192.161 Management of Graph Data

(4.0 VU / 6.0 ECTS)

2025W

Integrity Constraints for RDF

Katja Hose
Maxime Jakubowski

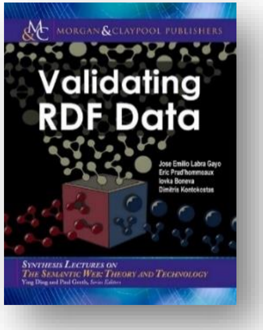
mogda@list.tuwien.ac.at

Learning Goals

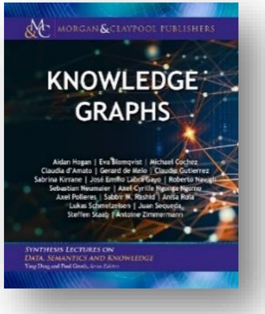
- Motivation
- SHACL – Shapes Constraint Language
- SheX

Literature

- Recommended Literature
  - "Validating RDF data", 2017
  - "Knowledge Graphs", 2021



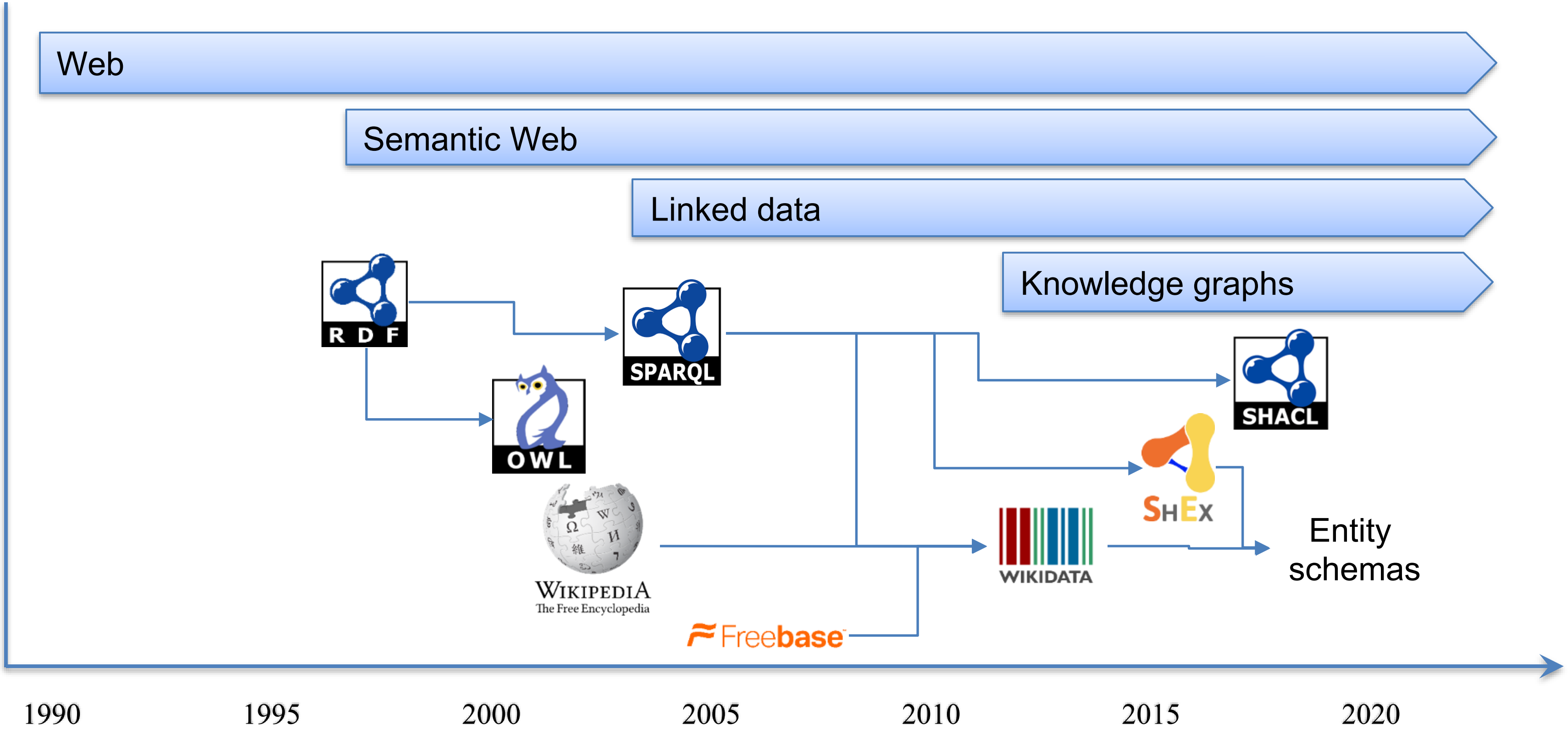
2017 HTML version:
http://book.validatingrdf.com


