Validando datos RDF

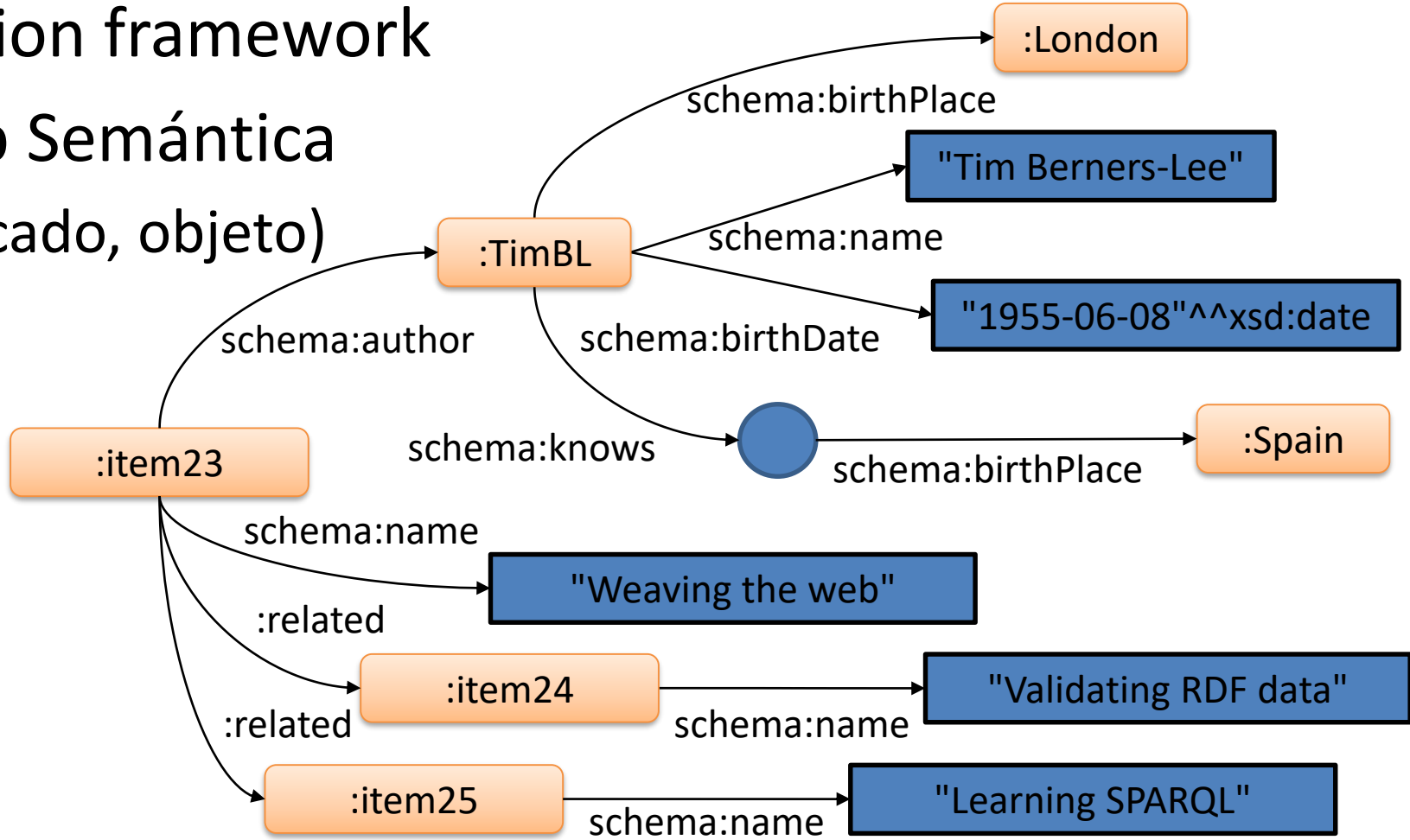
Jose Emilio Labra Gayo
WESO Research group
University of Oviedo, Spain

Grafos RDF (repaso)

RDF = resource description framework

Lingua franca de la Web Semántica

- Tripletas: (sujeto, predicado, objeto)
- Predicados = URIs
  - Interoperabilidad



Ecosistema RDF

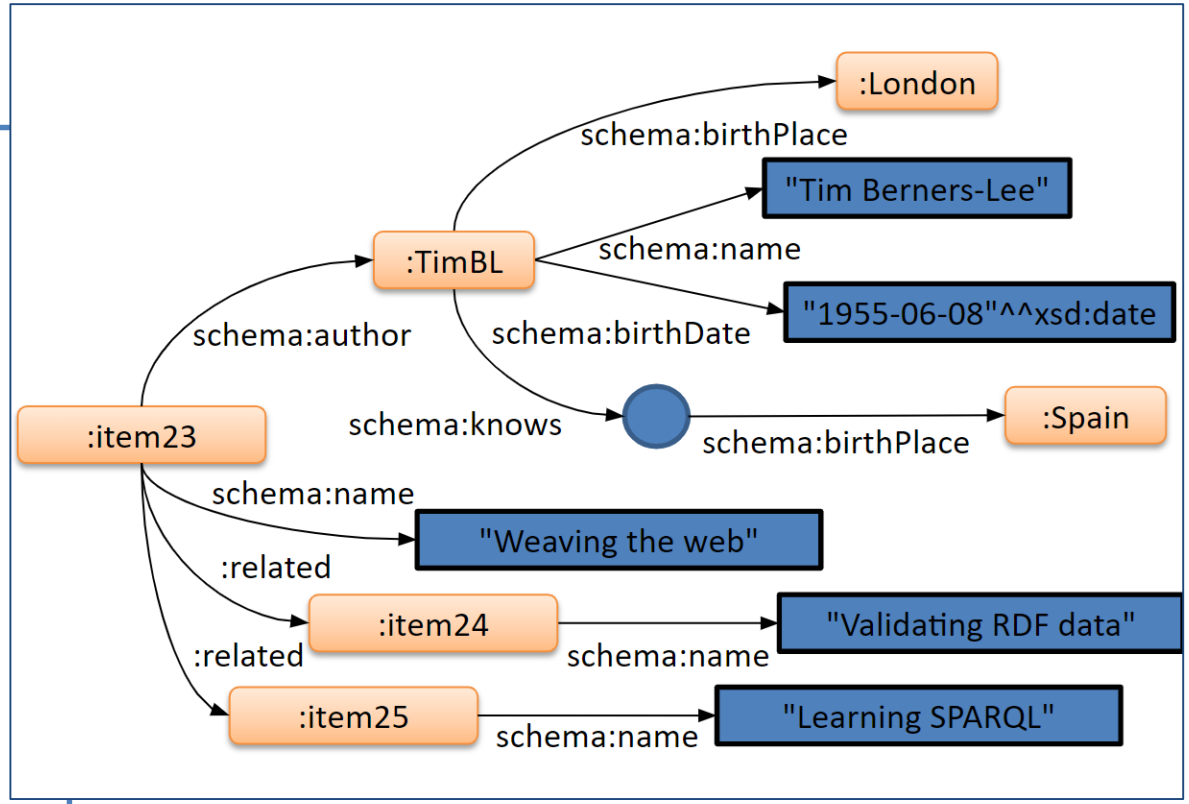
Un modelo de datos, varias sintaxis: Turtle, N-Triples, JSON-LD, …

