parts

Consuming & producing RDF

- Describing and validating RDF content
- SPARQL endpoints are not well documented
  - Typical documentation = set of SPARQL queries
  - Difficult to know where to start doing queries



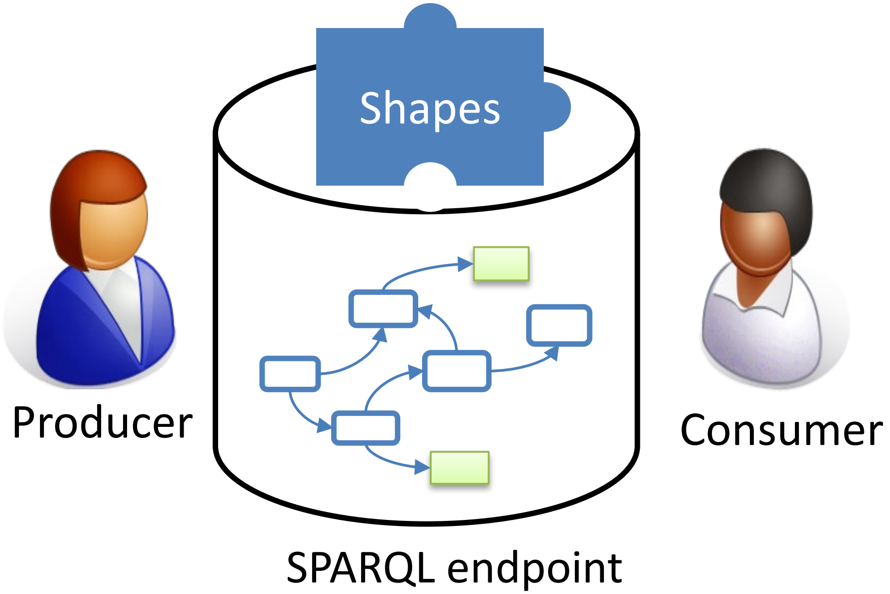
Why describe & validate RDF?

For producers

- Developers can understand the contents they are going to produce
- They can ensure they produce the expected structure
- Advertise and document the structure
- Generate interfaces

For consumers

- Understand the contents
- Verify the structure before processing it
- Query generation & optimization



Similar technologies

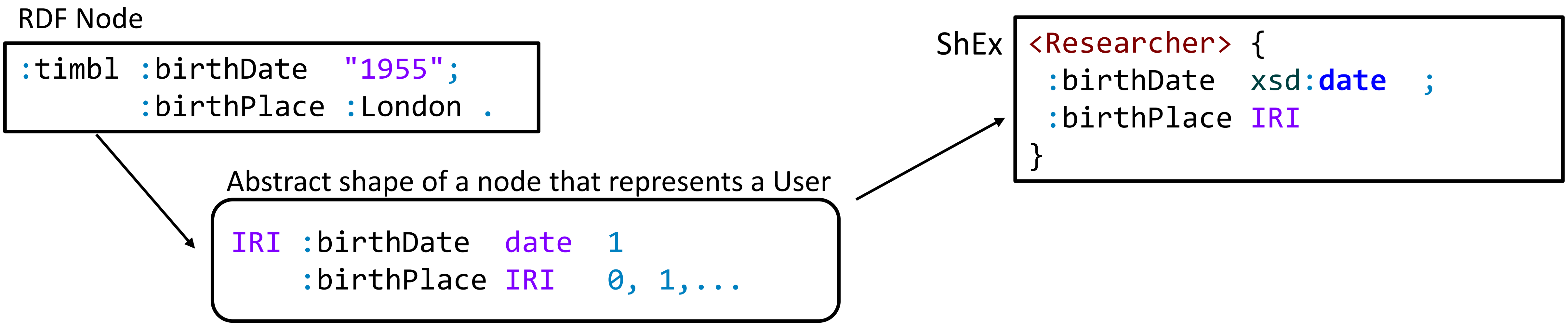
Technology	Schema
Relational Databases	DDL
XML	DTD, XML Schema, RelaxNG, Schematron
Json	Json Schema
RDF	?

Fill that gap

What is an RDF shape?

A shape can describe

- The form of the node itself (node constraint)
- The number of possible arcs incoming/outgoing from a node
- The possible values associated with those arcs



RDF Node

```
:timbl :birthDate  "1955";
       :birthPlace :London .
```

Abstract shape of a node that represents a User

```
IRI :birthDate  date  1
    :birthPlace IRI   0, 1,...
```

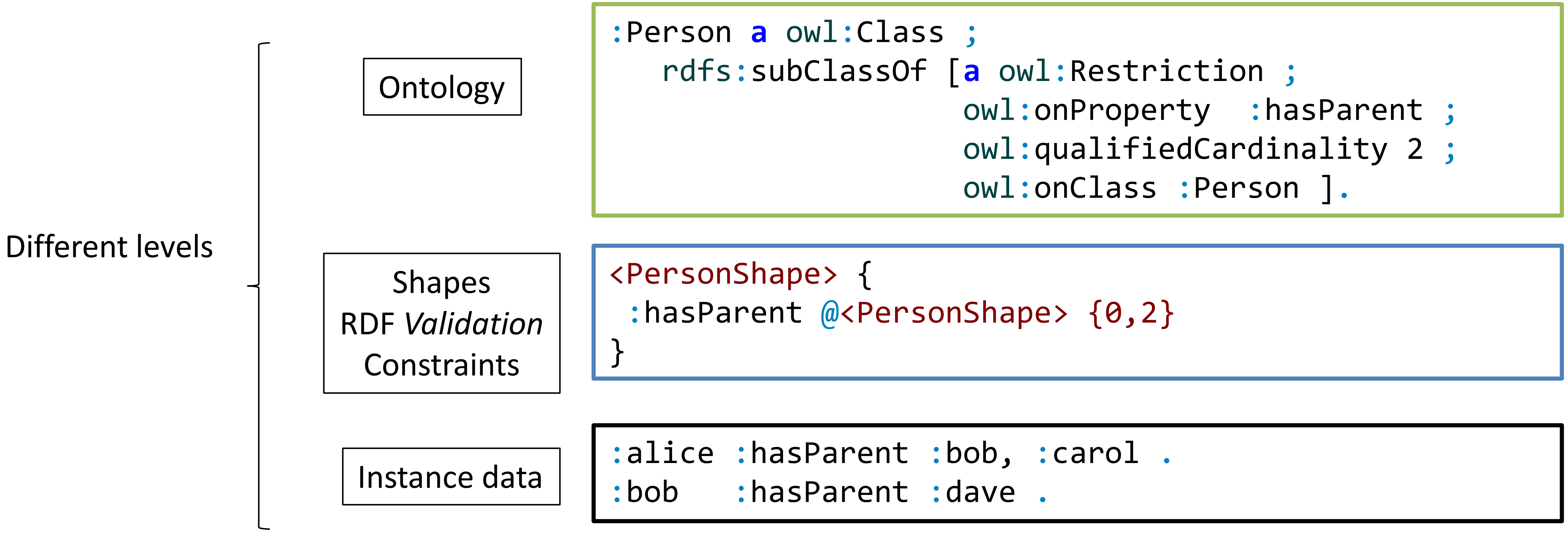
ShEx

```
<Researcher> {
 :birthDate  xsd:date  ;
 :birthPlace IRI
}
```

Shapes vs Ontologies

Ontologies ≠ Shapes ≠ instance data

- Ontologies are usually focused on domain entities (higher level)
- RDF validation/shapes focused on RDF graph features (lower level)



Different levels

Ontology

```
:Person a owl:Class ;
   rdfs:subClassOf [a owl:Restriction ;
                     owl:onProperty   :hasParent ;
                     owl:qualifiedCardinality 2 ;
                     owl:onClass :Person ].
```

Shapes
RDF *Validation* Constraints

```
<PersonShape> {
 :hasParent @<PersonShape> {0,2}
}
```

Instance data

```
:alice  :hasParent :bob, :carol .
:bob    :hasParent :dave .
```

ShEx and SHACL

2013 RDF Validation Workshop

- Conclusions of the workshop:
  - *There is a need of a higher level, concise language for RDF Validation*
- ShEx initially proposed (v 1.0)

2014 W3c Data Shapes WG chartered

2017 SHACL accepted as W3C recommendation

2017 ShEx 2.0 released as W3C Community group draft

2019 ShEx adopted by Wikidata

Short intro to ShEx

ShEx (Shape Expressions Language)

Concise and human-readable

Syntax similar to SPARQL, Turtle

Semantics inspired by regular expressions & RelaxNG

2 syntaxes: Compact and RDF/JSON-LD

Official info: http://shex.io

Semantics: http://shex.io/shex-semantics/, primer: http://shex.io/shex-primer

ShEx implementations and playgrounds

Implementations:

- shex.js: Javascript
- SHaclEX: Scala (Jena/RDF4j)
- PyShEx: Python
- shex-java: Java
- Ruby-ShEx: Ruby
- Elixir

Online demos & playgrounds

- ShEx-simple
- RDFShape
- ShEx-Java
- ShExValidata
- Wikishape

Simple example

Prefix declarations as in Turtle/SPARQL

```
prefix schema: <http://schema.org/>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>

<User> IRI {
 schema:name  xsd:string   ;
 schema:knows @<User>    *
}
```

Nodes conforming to <User> shape must:

- Be IRIs
- Have exactly one schema:name with a value of type xsd:string
- Have zero or more schema:knows whose values conform to <User>

RDF Validation using ShEx

Schema

```
<User> IRI {
 schema:name   xsd:string    ;
 schema:knows @<User>      *
}
```

Shape map

```
:alice@<User>✓
:bob   @<User>✓
:carol@<User>✗
:dave  @<User>✗
:emily@<User>✗
:frank@<User>✓
:grace@<User>✗
```

Try it (RDFShape): https://goo.gl/97bYdv
Try it (ShExDemo):https://goo.gl/Y8hBsW

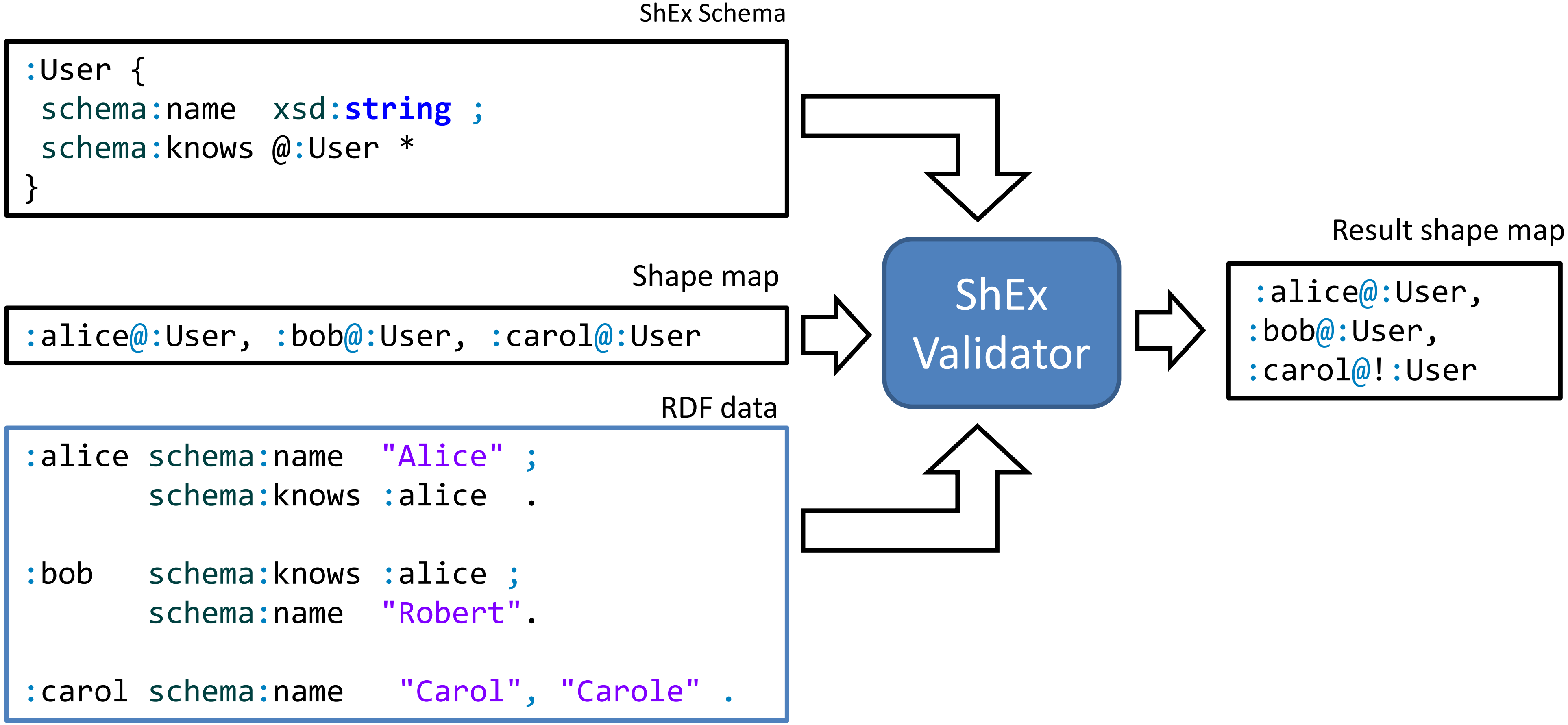
Data

```
:alice schema:name   "Alice"  ;
       schema:knows :alice   .

:bob   schema:knows :alice ;
       schema:name  "Robert".

:carol schema:name    "Carol", "Carole" .

:dave  schema:name    234 .

:emily foaf:name      "Emily" .

:frank schema:name "Frank" ;
       schema:email <mailto:frank@example.org> ;
       schema:knows :alice, :bob .

:grace schema:name "Grace" ;
       schema:knows :alice, _:1 .

_:1 schema:name  "Unknown" .
```