MORGAN & CLAYPOOL PUBLISHERS
KNOWLEDGE GRAPHS
Aidan Hogan | Eva Blomqvist | Michael Cochez
Claudia d'Amato | Gerard de Melo | Claudio Gutierrez
Sabrina Kirrane | José Emilio Labra Gayo | Roberto Navigli
Sebastian Neumaier | Axel-Cyrille Ngonga Ngomo
Axel Polleres | Sabbir M. Rashid | Anisa Rula
Lukas Schmelzeisen | Juan Sequeda
Steffen Staab | Antoine Zimmermann
SYNTHESIS LECTURES ON
DATA, SEMANTICS AND KNOWLEDGE
Ying Ding and Paul Groth, Series Editors

2021, HTML version
https://kgbook.org/

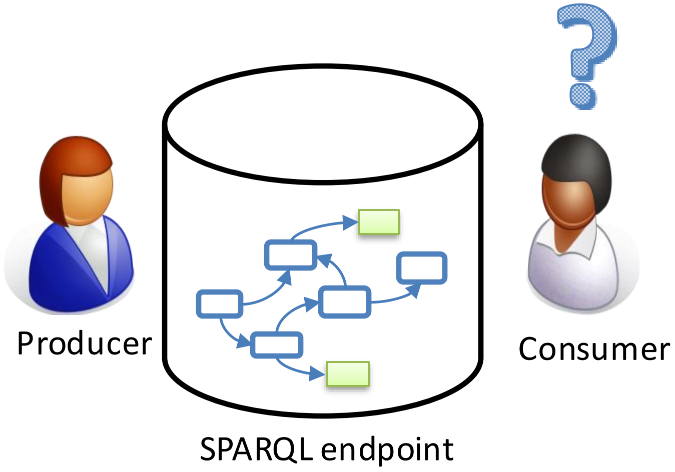
Evolution

Timeline with some concepts and technologies...