Vocabularios: RDF Schema, OWL, SKOS, etc.

Lenguaje de consulta SPARQL

Turtle

```
prefix :       <http://example.org/>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix schema: <http://schema.org/>

:item23 schema:name       "Weaving the web"       ;
        schema:author     :timbl                  ;
        schema:related    :item24, :item25        .
:timbl  schema:name       "Tim Berners-Lee"       ;
        schema:birthDate  "1955-06-08"^^xsd:date  ;
        schema:birthPlace :london                 ;
        schema:knows      _:1                     .
_:1     schema:birthPlace :Spain                  .
:item24 schema:title      "Validating RDF data"   .
:item25 schema:title      "Learning SPARQL"       .
```



https://rdfshape.weso.es/link/16481303796

RDF, las partes buenas...

RDF como lenguaje de integración

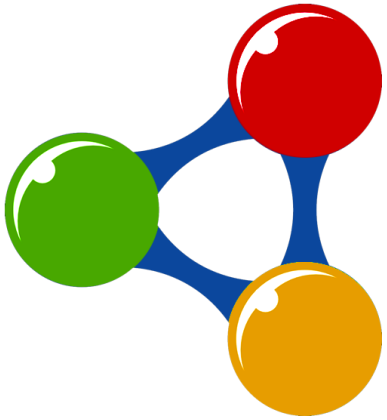
- RDF es la *lingua franca* de la web semántica y datos enlazados
- La base de la representación del conocimiento

Flexibilidad de RDF

- Los datos pueden adaptarse a multiples entornos
- Reutilizable por defecto

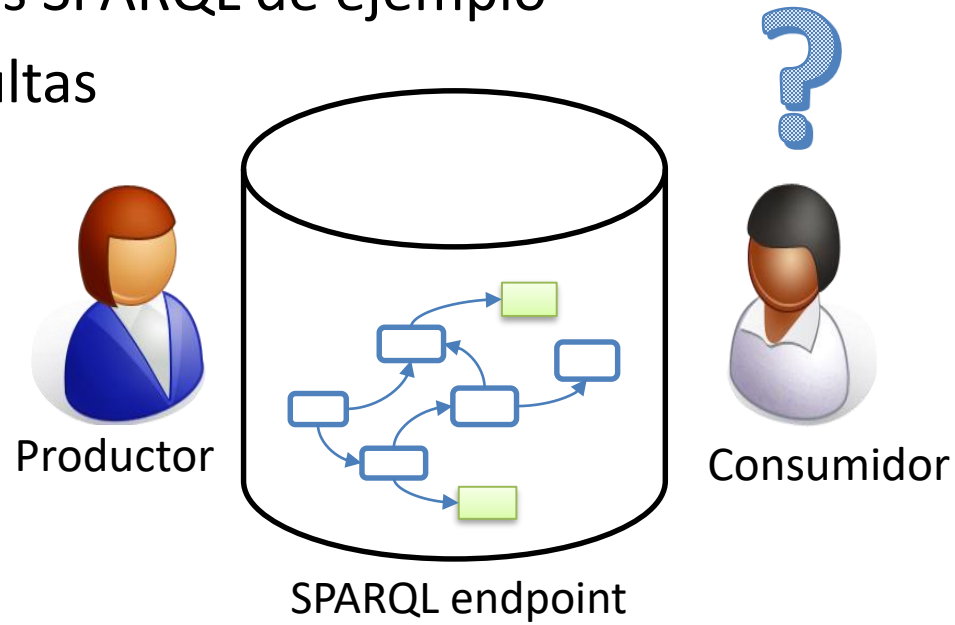
Ecosistema de herramientas RDF

- Almacenes de datos RDF & SPARQL
- Varias serializaciones: Turtle, JSON-LD, RDF/XML...
- Puede integrarse en HTML (Microdata/RDFa)



RDF, las otras partes

- Consumir & producir RDF
  - Describir y validar contenido RDF
  - Los SPARQL endpoints no suelen estar bien documentados
    - Documentación habitual = conjunto de consultas SPARQL de ejemplo
    - Difícil saber por dónde empezar a realizer consultas