Validation process

Input: RDF data, ShEx schema, Shape map

Output: Result shape map



ShEx Schema

```
:User {
 schema:name   xsd:string ;
 schema:knows @:User *
}
```

Shape map

```
:alice@:User, :bob@:User, :carol@:User
```

RDF data

```
:alice schema:name   "Alice" ;
       schema:knows :alice   .

:bob   schema:knows :alice ;
       schema:name   "Robert".

:carol schema:name    "Carol", "Carole" .
```

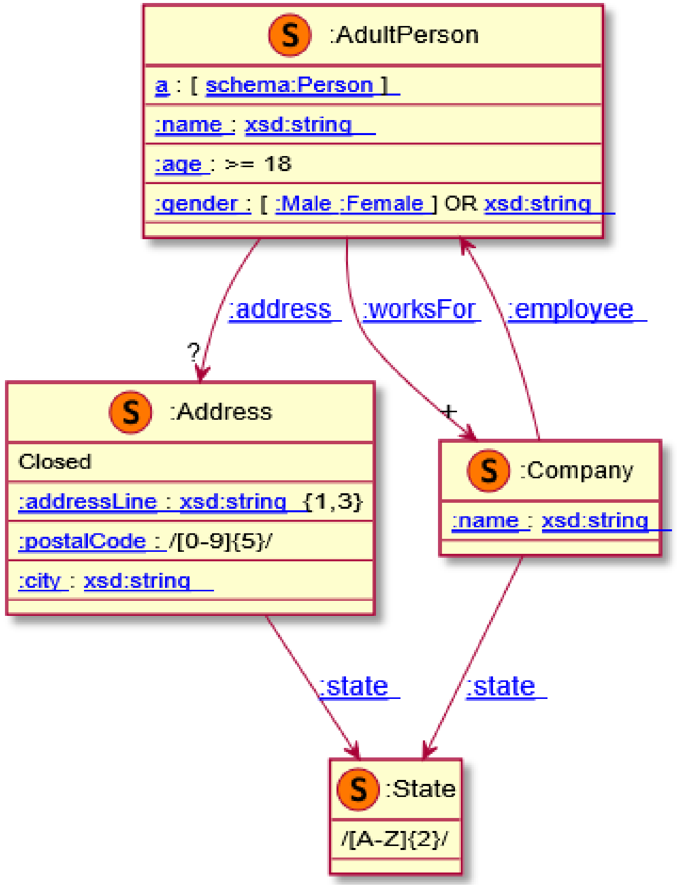
Result shape map

```
:alice@:User,
:bob@:User,
:carol@!:User
```

Example with more ShEx features

```
:AdultPerson EXTRA rdf:type {
 rdf:type    [ schema:Person ]               ;
 :name       xsd:string                      ;
 :age        MinInclusive 18                 ;
 :gender     [:Male :Female] OR xsd:string ;
 :address    @:Address ?                     ;
 :worksFor   @:Company +
}
:Address CLOSED {
 :addressLine xsd:string {1,3}
 :postalCode  /[0-9]{5}/
 :state       @:State
 :city        xsd:string
}
:Company {
 :name      xsd:string
 :state     @:State
 :employee @:AdultPerson *
}
:State   /[A-Z]{2}/
```

```
:alice rdf:type :Student, schema:Person ;
 :name        "Alice" ;
 :age         20 ;
 :gender      :Male ;
 :address     [
  :addressLine   "Bancroft Way" ;
  :city          "Berkeley" ;
  :postalCode    "55123" ;
  :state         "CA"
 ] ;
 :worksFor   [
  :name         "Company" ;
  :state        "CA"      ;
  :employee     :alice
] .
```



Try it: https://tinyurl.com/yd5hp9z4

SHACL

SHACL (Shapes Constraint Language)

W3C recommendation:

https://www.w3.org/TR/shacl/ (July 2017)

RDF vocabulary

2 parts: SHACL-Core, SHACL-SPARQL

SHACL implementations

Name	Parts	Language - Library	Comments
Topbraid SHACL API	SHACL Core, SPARQL	Java (Jena)	Used by TopBraid composer
SHACL playground	SHACL Core	Javascript (rdflib.js)	http://shacl.org/playground/
SHACL-S Part of SHaclEX	SHACL Core	Scala (Jena, RDF4j)	http://rdfshape.weso.es
pySHACL	SHACL Core, SPARQL	Python (rdflib)	https://github.com/RDFLib/pySHACL
Corese SHACL	SHACL Core, SPARQL	Java (STTL)	http://wimmics.inria.fr/corese
RDFUnit	SHACL Core, SPARQL	Java (Jena)	https://github.com/AKSW/RDFUnit
Jena SHACL	SHACL Core, SPARQL	Java (Jena)	https://jena.apache.org/
RDf4j SHACL	SHACL Core	Java (RDF4J)	https://rdf4j.org
Stardog	SHACL Core, SPARQL	Java	https://www.stardog.com
Zazuko SHACL	SHACL Core	Javascript	https://github.com/zazuko/rdf-validate-shacl

In this tutorial we will use RDFShape online demo which supports:

- SHaclEX (SHACL-s)
- JenaSHACL
- SHACL TQ (SHACL TopBraid API)

Basic example

```
prefix :       <http://example.org/>
prefix sh:     <http://www.w3.org/ns/shacl#>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>

:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:nodeKind sh:IRI ;
   sh:property :hasName,
               :hasEmail .
:hasName sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string .
:hasEmail sh:path schema:email ;
   sh:minCount 1;
   sh:maxCount 1;
   sh:nodeKind sh:IRI .
```

Shapes graph

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   schema:firstName "Bob" ;
       schema:email <mailto:bob@mail.org> . ☹

:carol schema:name "Carol" ;          ☹
       schema:email "carol@mail.org" .
```

Data graph

Try it. RDFShape https://goo.gl/ukY5vq

Same example with blank nodes

```
prefix :       <http://example.org/>
prefix sh:     <http://www.w3.org/ns/shacl#>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>

:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:nodeKind sh:IRI ;
   sh:property [
    sh:path schema:name ;
    sh:minCount 1; sh:maxCount 1;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
   sh:path schema:email ;
   sh:minCount 1; sh:maxCount 1;
   sh:nodeKind sh:IRI ;
  ] .
```

Shapes graph

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   schema:firstName "Bob" ;
       schema:email <mailto:bob@mail.org> .           ☹

:carol schema:name "Carol" ;                          ☹
       schema:email "carol@mail.org" .
```

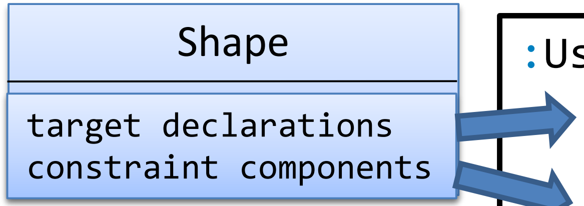
Data graph

Try it. RDFShape https://goo.gl/ukY5vq

Some definitions about SHACL

Shape: collection of targets and constraints components

- Targets: specify which nodes in the data graph must conform to a shape
- Constraint components: Determine how to validate a node



Shape

target declarations
constraint components

```
:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:nodeKind sh:IRI ;
   sh:property :hasName,
               :hasEmail .
:hasName sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string .
. . .
```

Validation Report

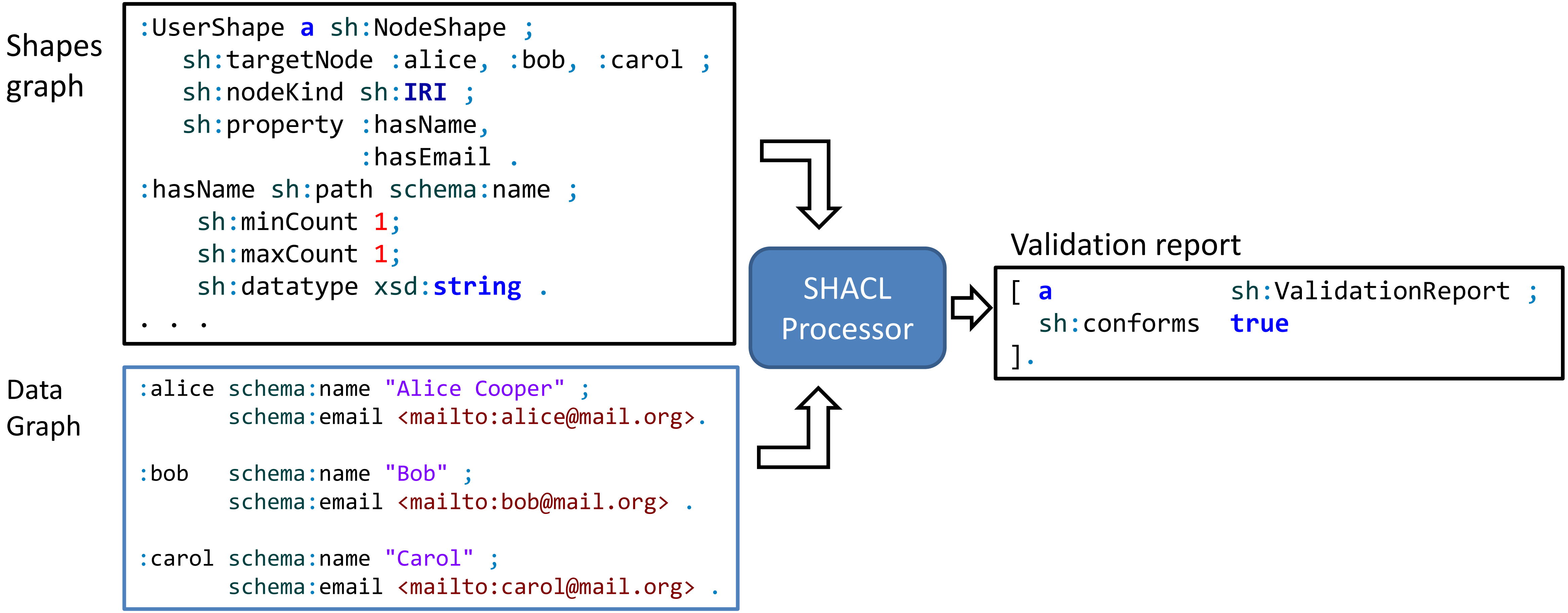
Output of validation process = list of violation errors

No errors ⇒ RDF conforms to shapes graph

```
[ a            sh:ValidationReport ;
  sh:conforms  true
].
```

```
[ a            sh:ValidationReport ;
  sh:conforms  false ;
  sh:result    [
   a             sh:ValidationResult ;
   sh:focusNode  :bob ;
   sh:message
     "MinCount violation. Expected 1, obtained: 0" ;
   sh:resultPath schema:name ;
   sh:resultSeverity sh:Violation ;
   sh:sourceConstraintComponent
     sh:MinCountConstraintComponent ;
   sh:sourceShape  :hasName
] ;
...
```

SHACL processor



Shapes graph

```
:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:nodeKind sh:IRI ;
   sh:property :hasName,
               :hasEmail .
:hasName sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string .
. . .
```

Data Graph

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org>.