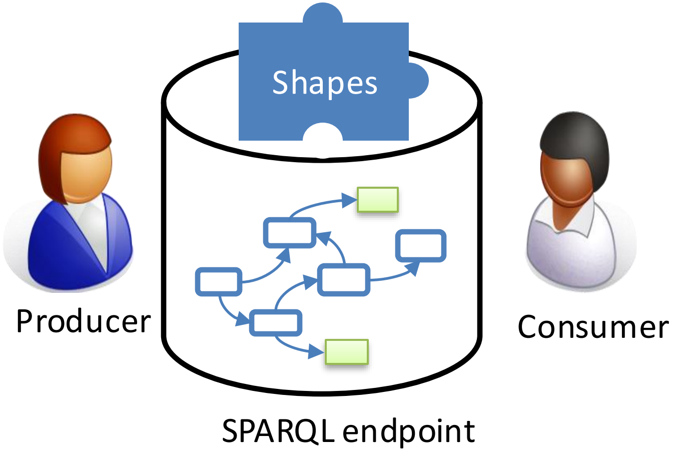
RDF, the other parts

- 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



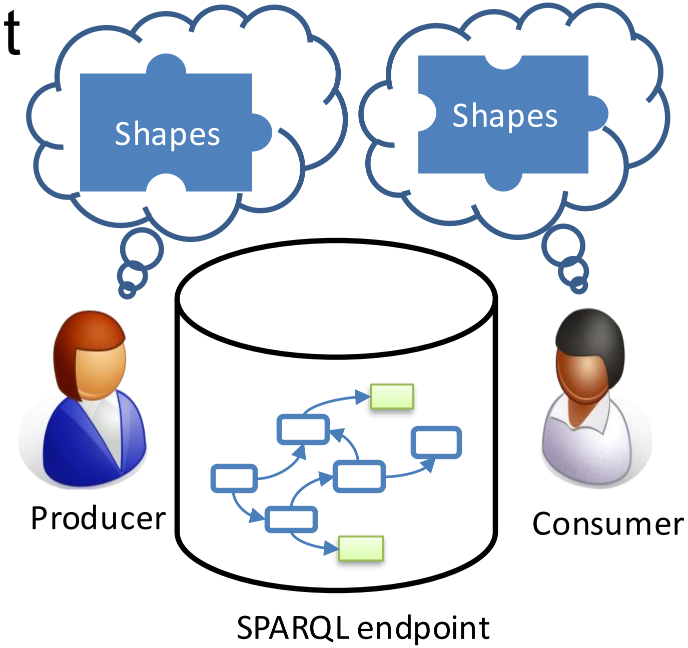
Why describe & validate RDF?

- 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



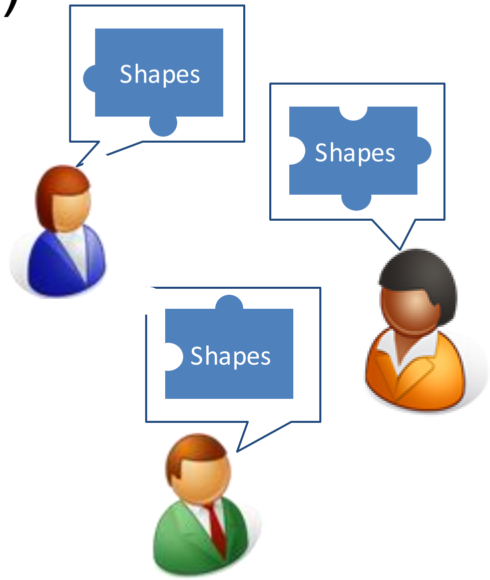
Schemas for RDF?

- RDF flexibility doesn't want to impose a schema, but...
- In practice, there are **implicit schemas**
  - Assumed by producers and consumers
- Shapes can make schemas explicit
  - Handle malformed/incomplete data
  - Avoid defensive programming



Shapes for consensus building

- Help domain experts define their own data models
    - Understandable by domain experts
    - …and machine processable
- Initial motivation: clinical data models (FHIR)
    - Distributed data model
        - Different location, authorities,...
    - Extensible data models



ShEx & 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
- 2024 IEEE ShEx (*work in progress*)

SHACL

SHACL Shapes Constraint Language

Language for validating RDF graphs against a set of **conditions**

W3C recommendation since July 2017: https://www.w3.org/TR/shacl/

data graph: RDF graph to be validated against a shapes graph

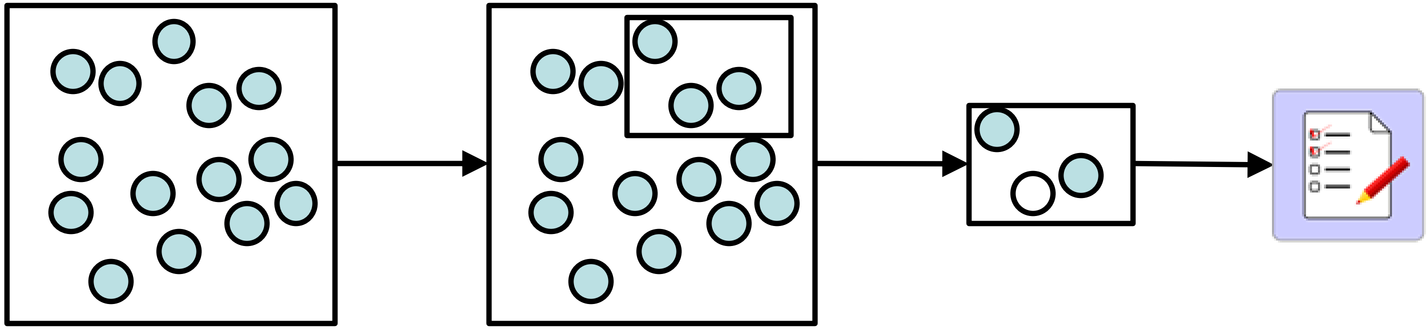
shapes graph: RDF graphs with **conditions** provided as shapes and other constructs

descriptions of the data graphs that do satisfy these conditions

SHACL Shapes Constraint Language

Language for validating RDF graphs against a set of conditions

W3C recommendation since July 2017



Data graph

Apply targets to select focus nodes

Filter some of the nodes

Generate validation report based on constraints

Some 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
rudof	SHACL core (in progress)	Rust	https://rudof-project.github.io/