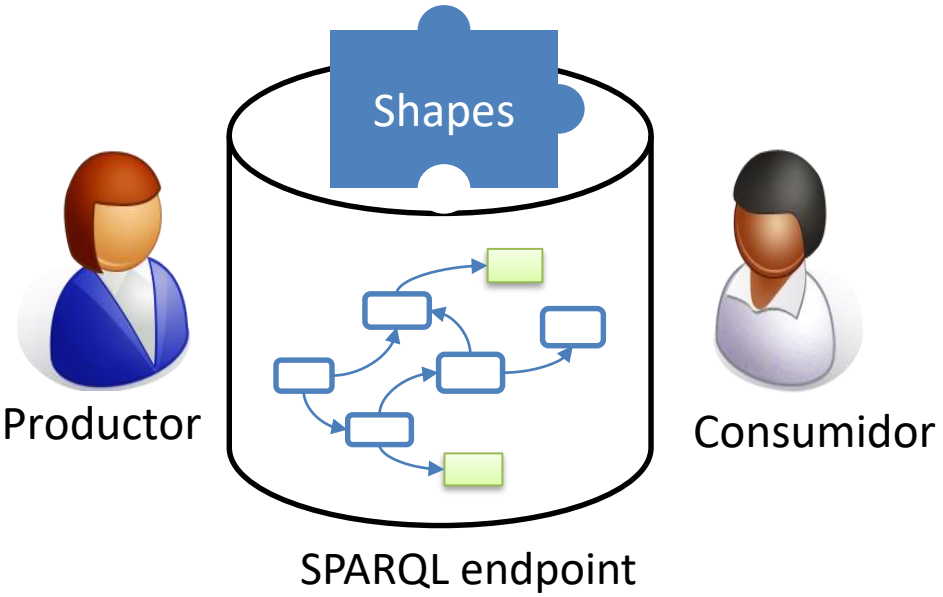
¿Porqué describir y validar RDF?

Para los productores

- Los desarrolladores pueden comprender los contenidos que van a producir
- Pueden asegurarse de que producen la estructura esperada
- Anunciar y documentar dicha estructura
- Generar *interfaces*

Para consumidores

- Comprender los contenidos
- Verificar la estructura antes de procesarla
- Generar consultas y optimización



Tecnologías similares

Tecnología	Esquemas
Bases de datos relacionales	DDL
XML	DTD, XML Schema, RelaxNG, Schematron
Json	Json Schema
RDF	?

Rellenar ese hueco

¿Esquemas para RDF?

RDF no impone un esquema pero…

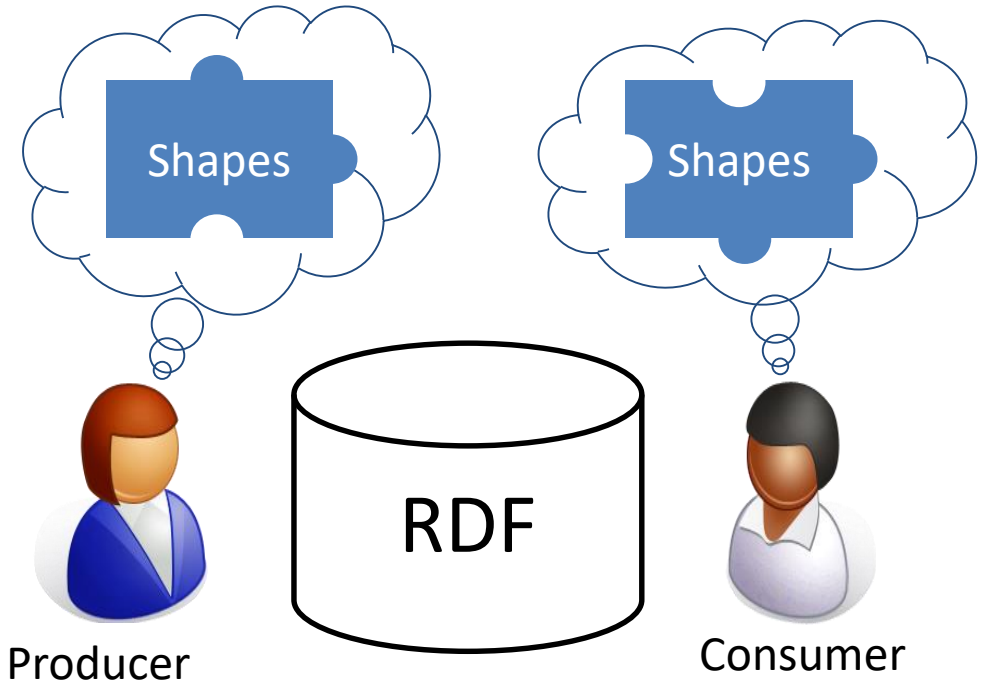
En la práctica, hay **esquemas implícitos**