:bob   schema:name "Bob" ;
       schema:email <mailto:bob@mail.org> .

:carol schema:name "Carol" ;
       schema:email <mailto:carol@mail.org> .
```

Validation report

```
[ a             sh:ValidationReport ;
  sh:conforms   true
].
```

Longer example

In ShEx

```
:AdultPerson EXTRA a {
 a           [ schema:Person ]          ;
 :name       xsd:string                 ;
 :age        MinInclusive 18            ;
 :gender     [:Male :Female] OR xsd:string ;
 :address    @:Address ?                ;
 :worksFor   @:Company +                ;
}
:Address CLOSED {
 :addressLine xsd:string {1,3}          ;
 :postalCode  /[0-9]{5}/                ;
 :state        @:State                  ;
 :city         xsd:string
}
:Company {
 :name      xsd:string                  ;
 :state     @:State                     ;
 :employee @:AdultPerson *              ;
}
:State   /[A-Z]{2}/
```

Try it: https://tinyurl.com/ycl3mkzr

In SHACL

```
:AdultPerson a sh:NodeShape ;
  sh:property [
   sh:path rdf:type ;
   sh:qualifiedValueShape [
      sh:hasValue schema:Person
   ];
   sh:quali
   sh:quali
 ] ;
 sh:targetN
   sh:prope
    sh:minC
    sh:data
  ] ;
  sh:proper
   sh:minCo
   sh:in (
  ] ;
  sh:proper
   sh:maxCo
   sh:minIn
  ] ;
 sh:propert
   sh:node :Address
   sh:minCount 1 ; sh
  ] ;
  sh:property [ sh:path :worksFor ;
    sh:node :Company ;
    sh:minCount 1 ; sh:maxCoun
  ].
```

```
:Address a sh:NodeShape ;
 sh:closed true ;
 sh:property [ sh:path :addressLine;
   sh:datatype xsd:string ;
   sh:min
 ] ;
 sh:prope
   sh:pat
   sh:min
 ] ;
 sh:prope
   sh:dat
   sh:min
 ] ;
 sh:prope
   sh:nod
 ] .
```

```
:Company a sh:NodeShape ;
 sh:property [ sh:path :name ;
   sh:datatype xsd:string
   ] ;
 sh:property [
   sh:path :state ;
   sh:node :State
 ] ;
 sh:property [ sh:path :employee ;
   sh:node :AdultPerson ;
 ] ;.

:State a sh:NodeShape ;
   sh:pattern "[A-Z]{2}" .
```

Its recursive!!! (not well defined SHACL)
Implementation dependent feature

ShEx and SHACL compared

Some similarities

- Similar goal: describe and validate RDF graphs
- Both employ the word "shape"
- Node constraints similar in both languages
- Constraints on incoming/outgoing arcs
- Both allow to define cardinalities
- Both have RDF syntax
- Both have an extension mechanism

ShEx and SHACL compared

Main differences

	ShEx	SHACL
Underlying philosophy	Structure definition	Constraint checking
Syntax	Compact syntax + RDF	RDF
Notion of shape	Only structure	Structure + target decls.
Default cardinalities	{1,1}	{0,*}
Shapes and inference	No	SHACL specific entailment
Recursion	Part of the language	Undefined
Repeated properties	Part of the language	Conjunction by default Requires qualifiedValueShapes
Property paths	Nested shapes	SPARQL like
Property pair comparisons	Unsupported in current version	Part of the language
Extension mechanism	Semantic actions	SHACL-SPARQL
Validation triggering	Query shape map	Target declarations
Result of validation	Result shape map	Validation report

ShEx: Shape Expressions

Describe RDF data

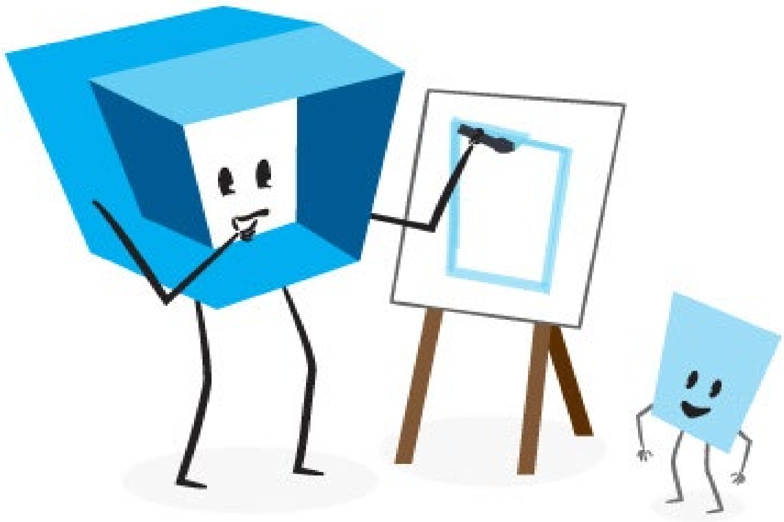
Descriptions focus on what is the shape of the RDF data graph

Focus = RDF graph

Validate = check if data matches the descriptions

Main focus = nodes that match (shape maps)

It can also check nodes that don’t match



Description

SHACL: Shapes Constraint Language

Constraints on RDF data

Constraints focus on the RDF data graph and things we don’t allow

Focus = RDF graph (data)

Validate = check if data doesn’t violate the constraints

Focus = nodes that don’t pass the constraints (violations)

But it can also check nodes that pass the constraints

Constraint

ShEx for Property graphs

- Recent paper: "ProGS: Property graph shapes language"
  - https://arxiv.org/abs/2107.05566
  - Extends SHACL to support Property graphs
- In the same way, PShEx has been defined as a ShEx extension to support Property graphs
  - Adds constraints on nodes/property qualifiers
  - Proposal recently published at: https://arxiv.org/abs/2110.11709

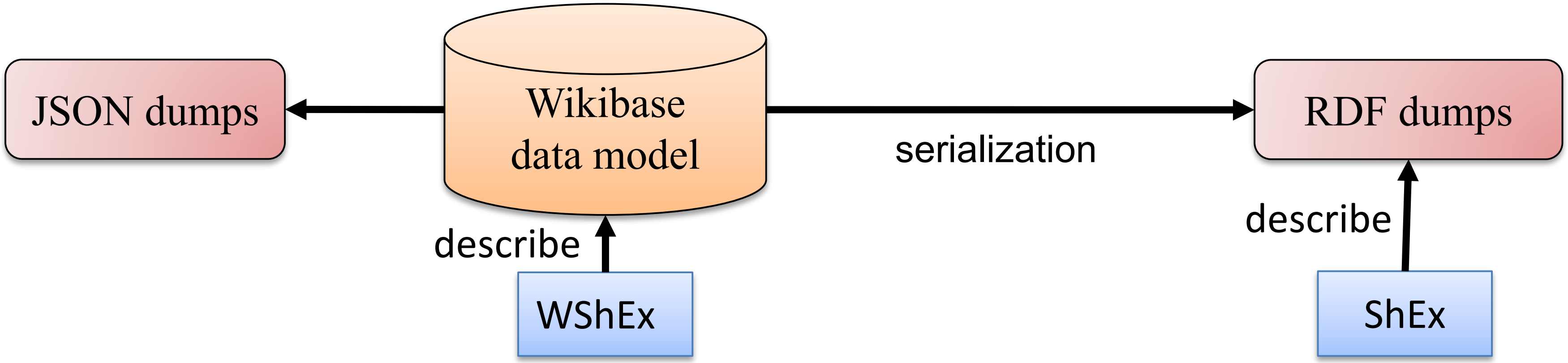
ShEx for Wikibase graphs

Wikidata already added support for ShEx

- Entity schemas extension

WShEx = extension of ShEx that supports qualifiers/references

- Can be used to describe and validate Wikibase graphs
- Defined in: https://arxiv.org/abs/2110.11709

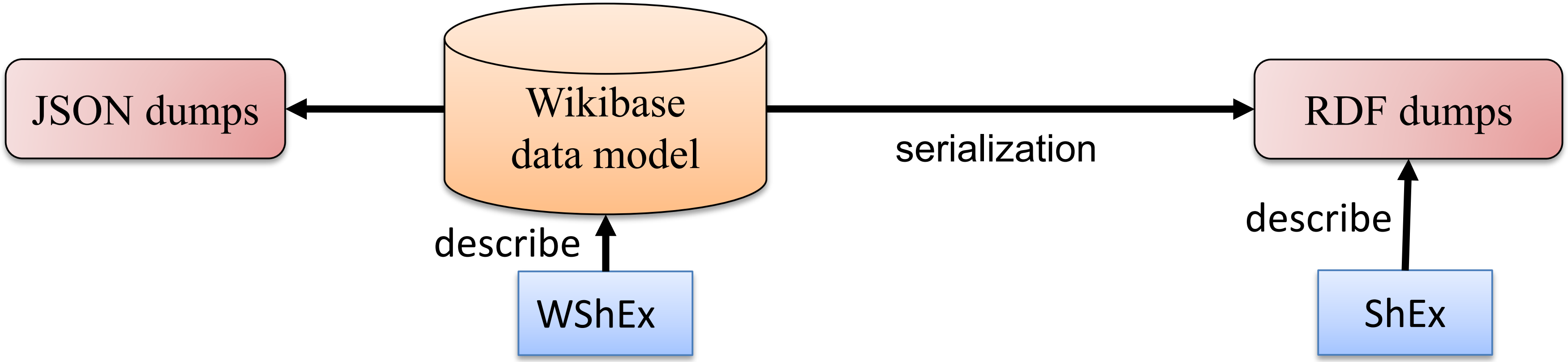


WShEx

```
<Researcher> {
 <birthPlace>       @<Place> ;
 <awarded>          @<Award> {{
    <togetherWith> @<Researcher> *
  }} *
}
<Place> {
 <country>          @<Country>
}
<Country> {}
```

ShEx