Playground https://tinyurl.com/y46b2f8q

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

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 ;
  ] .
```

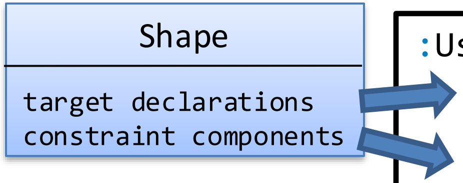
Shapes graph

```
: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

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



```
: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

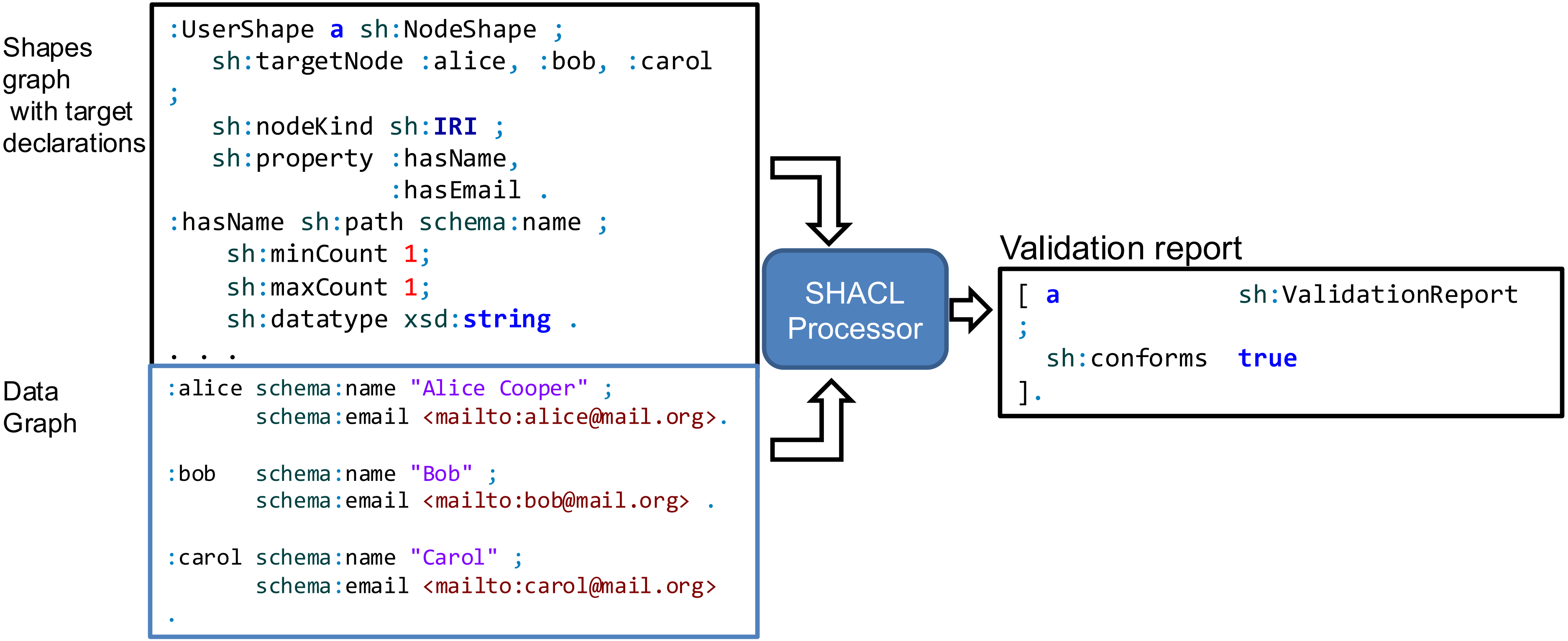
The output of the validation process is a list of violation errors

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 with target declarations