Son asumidos

Las shapes hacen explícitos los esquemas

Manejar datos malformados o incompletos

Evitar programación defensiva



Enfocar discusiones en lo que es importante

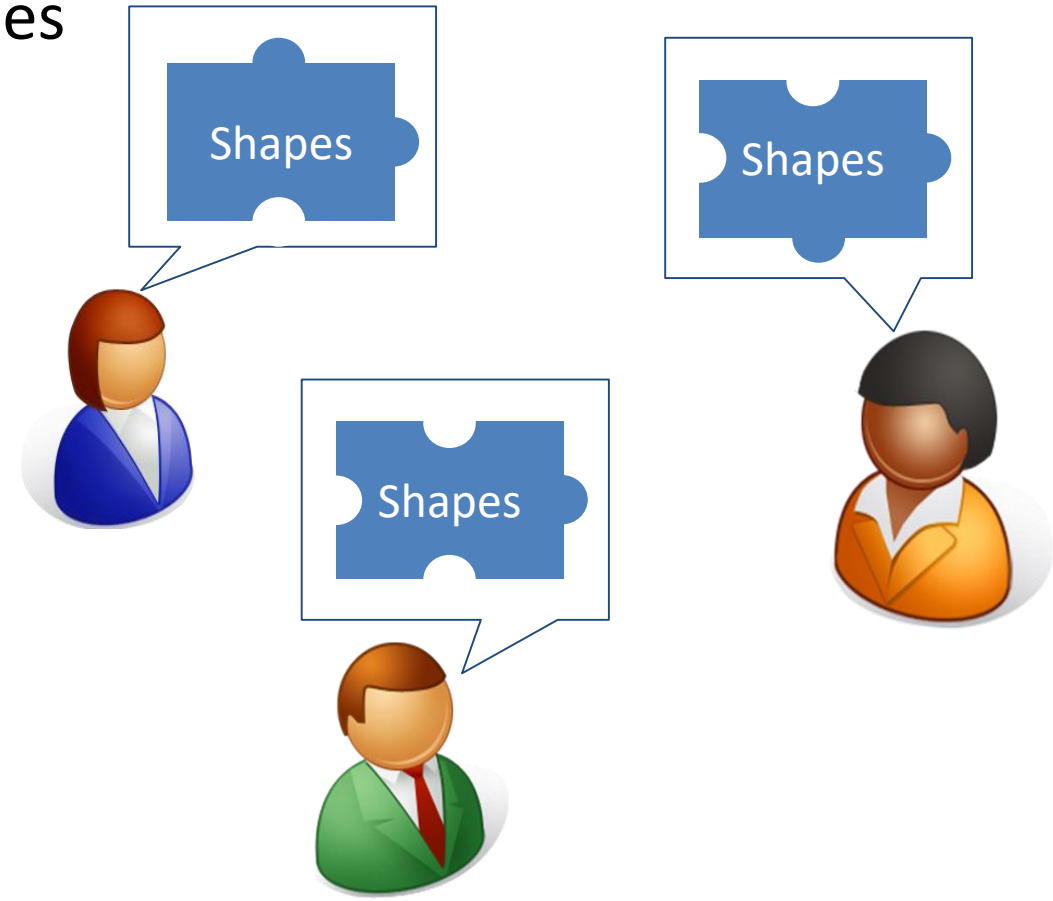
Motivación inicial: modelos de datos clínicos (FHIR)

Modelos de contenido distribuidos y extensibles

- Distribuidos por lugar
- Distribuidos por autoridad
- Extensibles

Definir esquemas compartidos

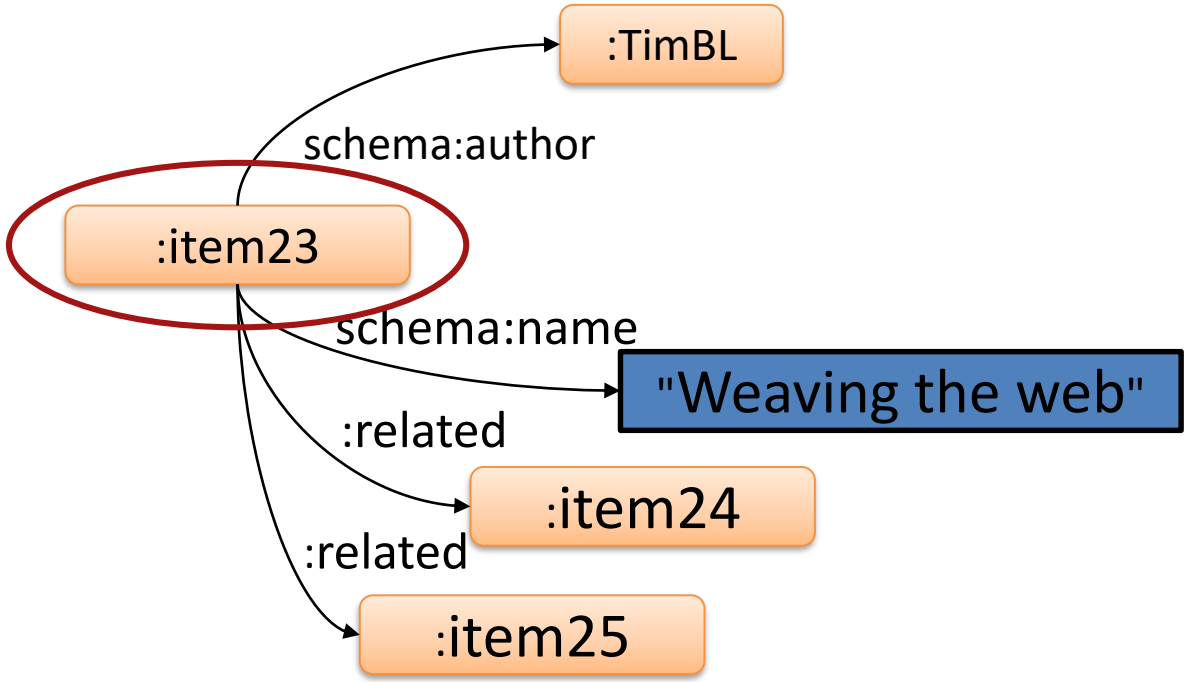
- Comprensibles por expertos de dominio
- …y procesables por máquinas



¿Qué es una *shape*?

Una *shape* describe

- La forma de un nodo (node constraint)
- Arcos que entran y salen de un nodo
- Valores asociados con dichos arcos



RDF Node

```
:item23 schema:author :timbl                ;
        schema:name   "Weaving the web"     ;
        :related      :item24  , :item25 .
```

ShEx

```
:Book IRI and {
 schema:author @:Author    + ;
 schema:name   xsd:string    ;
 :related      @:Book      *
}
```

https://rdfshape.weso.es/link/16478907876

ShEx y SHACL

2013 RDF Validation Workshop

- Conclusiones del workshop:
  - *Hay una necesidad para un lenguaje para validación de RDF concise y de alto nivel*
- ShEx se propuso inicialmente (v 1.0)