```
<Researcher> {
 wdt:birthPlace @<Place> ;
 p:birthPlace    {
    ps:birthPlace   @<Place>
 } ;
 wdt:awarded        @<Awarded> ;
 p:awarded {
   ps:awarded       @<Awarded> ;
   pq:togetherWith @<Researcher> *;
 } *
}
<Place> {
 wdt:country        @<Country> ;
 p:country {
  ps:country        @<Country> }
}
<Country> {}
```



Shapes Applications and tools

Wikidata and wikibase

In May, 2019, Wikidata announced ShEx adoption

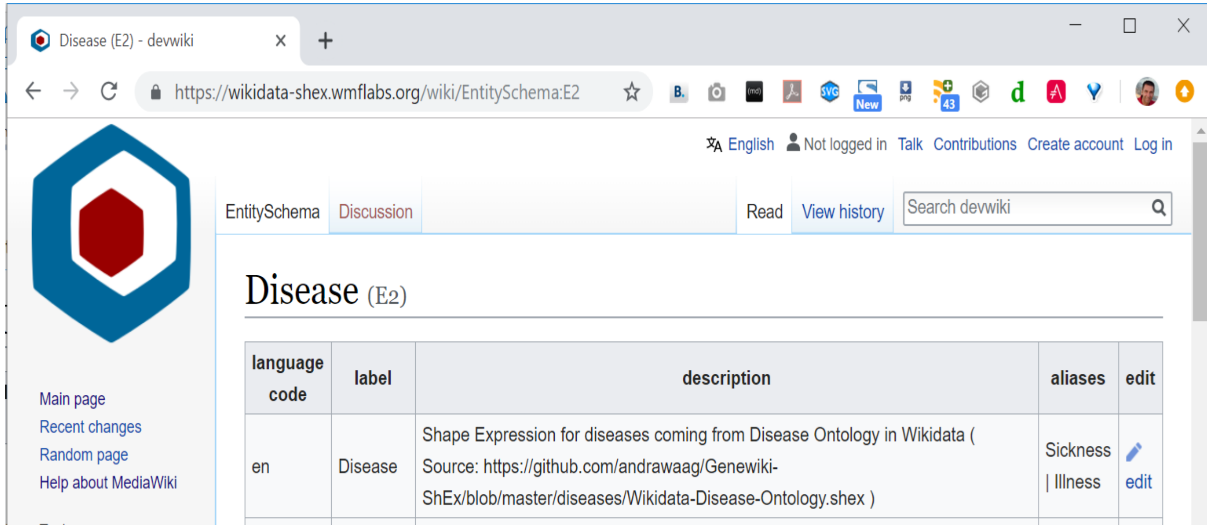
New namespace for schemas

Example:

https://www.wikidata.org/wiki/EntitySchema:E2

Wikibase also contains entity schemas

Online demo: [wikishape]()



Disease (E2)

language code	label	description	aliases	edit
en	Disease	Shape Expression for diseases coming from Disease Ontology in Wikidata (Source: https://github.com/andrawaag/Genewiki-ShEx/blob/master/diseases/Wikidata-Disease-Ontology.shex)	Sickness | Illness	edit

Solid project

SOLID (SOcial Linked Data): Promoted by Tim Berners-Lee

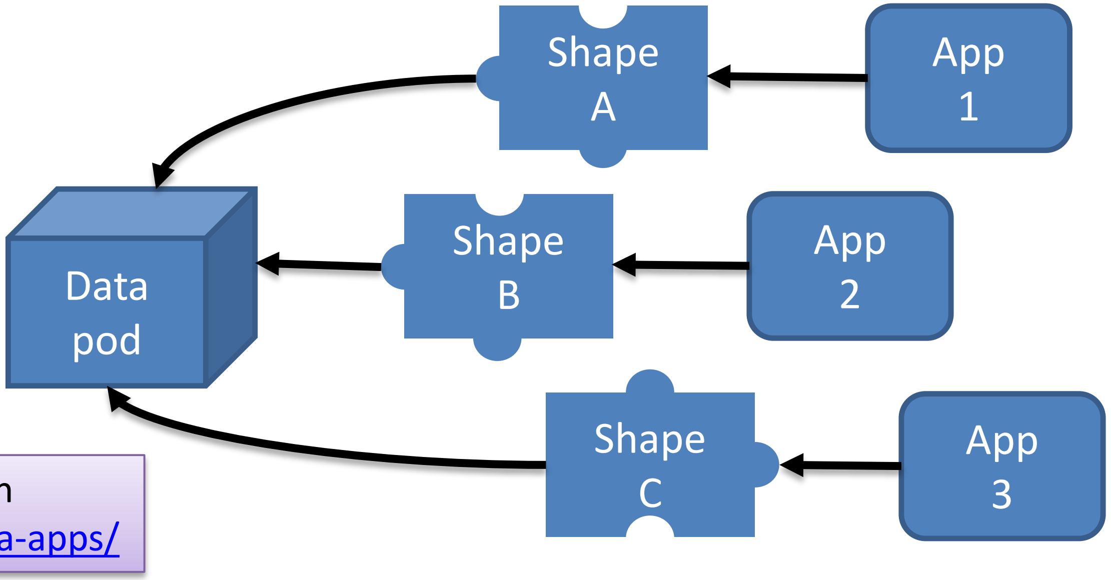
Goal: Re-decentralize the Web

Separate data from apps

Give users more control about their data

Internally using linked data & RDF

Shapes needed for interoperability



"...I just can't stop thinking about shapes.", Ruben Verborgh
https://ruben.verborgh.org/blog/2019/06/17/shaping-linked-data-apps/

Other use cases

HL7 FHIR.

Example: https://www.hl7.org/fhir/observation.html

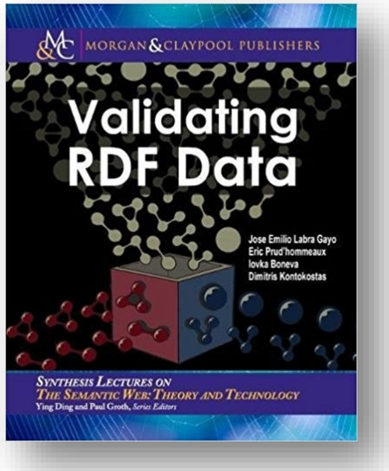
ELI validator

SHACL shapes obtained from Excel sheets:

https://webgate.ec.europa.eu/eli-validator/home

SHACL adoption supported by Top Quadrant

See: https://www.topquadrant.com/technology/shacl/



More info:
Chapter 6 of Validating RDF data: http://book.validatingrdf.com/bookHtml012.html

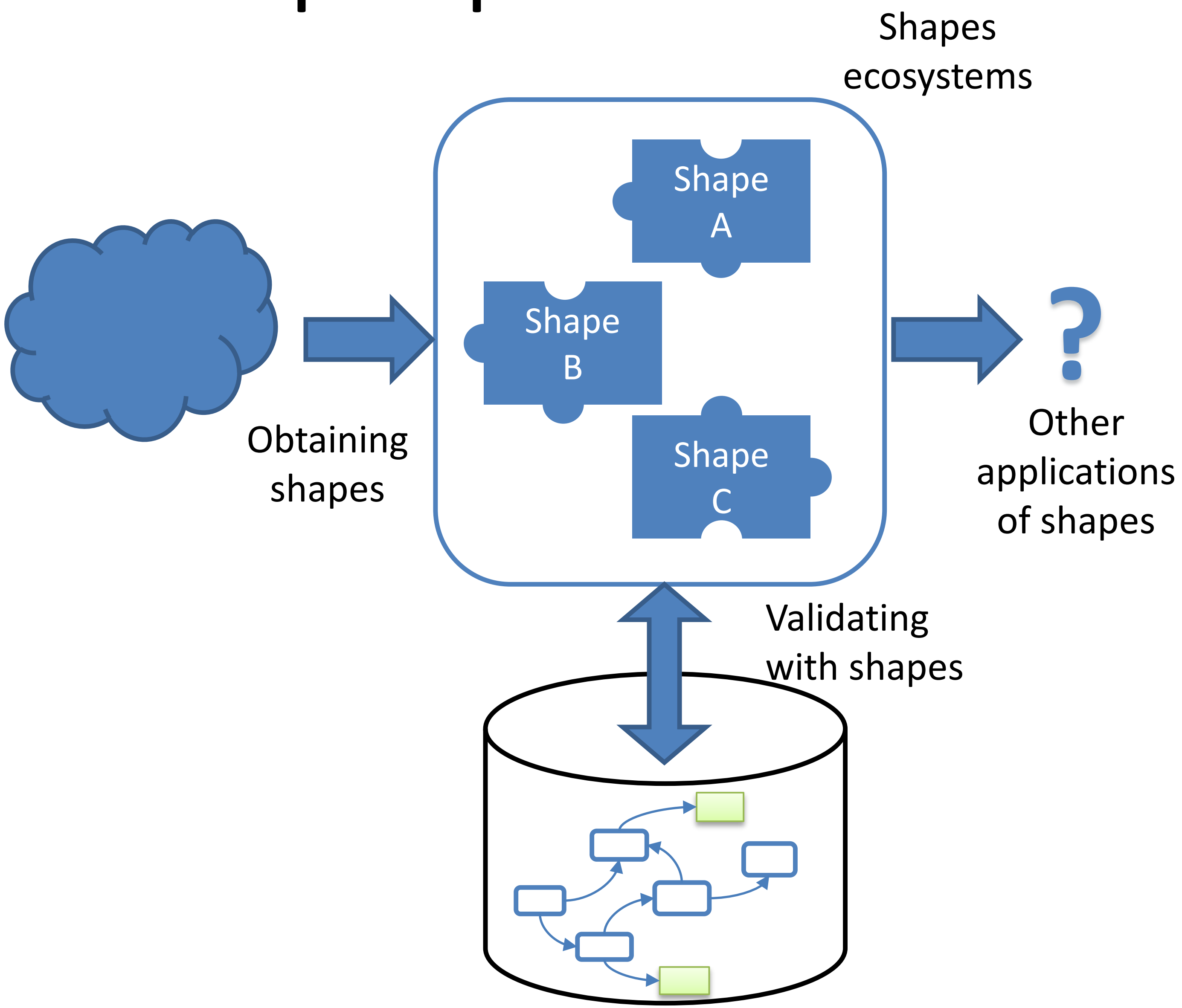
Tools: challenges and perspectives

Validating with shapes

Obtaining shapes

Other applications of shapes

Shapes ecosystems



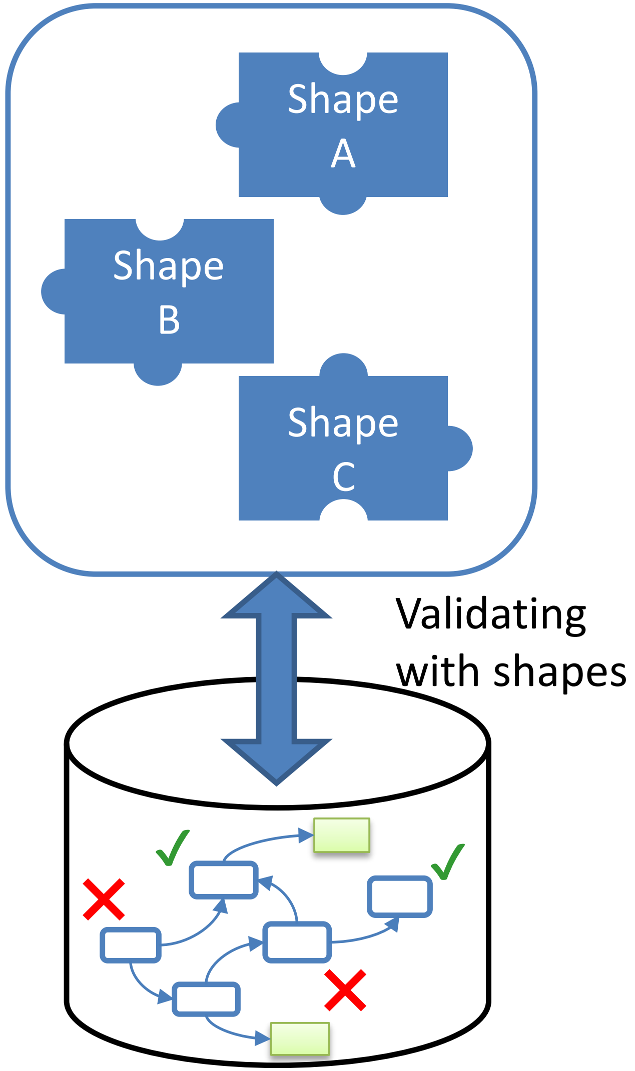
Validating with shapes

Libraries and command line validators

Online demos

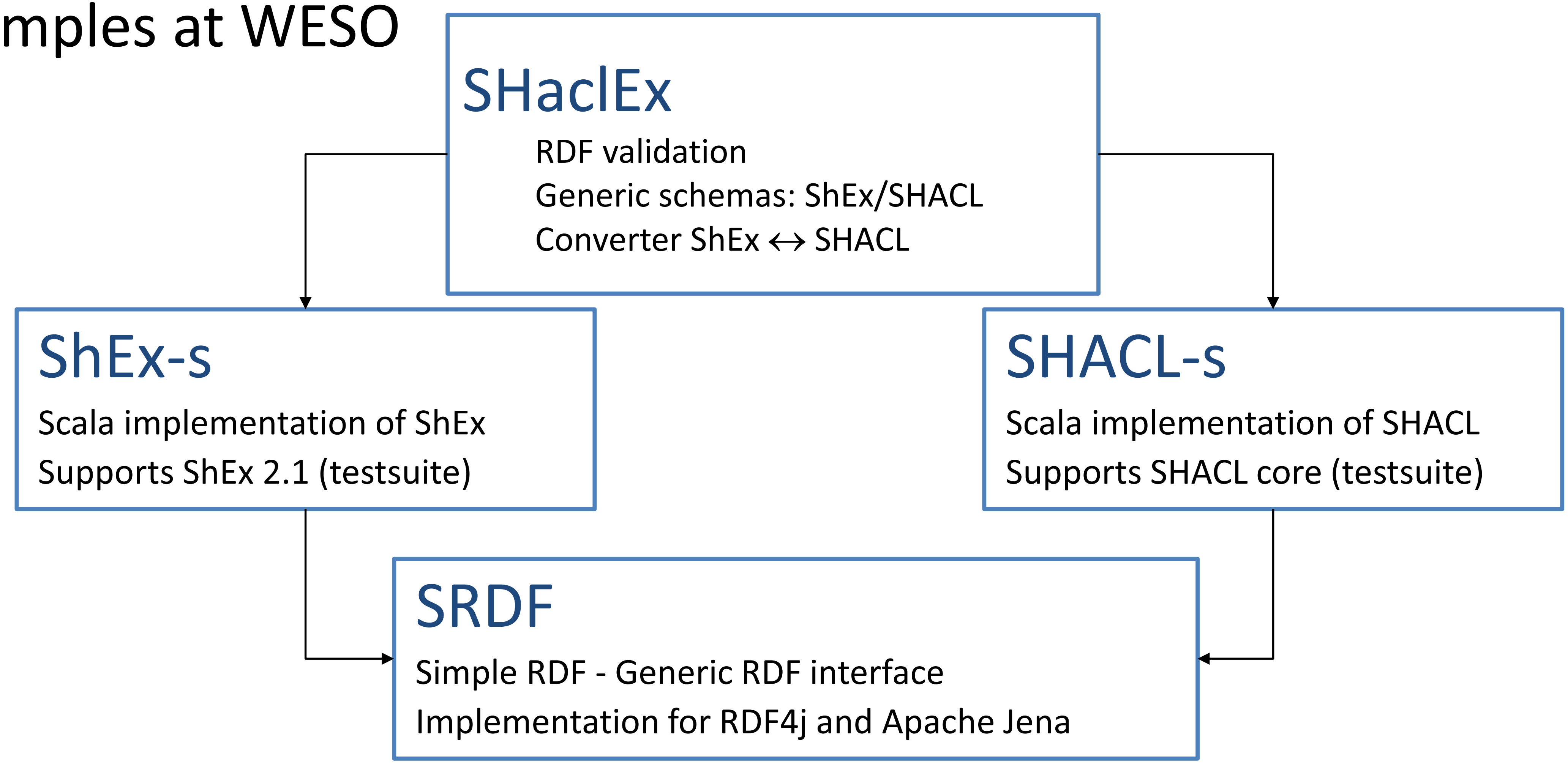
Integrated in ontology editors

Continuous integration with Shapes



Libraries and command line validators

Examples at WESO



SHaclEx

RDF validation

Generic schemas: ShEx/SHACL

Converter ShEx ↔ SHACL

ShEx-s

Scala implementation of ShEx

Supports ShEx 2.1 (testsuite)

SHACL-s

Scala implementation of SHACL

Supports SHACL core (testsuite)

SRDF

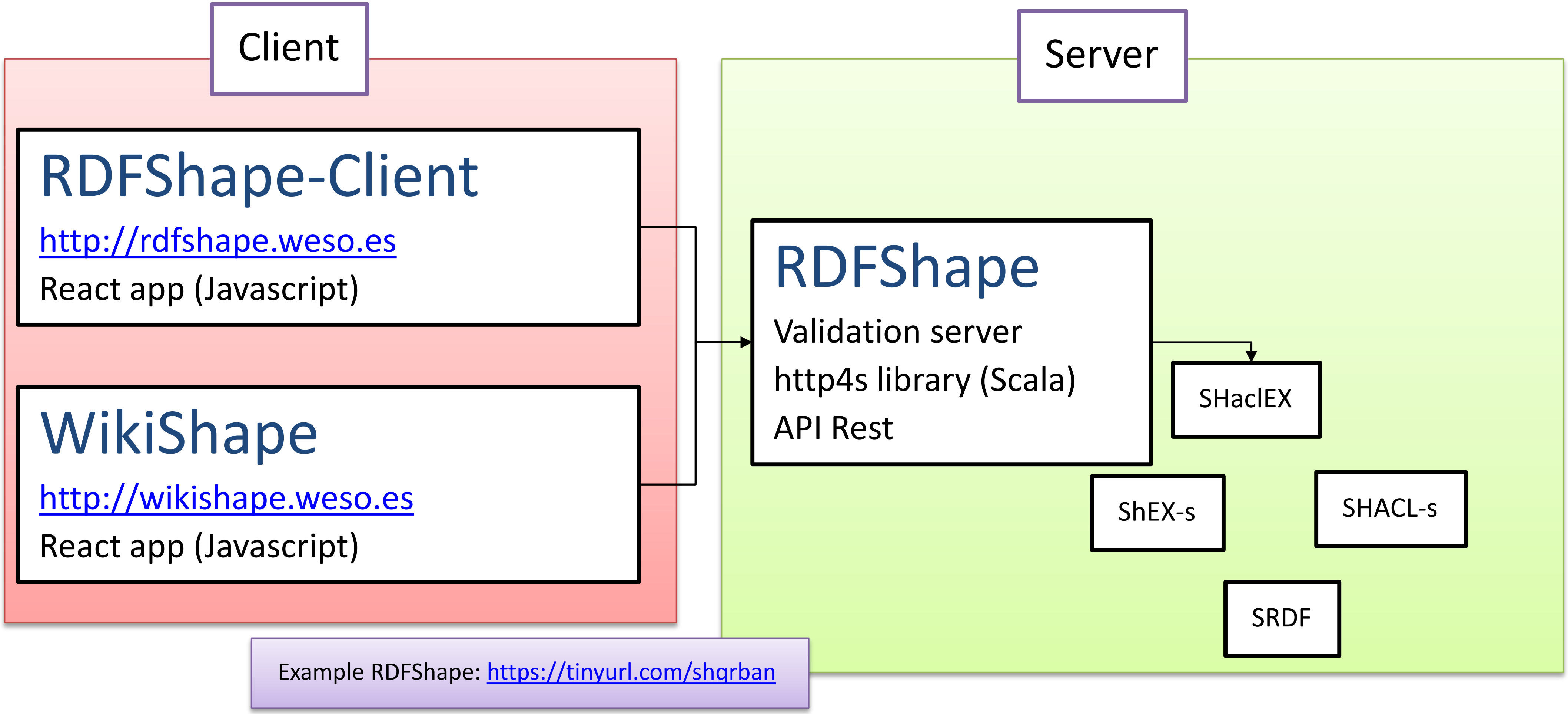