```
: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

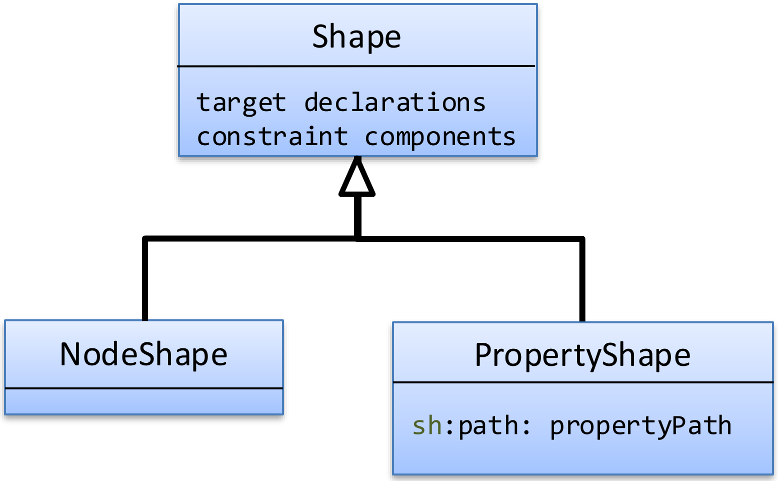
```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org>.

: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
].
```

Node and property shapes

- 2 types of shapes:
  - NodeShape: constraints about shapes of nodes
  - PropertyShapes: constraints on property path values of a node



NodeShape

```
: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 .
. . .
```

PropertyShape

Node Shapes

Constraints about a focus node

```
:UserShape  a sh:NodeShape ;
      sh:nodeKind      sh:IRI ;
      sh:targetClass  :User .
```

```
:alice a :User .

<http://example.org/bob> a :User .

_:1 a :User .     ☹
```

Property shapes

- Constraints about a given property and its values for the focus node
  - `sh:property` associates a shape with a property shape
  - `sh:path` identifies the path

```
:User a sh:NodeShape ;
   sh:property [
      sh:path     schema:email ;
      sh:nodeKind sh:IRI
   ] .
```

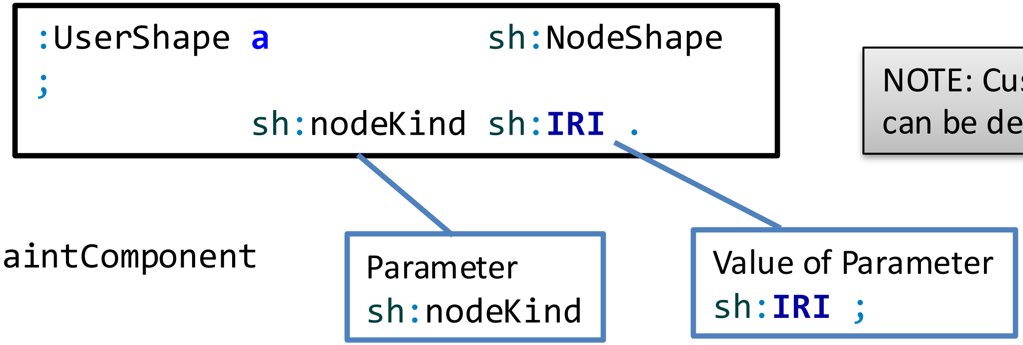
```
:alice a :User ;
       schema:email <mailto:alice@mail.org> .

:bob   a :User;
       schema:email <mailto:bob@mail.org> .

:carol a :User;
       schema:email "carol@mail.org" .
```

☹

Constraint components

- Nodes that declare constraints associated with shapes
  - They have parameters whose values specify the constraints
  - SHACL-core provides a list of predefined constraint components
    - Most of them have one parameter which identifies them

Convention:
Parameter: sh:xx
C. Component: sh:xxConstraintComponent



```
:UserShape a              sh:NodeShape
;
              sh:nodeKind sh:IRI .
```

Parameter
sh:nodeKind

Value of Parameter
sh:IRI ;