2014 W3C Data Shapes WG chartered

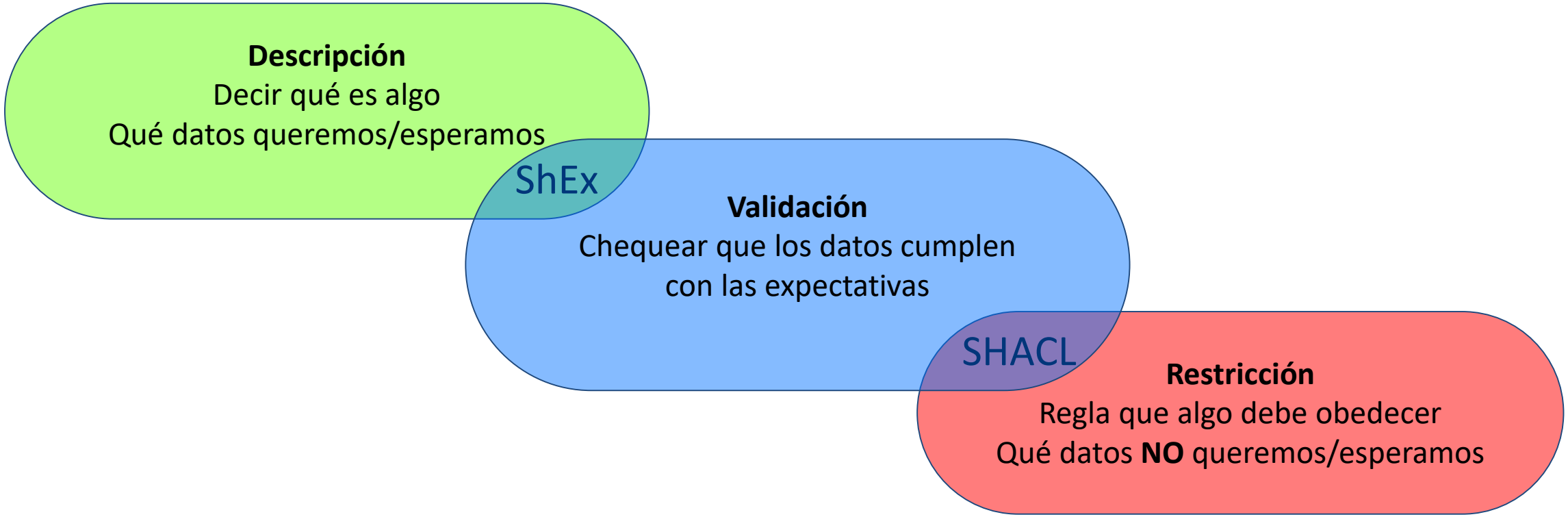
2017 SHACL aceptado como recomendación W3C

2017 ShEx 2.0 liberado como borrador W3C Community group

2019 ShEx adoptado en Wikidata

2023 IEEE Shape Expressions Schema iniciado

Descripción - Validación - Restricciones



Introducción a Shape Expressions

WESO

Pequeña introducción a ShEx

ShEx (Shape Expressions Language)

Conciso y legible por humanos

Syntaxis similar a SPARQL/Turtle

Semántica inspirada por expresiones regulares y RelaxNG

2 syntaxis: Compacta y RDF/JSON-LD

Información oficial: http://shex.io

Semántica: http://shex.io/shex-semantics/, introducción: http://shex.io/shex-primer

Implementaciones y playgrounds de ShEx

Implementaciones:

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

Demos Online & playgrounds

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

Ejemplo Simple

Declaraciones de prefijos como Turtle/SPARQL

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

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

Nodos que cumplen la shape `<User>` deben:

- Ser IRIs
- Tener exactamente un `schema:name` con un valor de tipo `xsd:string`
- Tener cero o más `schema:knows` cuyos valores cumplan con `<User>`

Validación RDF usando ShEx

Esquema

```
<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

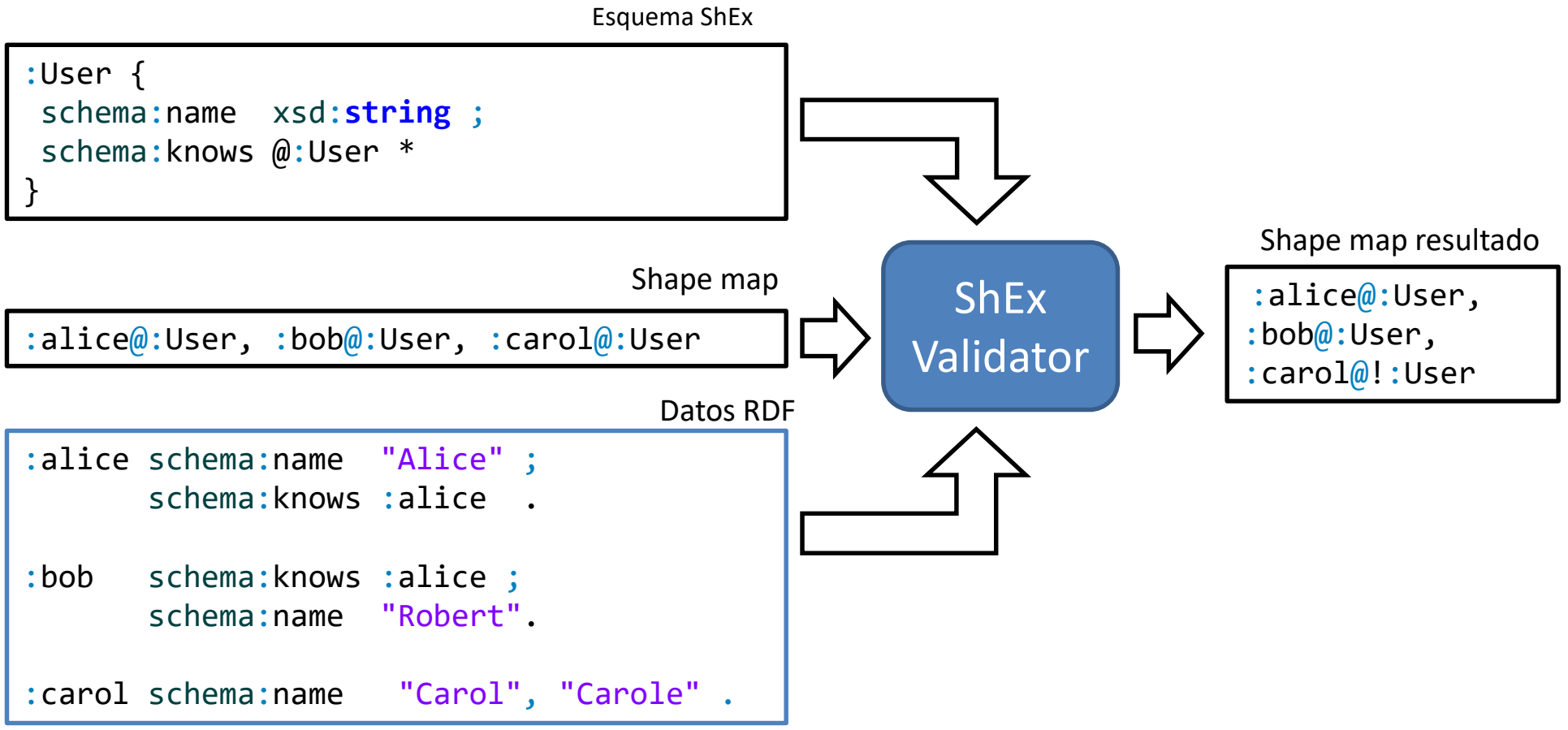
Data