Simple RDF - Generic RDF interface

Implementation for RDF4j and Apache Jena

All libraries are available at: https://github.com/weso/

Online demos

Web Demos and playgrounds



Client

RDFShape-Client

http://rdfshape.weso.es

React app (Javascript)

WikiShape

http://wikishape.weso.es

React app (Javascript)

Server

RDFShape

Validation server

http4s library (Scala)

API Rest

SHaclEX

ShEX-s

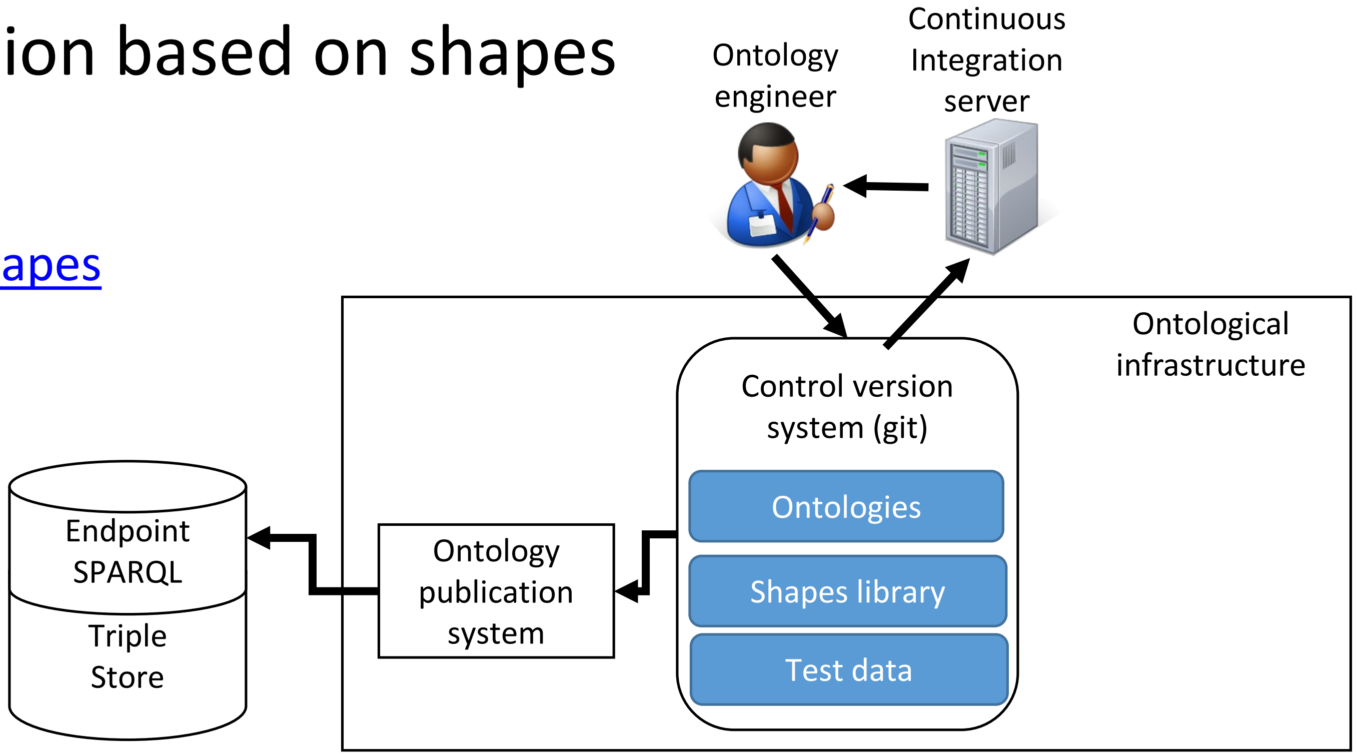
SHACL-s

SRDF

Example RDFShape: https://tinyurl.com/shqrban

Continuous integration with Shapes

- Coexistence between ontologies/shapes
  - Shapes can validate the behaviour of inference systems
  - Shapes pre- and post- inference
  - TDD and continuous integration based on shapes
- Gene Ontology Shapes:
  - https://github.com/geneontology/go-shapes


Ontology engineer
Continuous Integration server
Ontological infrastructure
Control version system (git)
Ontologies
Shapes library
Test data
Ontology publication system
Endpoint SPARQL
Triple Store

Continuous integration with Shapes

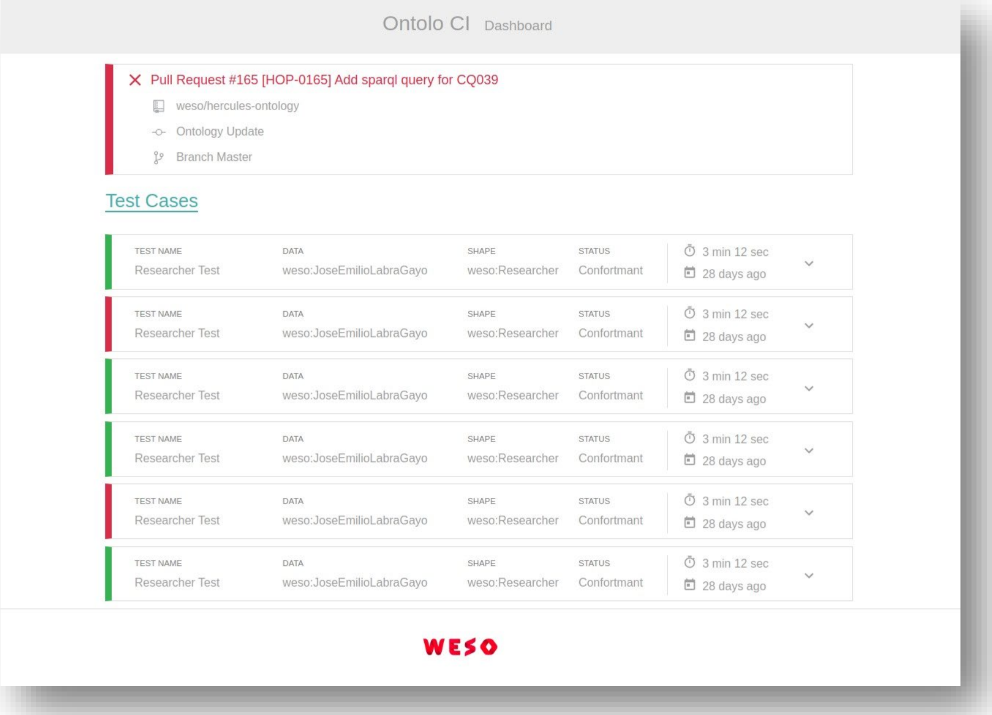
Ontolo-ci: https://www.weso.es/ontolo-ci/

Developed as part of HERCULES-Ontology

Test-Driven-Development applied to Ontologies

Input:

- Ontologies
- Shapes
- Test data
- Input shape map (SPARQL competency question)
- Expected result shape map



Ontolo CI Dashboard

✕ Pull Request #165 [HOP-0165] Add sparql query for CQ039

weso/hercules-ontology

Ontology Update

Branch Master

Test Cases

	TEST NAME	DATA	SHAPE	STATUS		
🟩	Researcher Test	weso:JoseEmilioLabraGayo	weso:Researcher	Confortmant	⏱ 3 min 12 sec	28 days ago
🟥	Researcher Test	weso:JoseEmilioLabraGayo	weso:Researcher	Confortmant	⏱ 3 min 12 sec	28 days ago
🟩	Researcher Test	weso:JoseEmilioLabraGayo	weso:Researcher	Confortmant	⏱ 3 min 12 sec	28 days ago
🟩	Researcher Test	weso:JoseEmilioLabraGayo	weso:Researcher	Confortmant	⏱ 3 min 12 sec	28 days ago
🟥	Researcher Test	weso:JoseEmilioLabraGayo	weso:Researcher	Confortmant	⏱ 3 min 12 sec	28 days ago
🟩	Researcher Test	weso:JoseEmilioLabraGayo	weso:Researcher	Confortmant	⏱ 3 min 12 sec	28 days ago

WESO

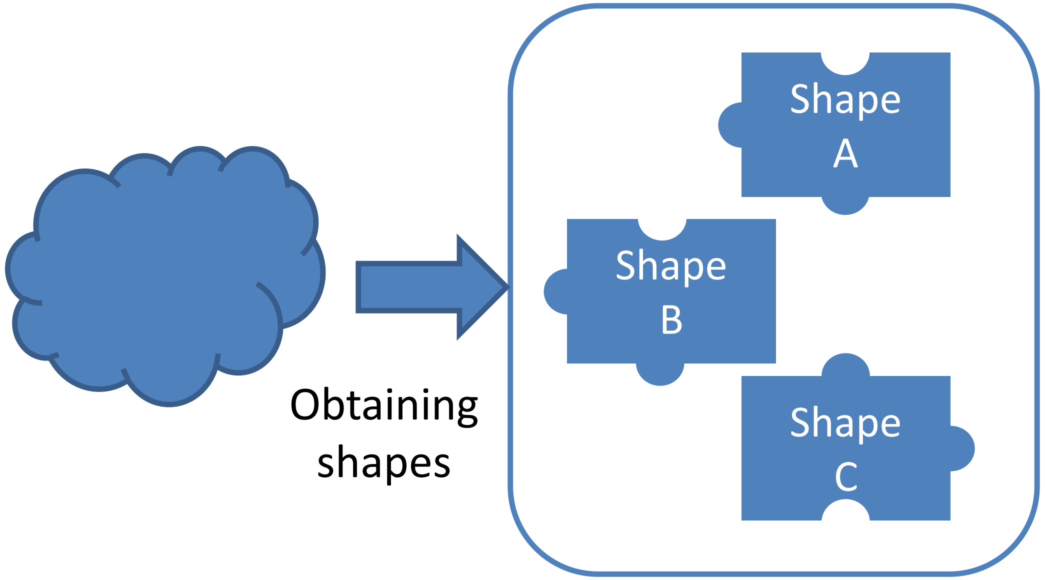
Obtaining shapes

Shapes editors

- Text-based editors
- Visual editors and visualizers

Obtaining shapes from...

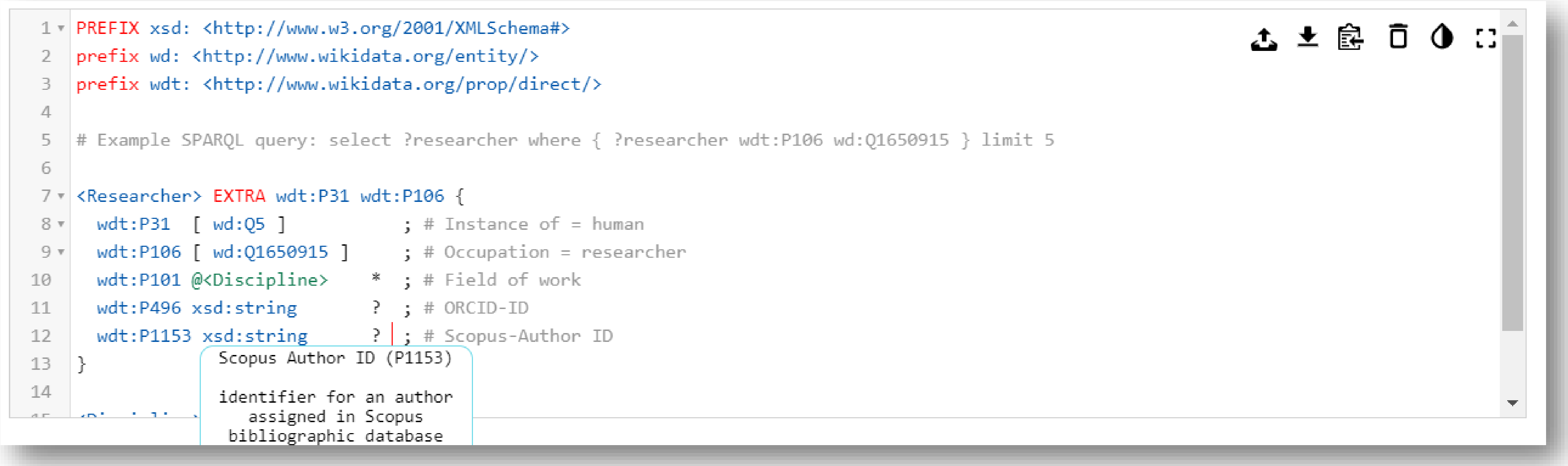
- Spreadsheets
- RDF data
- Ontologies
- Other schemas (XML Schema)



Text-based editors

YaSHE: Forked from YASGUI: http://www.weso.es/YASHE/

- Syntax highlighting
- Auto-completion