Constraint component
sh:nodeKindConstraintComponent

NOTE: Custom constraint components can be defined in SHACL-SPARQL

Repeated parameter

Each value of the parameter declares a different constraint

```
:UserShape a         sh:NodeShape;
           sh:class foaf:Person ;
           sh:class schema:Person .
```

```
:alice a schema:Person, foaf:Person .

:bob a schema:Person .                ☹
```

SHACL Core constraint components

Type	Constraints
Cardinality	`minCount, maxCount`
Types of values	`class, datatype, nodeKind`
Values	`node, in, hasValue, property`

Range of values	`minInclusive, maxInclusive`
	`minExclusive, maxExclusive`

String based	`minLength, maxLength, pattern`
Language based	`languageIn, uniqueLang`
Logical constraints	`not, and, or, xone`
Closed shapes	`closed, ignoredProperties`
Property pair constraints	`equals, disjoint, lessThan, lessThanOrEquals`
Non-validating constraints	`name, description, order, group`

Qualified shapes	`qualifiedValueShape, qualifiedValueShapesDisjoint`
	`qualifiedMinCount, qualifiedMaxCount`

Target declarations

Targets specify nodes that must be validated against the shape

Value	Description
`targetNode`	Directly point to a node
`targetClass`	All nodes that have a given type
`targetSubjectsOf`	All nodes that are subjects of some predicate
`targetObjectsOf`	All nodes that are objects of some predicate

Target node

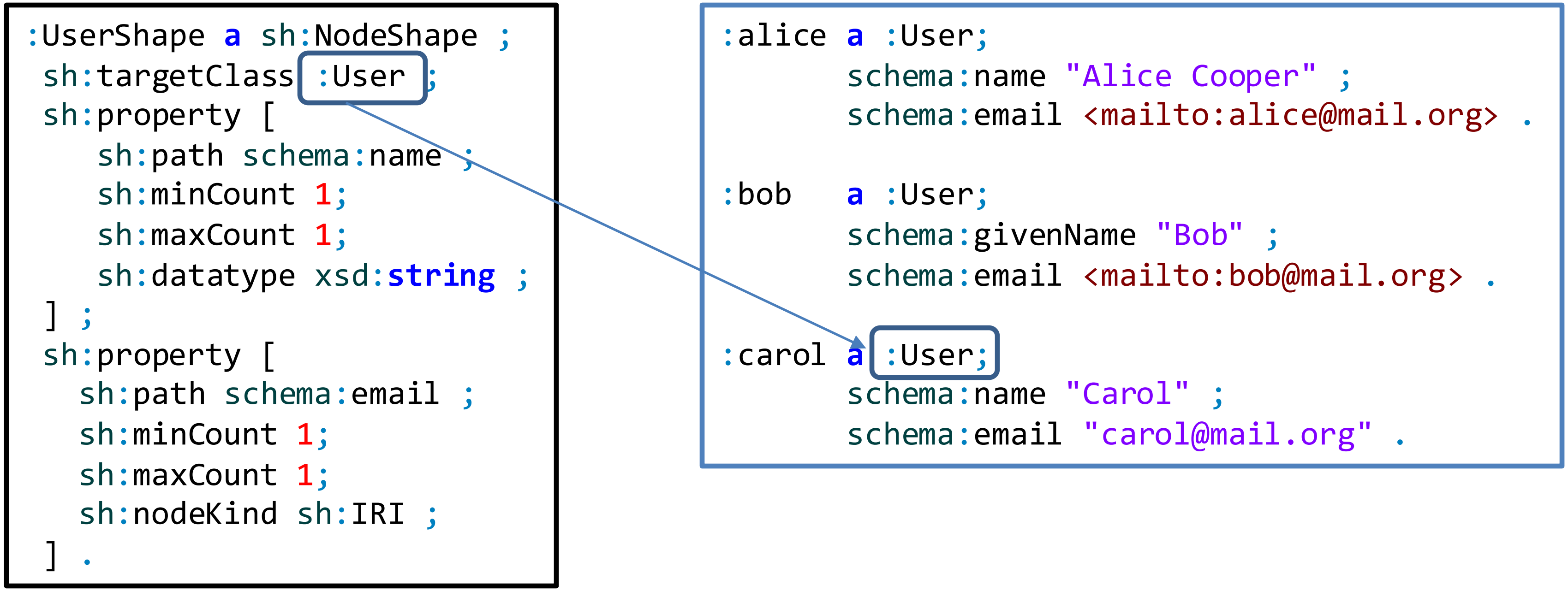
Directly declare which nodes must validate the against the shape

```
:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   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 ;
  ] .
```

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

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

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

Target class

- Selects all nodes that have a given class
  - Looks for rdf:type declarations



```
:UserShape a sh:NodeShape ;
 sh:targetClass :User ;
 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 ;
 ] .
```

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

:bob   a :User;
       schema:givenName "Bob" ;
       schema:email <mailto:bob@mail.org> .