```
: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" .
```

Proceso de validación

Entrada: Datos RDF, Esquema ShEx, Shape map

Salida: Shape map de resultado



Restricciones de Nodos (Node constraints)

Describen la forma de un nodo

```
:Book {
 :name          xsd:string                          ;
 :datePublished xsd:date                            ;
 :numberOfPages MinInclusive 1                      ;
 :author        @:Person                            ;
 :genre         [ :Action :Comedy :NonFiction ]     ;
 :isbn          /isbn:[0-9X]{10}/                   ;
 :publisher     IRI                                 ;
 :audio         .                                   ;
 :maintainer    @:Person OR @:Organization
}

:Person {}
:Organization {}
```

```
:item23
 :name          "Weaving the Web"                   ;
 :datePublished "2012-03-05"^^xsd:date              ;
 :numberOfPages 272                                 ;
 :author        :timbl                              ;
 :genre         :NonFiction                         ;
 :isbn          "isbn:006251587X"                   ;
 :publisher     <http://www.harpercollins.com/>     ;
 :audio         <http://audio.com/item23>           ;
 :maintainer    :alice                              .
```

Try it: (RDFShape)

Cardinalidades

Inspiradas por expresiones regulares: +, ?, *, {m,n}

Por defecto {1,1}

```
:Book {
 :name           xsd:string          ;
 :numberOfPages  xsd:integer  ?      ;
 :author         @:Person     +      ;
 :publisher      IRI          ?      ;
 :maintainer     @:Person     {1,3}  ;
 :related        @:Book       *
}
:Person {}
:Organization {}
```

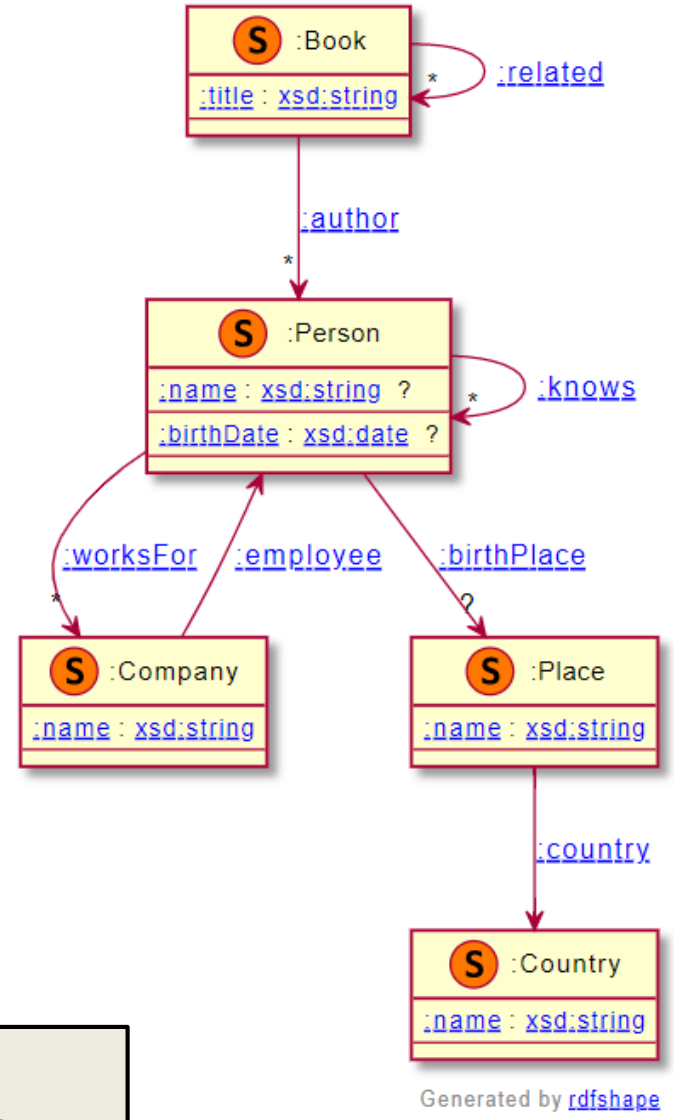
```
:item23
  :name           "Weaving the Web"       ;
  :numberOfPages  272                     ;
  :author         :timbl, :markFischetti  ;
  :maintainer     :alice,:bob             .
```

Try it: RDFShape

Esquemas recursivos

Se permiten esquemas cíclicos (recursivos)