```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
prefix wd: <http://www.wikidata.org/entity/>
prefix wdt: <http://www.wikidata.org/prop/direct/>

# Example SPARQL query: select ?researcher where { ?researcher wdt:P106 wd:Q1650915 } limit 5

<Researcher> EXTRA wdt:P31 wdt:P106 {
  wdt:P31  [ wd:Q5 ]           ; # Instance of = human
  wdt:P106 [ wd:Q1650915 ]     ; # Occupation = researcher
  wdt:P101 @<Discipline>    *  ; # Field of work
  wdt:P496 xsd:string       ?  ; # ORCID-ID
  wdt:P1153 xsd:string      ?  ; # Scopus-Author ID
}

```

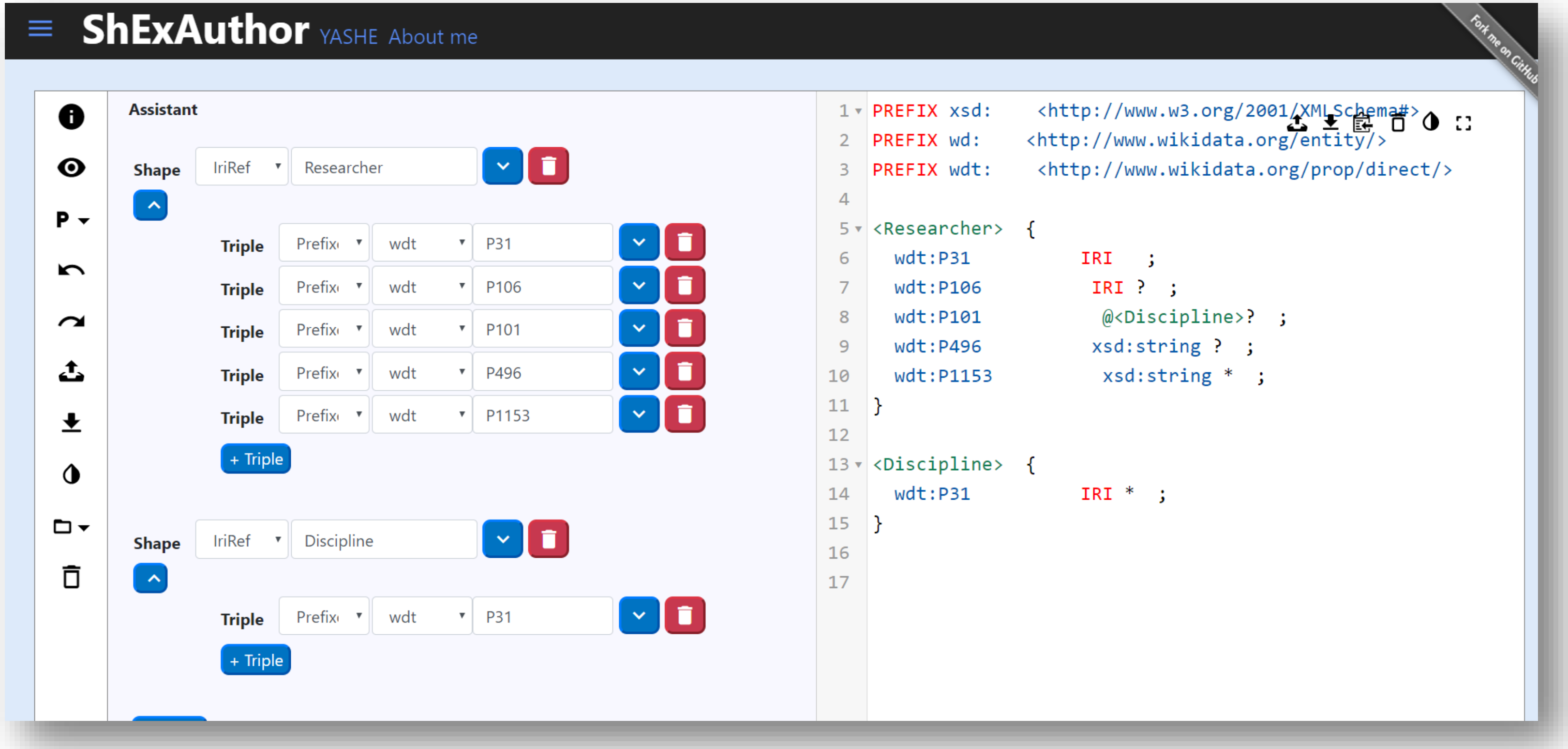
Scopus Author ID (P1153)

identifier for an author assigned in Scopus bibliographic database

Shapes author tools: ShEx Author

ShEx-Author: Inspired by Wikidata Query Service

2 column: Visual one synchronized with text based



Shape IriRef Researcher

Triple Prefix wdt P31

Triple Prefix wdt P106

Triple Prefix wdt P101

Triple Prefix wdt P496

Triple Prefix wdt P1153

Shape IriRef Discipline

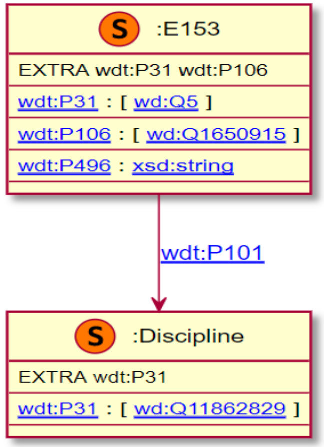
Triple Prefix wdt P31