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

targetSubjectsOf

```
:UserShape a sh:NodeShape;
 sh:targetSubjectsOf :teaches ;
 sh:property [
    sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
 ] .
```

```
:alice :teaches :Algebra ;      #Passes as :UserShape
       schema:name "Alice" .

:bob   :teaches :Logic ;        #Fails as :UserShape
       foaf:name "Robert" .

:carol foaf:name 23 .           # Ignored
```

targetObjectsOf

```
:UserShape a sh:NodeShape;
 sh:targetObjectsOf :isTaughtBy ;
 sh:property [
    sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
 ] .
```

```
:alice schema:name "Alice" . #Passes as :UserShape

:bob   foaf:name "Robert" .  #Fails as :UserShape

:carol foaf:name 23 .        # Ignored

:algebra :isTaughtBy :alice, :bob .
```

Targets and focus nodes - examples

Target definitions in a shape graph	Data Graph
ex:PersonShape sh:targetNode ex:Alice .	ex:Alice rdf:type ex:Person . ex:Alice ex:ssn "987-65-432A".
ex:PersonShape sh:targetClass ex:Person .	ex:Bob rdf:type ex:Person . ex:Bob ex:ssn "123-45-6789” . ex:Bob ex:ssn "124-35-6789” .
ex:PersonShape sh:targetSubjectsOf ex:ssn .	ex:Calvin rdf:type ex:Person . ex:Calvin ex:birthDate "1971-07-07"^^xsd:date . ex:Calvin ex:worksFor ex:UntypedCompany .
ex:PersonShape sh:targetObjectsOf ex:worksFor .	

Targets and focus nodes - examples

Shape Graph	Data Graph
ex:PersonShape rdf:type sh:NodeShape .	ex:Alice rdf:type ex:Person .
ex:PersonShape sh:targetNode **ex:Alice** .	ex:Alice ex:ssn "987-65-432A".
	ex:Bob rdf:type ex:Person .
ex:PersonShape sh:targetClass ex:Person .	ex:Bob ex:ssn "123-45-6789" .
	ex:Bob ex:ssn "124-35-6789" .
ex:PersonShape sh:targetSubjectsOf ex:ssn .	ex:Calvin rdf:type ex:Person .
	ex:Calvin ex:birthDate "1971-07-07"^^xsd:date .
	ex:Calvin ex:worksFor ex:UntypedCompany .
ex:PersonShape sh:targetObjectsOf ex:worksFor .	

Targets and focus nodes - examples

Shape Graph	Data Graph
ex:PersonShape rdf:type sh:NodeShape .	**ex:Alice** rdf:type ex:Person .
ex:PersonShape sh:targetNode **ex:Alice** .	ex:Alice ex:ssn "987-65-432A".
ex:PersonShape rdf:type sh:NodeShape .	ex:Bob rdf:type ex:Person .
ex:PersonShape sh:targetClass ex:Person .	ex:Bob ex:ssn "123-45-6789” .
	ex:Bob ex:ssn "124-35-6789” .
ex:PersonShape sh:targetSubjectsOf ex:ssn .	ex:Calvin rdf:type ex:Person .
	ex:Calvin ex:birthDate "1971-07-07"^^xsd:date .
	ex:Calvin ex:worksFor ex:UntypedCompany .
ex:PersonShape sh:targetObjectsOf ex:worksFor .	

Targets and focus nodes - examples

Shape Graph

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetNode **ex:Alice** .

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetClass **ex:Person** .

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetSubjectsOf ex:ssn .

ex:PersonShape sh:targetObjectsOf ex:worksFor .