```
:Book {
  :title   xsd:string   ;
  :author  @:Person    * ;
  :related @:Book      * ;
}
:Person {
  :name       xsd:string ? ;
  :birthDate  xsd:date   ? ;
  :birthPlace @:Place    ? ;
  :knows      @:Person   * ;
  :worksFor   @:Company  * ;
}
:Place {
  :name       xsd:string ;
  :country    @:Country  ;
}
:Country {
  :name       xsd:string
}
:Company {
  :name       xsd:string
  :employee   @:Person
}
```



Try it: RDFShape

Modelos de contenido Abiertos/Cerrados

La semántica de RDF asume contenido abierto (en general)

Shape expressions son abiertas por defecto

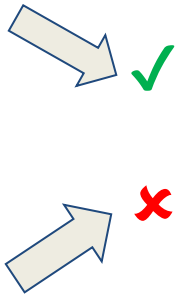
Facilita la extensibilidad

Pero…algunos casos de uso requieren modelos de contenido cerrados

Ejemplo avisar sobre expresiones *extrañas*

```
:Person {
 :name  xsd:string   ;
 :knows .          *
}
```

```
:Person CLOSED {
 :name  xsd:string   ;
 :knows .          *
}
```



```
:frank :name "Frank" ;
       :knows :alice, :bob ;
       :email <mailto:frank@e.com> .
```

Try it: RDFShape

Propiedades abiertas/cerradas

Los valores de propiedades son cerrados por defecto (*closed properties*)

```
:Book {
 :code /isbn:[0-9X]{10}/ ;
}
```

✓
```
:item23  :code "isbn:006251587X" .
```

✘
```
:item23  :code 23                .
```

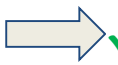
Las propiedades pueden repetirse

```
:Book {
  :code  /isbn:[0-9X]{10}/ ;
  :code  /isbn:[0-9]{13}/
}
```

✓
```
:item23 :code "isbn:006251587X"    ,
        :code "isbn:9780062515872" .
```

EXTRA declara propiedades como abiertas

```
:Book EXTRA :code {
 :code /isbn:[0-9X]{10}/ ;
}
```



✓
```
:item23 :code "isbn:006251587X" ,
        :code 23                .
```

Try it: RDFShape

Expresiones de tripletas

Expresiones regulares sin orden: *Regular bag expressions*

```
:Person {
  (:name  xsd:string |
   :firstName xsd:string + ;
   :lastName xsd:string
  ) ;
}
```

```
name |
firstName + ; lastName
```

```
:alice :name      "Alice Cooper" .

:bob   :firstName "Robert"       ;
       :lastName  "Smith"        .

:carol :firstName "Carol"        ;
       :lastName  "King"         ;
       :firstName "Carole"       .
```

```
:dave  :firstName "Dave"         ;
       :name      "Dave Navarro" .
```

Try it: RDFShape

Operadores lógicos

Se pueden combinar Shape Expressions con AND, OR, NOT

Algunas restricciones al uso de NOT combinado con recursividad

```
:Book {
 :name   xsd:string ;
 :author @:Person OR @:Organization ;
}

:AudioBook @:Book AND {
 :name          MaxLength 20 ;
 :readBy        @:Person     ;
} AND NOT {
 :numberOfPages . +
}

:Person {}
:Organization {}
```

```
:item24 :name          "Weaving the Web" ;
        :author        :timbl            ;
        :readBy        :timbl            .
```

```
:item23 :name          "Weaving the Web" ;
        :author        :timbl            ;
        :numberOfPages 272               ;
        :readBy        :timbl            .
```

Try it: RDFShape

Importar esquemas

La declaración `import` permite utilizar esquemas externos

http://validatingrdf.com/tutorial/examples/book.shex

```
:Book {
  :title   xsd:string   ;
}
:Person {
  :name        xsd:string ? ;
}
```

```
import <https://www.validatingrdf.com/examples/book.shex>

:AudioBook @:Book AND {
 :title          MaxLength 20 ;
 :readBy         @:Person     ;
}
```

```
:item24 :name          "Weaving the Web" ;
        :author        :timbl            ;
        :readBy        :timbl            .
```

Try it: RDFShape

Modelo de herencia para ShEx

`extends` permite reutilizar shapes existentes añadiendo contenido nuevo. Maneja propiedades y shapes cerradas

```
:Book {
 :name    xsd:string ;
 :author  @:Person   ;
 :code    /isbn:[0-9]{13}/   ;
 :code    /isbn:[0-9X]{10}/
}

:LibraryBook extends @:Book {
 :code    /internal:[0-9]*/ ;
}
```

```
:item23 :name        "Weaving the Web"     ;
        :author      :timbl                ;
        :code        "isbn:006251587X"     ;
        :code        "isbn:9780062515872" ;
        :code        "internal:234"        .
```

Otras características

Herencia múltiple

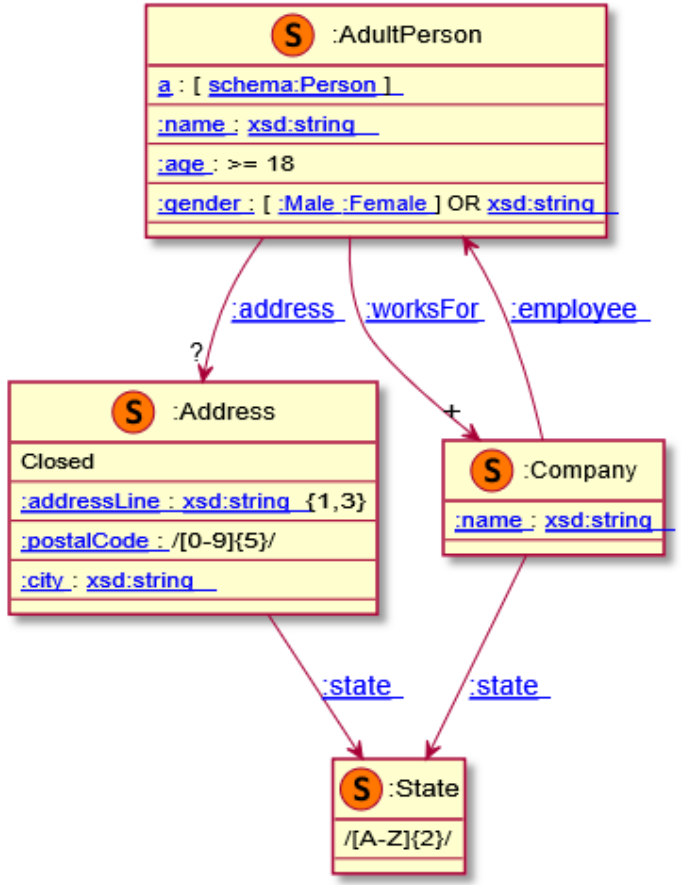
Shapes abstractas

Try it: RDFShape

Ejemplo con más caractérísticas

```
: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

Otras características de ShEx

Anotaciones procesables por la máquina

Rangos de conjuntos de valores

Valores de etiquetas de idioma

Acciones semánticas

Expresiones con nombre

Shapes anidadas

Shape maps

. . .