```
PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>
PREFIX wd:    <http://www.wikidata.org/entity/>
PREFIX wdt:    <http://www.wikidata.org/prop/direct/>

<Researcher>  {
  wdt:P31          IRI   ;
  wdt:P106          IRI ?  ;
  wdt:P101          @<Discipline>?  ;
  wdt:P496          xsd:string ?  ;
  wdt:P1153          xsd:string *   ;
}

<Discipline>  {
  wdt:P31          IRI *   ;
}
```

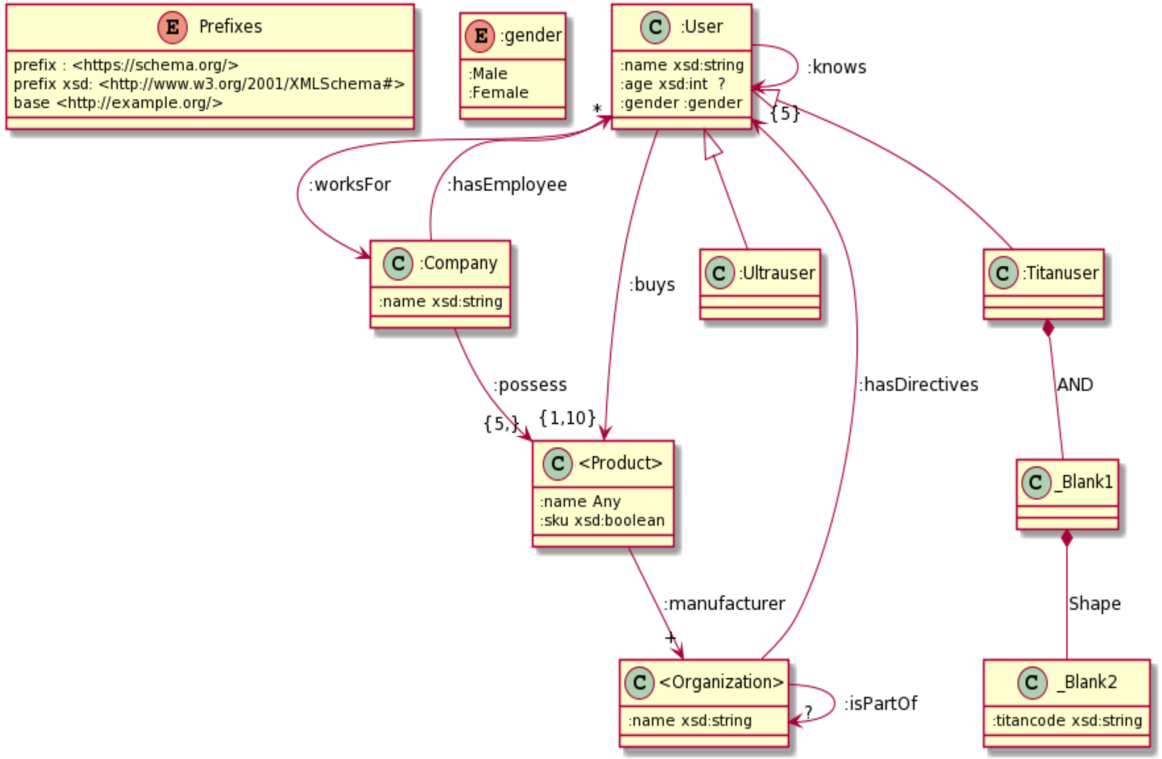
Shapes visualization

Integrated in RDFShape/Wikishape

- UMLSHaclEX UML diagrams for ShEx
- ShUMLex: Conversion to UML through XMI



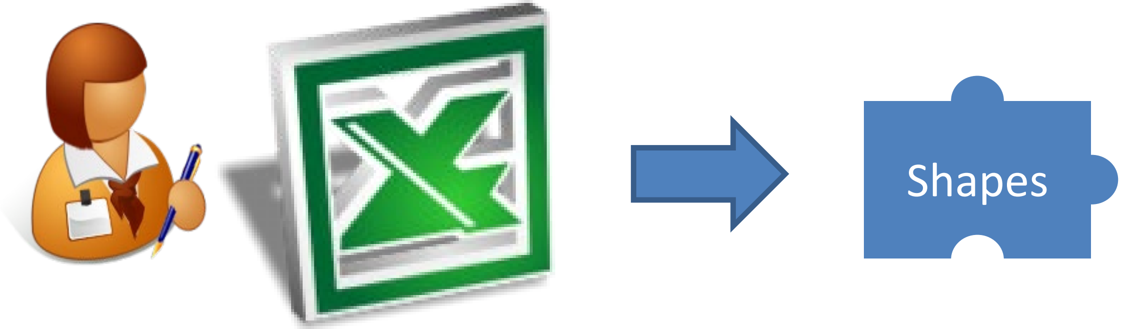
Generated by rdfshape



Shapes from spreadsheets

ShExCSV: CSV representation of Shapes

Hermes: ShExCSV processor, https://github.com/weso/hermes

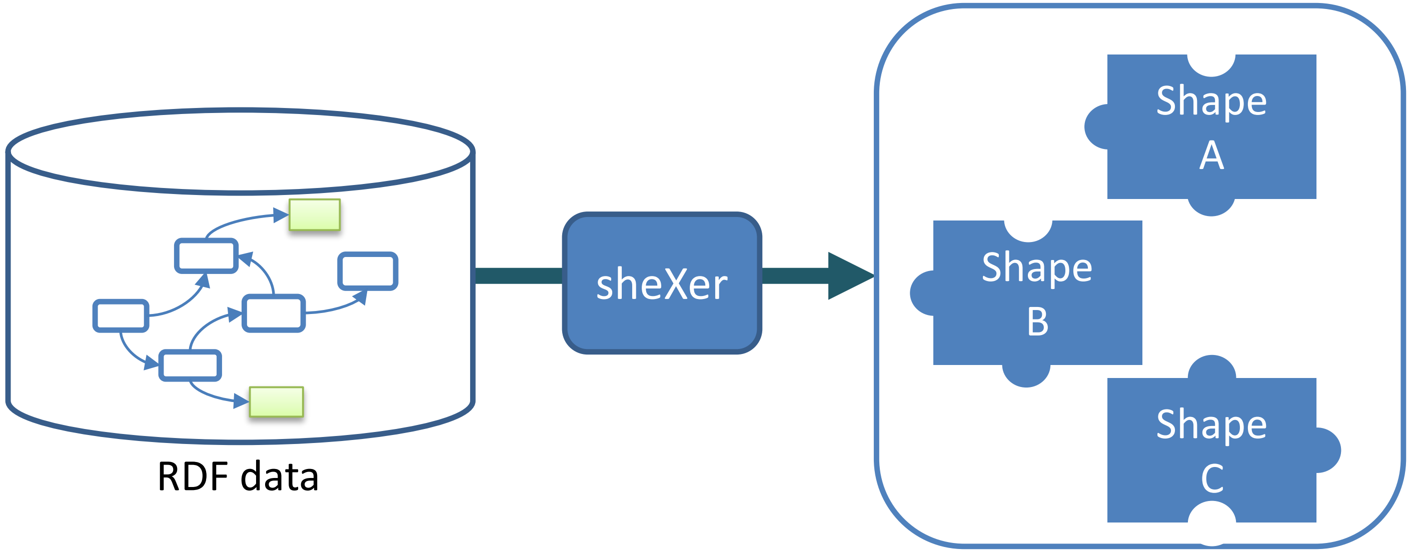


Generating Shapes from RDF data

Useful use case in practice

Some prototypes

sheXer: http://shexer.weso.es/

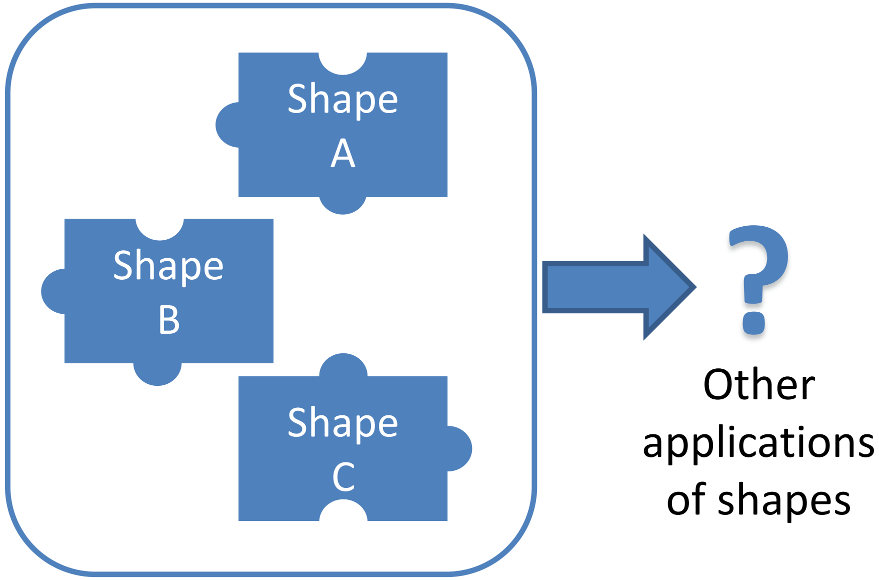


Other applications of Shapes

UIs and shapes

Generating code from Shapes

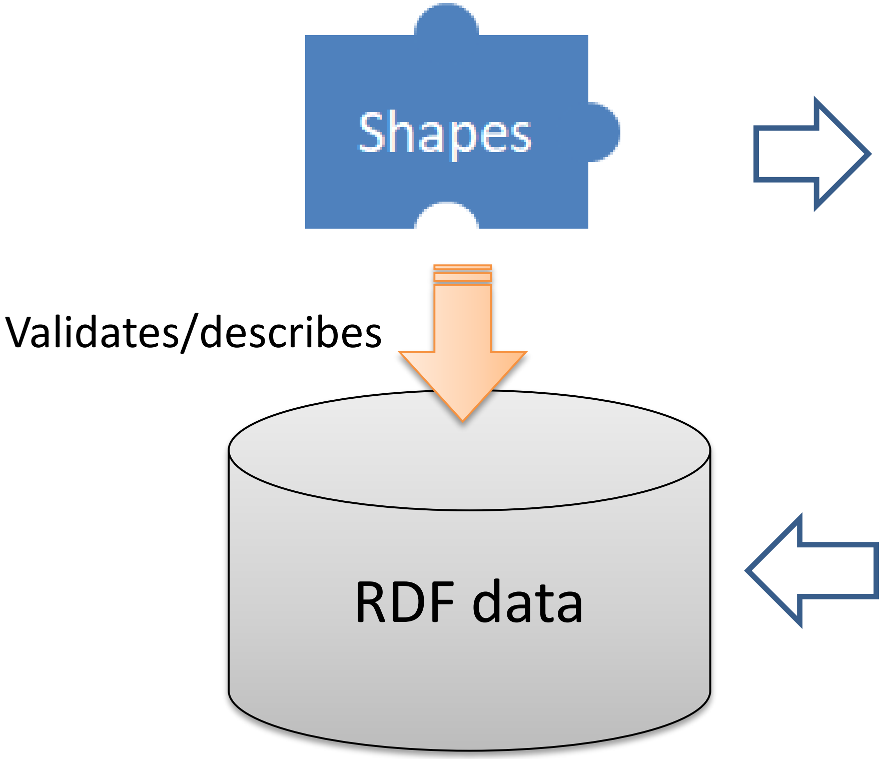
Generate subsettings

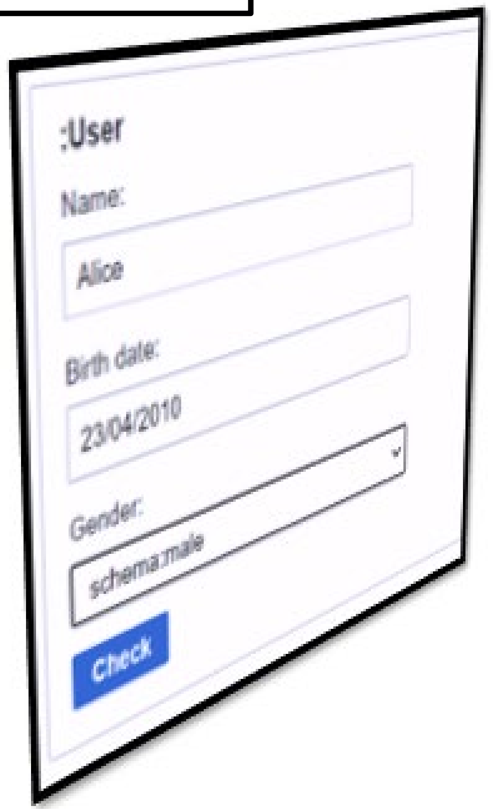


UI and shapes: ShapeForms

ShapeForms

```
prefix schema: <http://schema.org/>
prefix : <http://example.org/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix ui: <http://www.w3.org/ns/ui#>

start = @:User

:User {
 schema:name  xsd:string                    // ui:label "name" ;
 schema:birthDate xsd:dateTime              // ui:label "Birth date" ;
 schema:gender [ schema:Male schema:Female ] // ui:label "Gender" ;
}
```





:User

Name:

Alice

Birth date:

23/04/2010

Gender:

schema:male

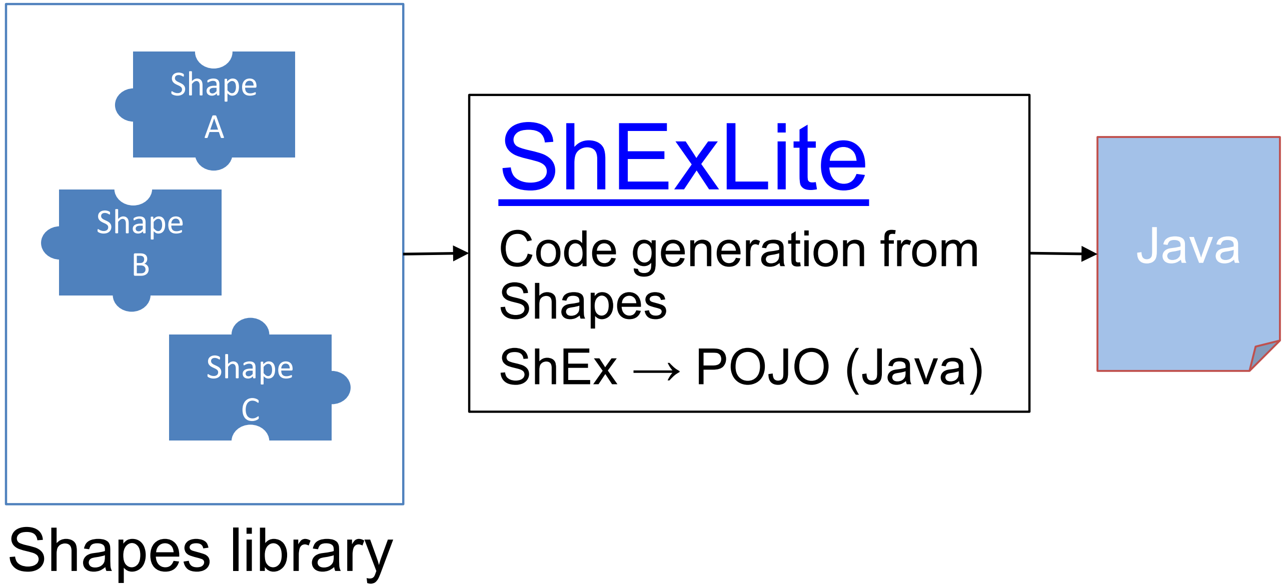
Generating code from shapes

Generate domain model from shapes

Entities (pseudo-shapes) defined with Excel (Google spreadsheets)

Shapes generation from those templates

Java code generation (POJOs) from those shapes

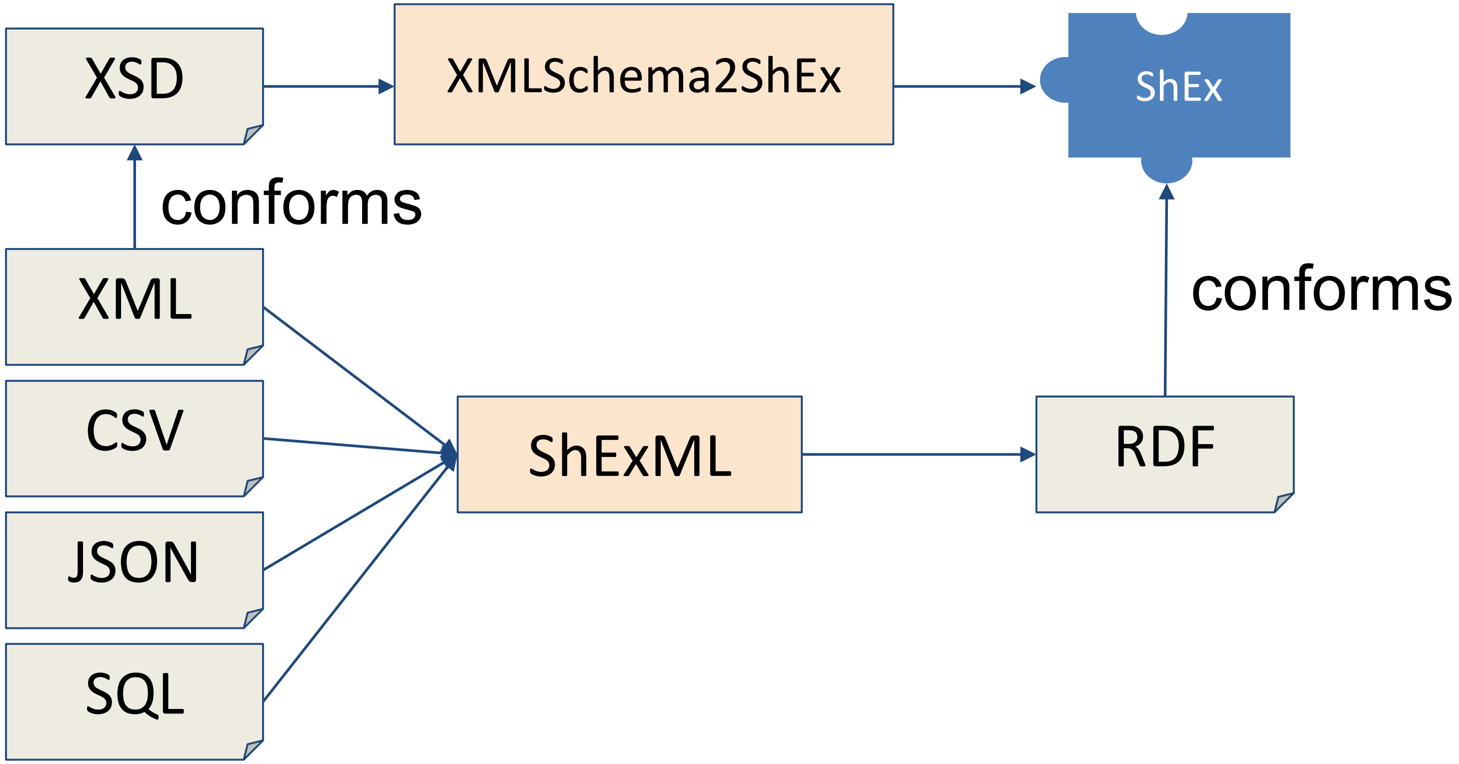


Shapes for data integration

XMLSchema2ShEx: Convert XML Schemas to shapes

ShExML: Domain specific language to convert data to RDF

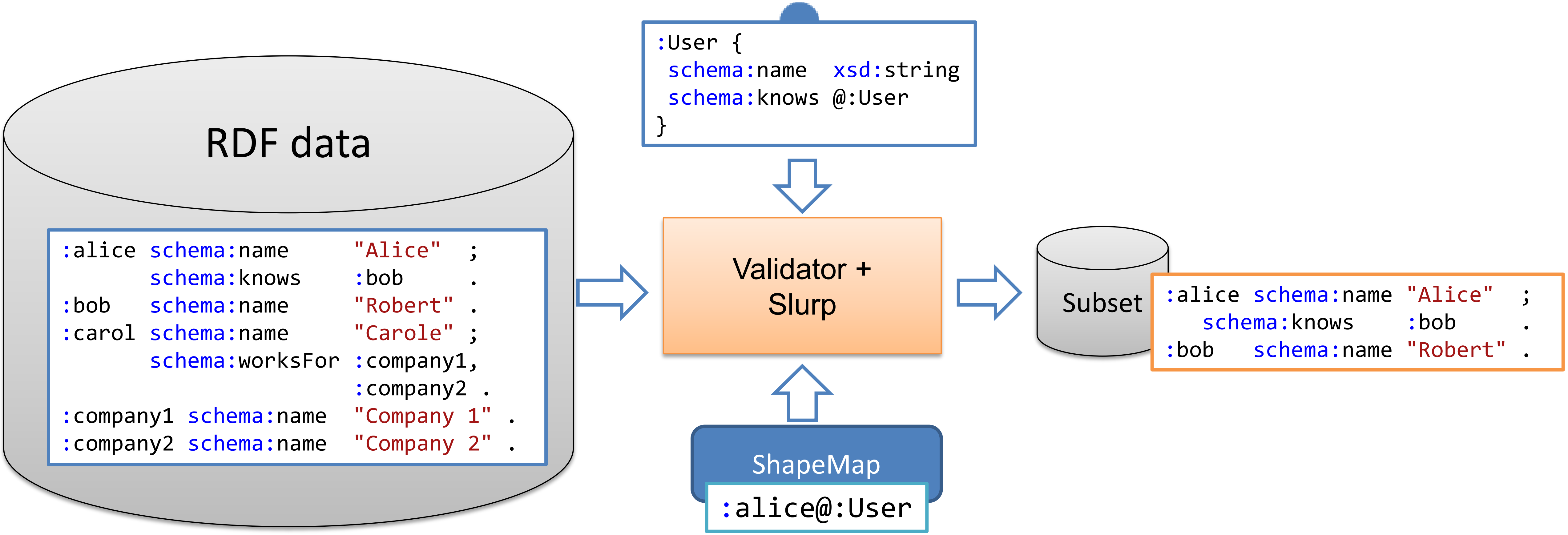
Input formats: CSV, XML, JSON, SQL



Subsetting based on Shapes

Generate subsets from ShEx

Slurp option: when validating, collect the affected nodes/triples



```
:alice schema:name      "Alice"   ;
       schema:knows     :bob      .
:bob   schema:name      "Robert"  .
:carol schema:name      "Carole"  ;
       schema:worksFor :company1,
                        :company2 .
:company1 schema:name   "Company 1" .
:company2 schema:name   "Company 2" .
```

```
:User {
 schema:name  xsd:string
 schema:knows @:User
}
```

```
:alice@:User
```

```
:alice schema:name "Alice"   ;
   schema:knows    :bob      .
:bob   schema:name "Robert"  .
```

Shapes ecosystems

- Wikidata provides a whole ShEx ecosystem
- Entity schemas can evolve and relate between each other
    - Directory: https://www.wikidata.org/wiki/Wikidata:Database_reports/EntitySchema_directory
    - Different schemas for the same entities?
        - Some schemas stress some aspects while others stress others
    - Evolution of schemas
    - Searching entity schemas

Conclusions

- ShEx and SHACL have had a great level of adoption
- They can be extended for other types of Knowledge graphs
  - Property graphs and wikibase graphs
- Towards shapes ecosystems
  - New tools and challenges

Backup slides

OWL: Ontologies

Declare classes, properties and entities in a domain

Focus on the domain model

Knowledge representation

Ontologies enable reasoning and classification

Open World Assumption

We assume there are a lot of knowledge we may not have yet

Rules

Declare conditions IF….THEN….

Focus on the domain model

Knowledge representation

Premises/conclusions

Open/Closed World Assumption

Both assumptions can be used depending on the domain