Introducción a SHACL

SHACL

SHACL (Shapes Constraint Language)

W3C recommendation:

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

Vocabulario RDF

2 partes: SHACL-Core, SHACL-SPARQL

Implementaciones SHACL

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

RDFShape soporta:
- SHaclEX (SHACL-s)
- JenaSHACL
- SHACL TQ (SHACL TopBraid API)

Ejemplo Básico

```
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

Mismo ejemplo con nodos anónimos

```
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" .
```

☹ ☹

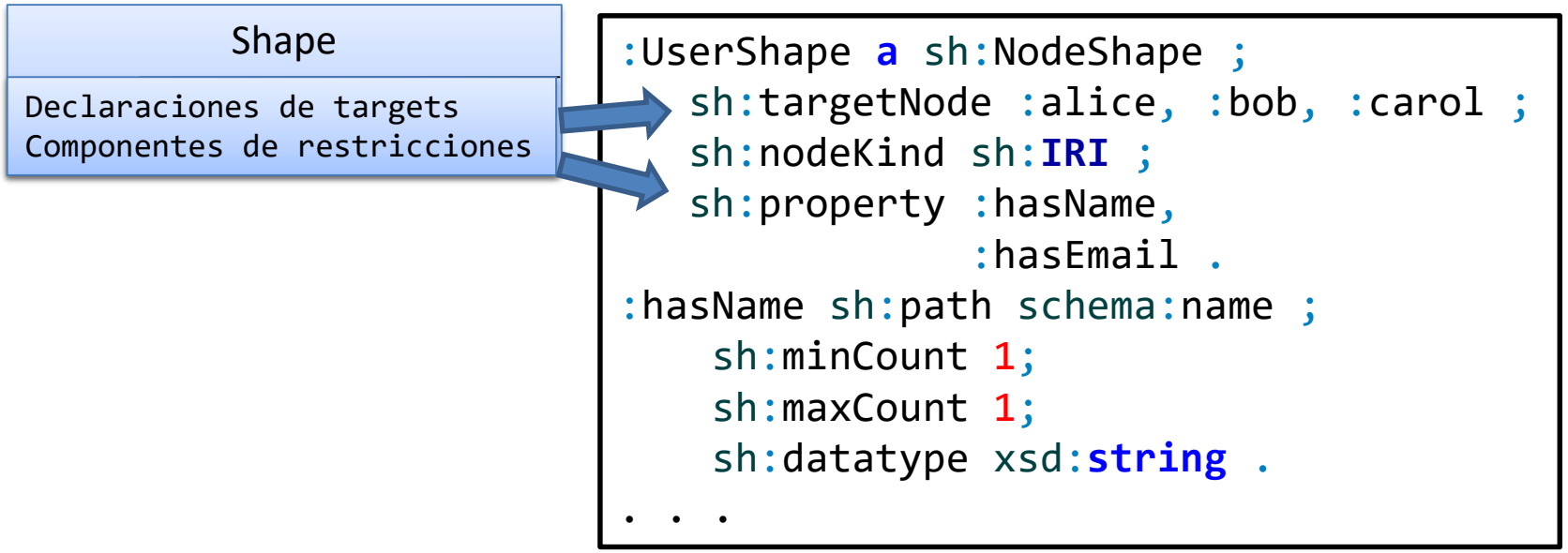
Data graph

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

Algunas definiciones de SHACL

Shape: colección de *targets* and y componentes de restricciones (constraints)

- Targets: especifican qué nodos en el grafo deben cumplir con la shape
- Componentes de restricciones: Determinan cómo validar un nodo



Informe de Validación

Salida del proceso de validación = lista de errores de violación

Si no hay errores ⇒ RDF cumple con el grafo de shapes

```
[ 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
] ;
...
```

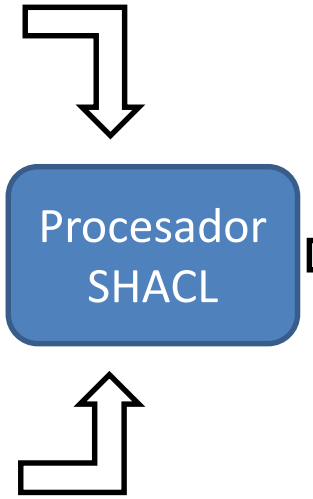
Procesador SHACL

Grafo de Shapes

```
: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 .
. . .
```

Grafo de datos

```
: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> .
```



Informe de validación

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

Ejemplo más largo

En 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}/
```

En 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
   sh:minCount 1 ; sh
 ] ;
  sh:property [ sh:path :worksFor ;
    sh:node :Company ;
    sh:minCount 1 ; sh:maxC
  ].
```

```
: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:NodeSh pe ;
   sh:pattern "[A ]{2}" .
```

Es recursivo!!! (no definido en SHACL)
Característica dependiente de la implementación

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

Shapes vs Ontologías

Ontologías ≠ Shapes ≠ datos (instancias)

- Las ontologías se enfocan normalmente en entidades de dominio (alto nivel)
- Validación RDF (shapes) se enfocan en características del grafo RDF (bajo nivel)

Diferentes niveles

Ontología

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

Shapes
Validación RDF
Restricciones

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

Datos de instancia

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