Data Graph

ex:Alice rdf:type ex:Person .

ex:Alice ex:ssn "987-65-432A".

ex:Bob rdf:type ex:Person .

ex:Bob ex:ssn "123-45-6789” .

ex:Bob ex:ssn "124-35-6789” .

ex:Calvin rdf:type ex:Person .

ex:Calvin ex:birthDate "1971-07-07"^^xsd:date .

ex:Calvin ex:worksFor ex:UntypedCompany .

Targets and focus nodes - examples

Shape Graph

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetNode **ex:Alice** .

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetClass **ex:Person** .

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetSubjectsOf **ex:ssn** .

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetObjectsOf ex:worksFor .

Data Graph

ex:Alice rdf:type ex:Person .

ex:Alice ex:ssn "987-65-432A".

ex:Bob rdf:type ex:Person .

ex:Bob ex:ssn "123-45-6789” .

ex:Bob ex:ssn "124-35-6789” .

ex:Calvin rdf:type ex:Person .

ex:Calvin ex:birthDate "1971-07-07"^^xsd:date .

ex:Calvin ex:worksFor ex:UntypedCompany .

Targets and focus nodes - examples

Shape Graph

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetNode **ex:Alice** .

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetClass **ex:Person** .

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetSubjectsOf **ex:ssn** .

ex:PersonShape rdf:type sh:NodeShape .

ex:PersonShape sh:targetObjectsOf **ex:worksFor** .

Data Graph

ex:Alice rdf:type ex:Person .

ex:Alice ex:ssn "987-65-432A".

ex:Bob rdf:type ex:Person .

ex:Bob ex:ssn "123-45-6789” .

ex:Bob ex:ssn "124-35-6789” .

ex:Calvin rdf:type ex:Person .

ex:Calvin ex:birthDate "1971-07-07"^^xsd:date .

ex:Calvin ex:worksFor **ex:UntypedCompany** .

Core constraint components

Type	Constraints
Cardinality	`minCount, maxCount`
Types of values	`datatype, class, nodeKind`
Values	`node, in, hasValue`

Range of values	`minInclusive, maxInclusive`
	`minExclusive, maxExclusive`

String based	`minLength, maxLength, pattern, stem, uniqueLang`
Logical constraints	`not, and, or, xone`
Closed shapes	`closed, ignoredProperties`
Property pair constraints	`equals, disjoint, lessThan, lessThanOrEquals`
Non-validating constraints	`name, value, defaultValue`

Qualified shapes	`qualifiedValueShape, qualifiedMinCount,`
	`qualifiedMaxCount`

Cardinality constraints

Constraint	Description
minCount	Restricts minimum number of triples involving the focus node and a given predicate. Default value: 0
maxCount	Restricts maximum number of triples involving the focus node and a given predicate. If not defined = unbounded

```
:User a sh:NodeShape ;
   sh:property [
    sh:path      schema:follows ;
    sh:minCount 2 ;
    sh:maxCount 3 ;
  ] .
```

```
:alice schema:follows :bob,
                      :carol .

:bob   schema:follows :alice .    ☹

:carol schema:follows :alice,
                      :bob,       ☹
                      :carol,
                      :dave .
```

Datatypes of values

Constraint	Description
datatype	Restrict the datatype of all value nodes to a given value

```
:User a sh:NodeShape ;
   sh:property [
    sh:path      schema:birthDate ;
    sh:datatype xsd:date ;
  ] .
```

```
:alice schema:birthDate "1985-08-20"^^xsd:date .

:bob   schema:birthDate "Unknown"^^xsd:date .      ☹

:carol schema:birthDate 1990 .                     ☹
```

Logical Operators

Constraint	Description
`and`	Conjunction of a list of shapes
`or`	Disjunction of a list of shapes
`not`	Negation of a shape
`xone`	Exactly one (similar XOR for 2 arguments)

or

```
:User a sh:NodeShape ;
  sh:or (
   [ sh:property [
      sh:predicate foaf:name;
      sh:minCount 1;
     ]
   ]
   [ sh:property [
      sh:predicate schema:name;
      sh:minCount 1;
     ]
   ]
  ) .
```

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

:bob   foaf:name "Robert" .

:carol rdfs:label "Carol" .       ☹
```

and

Default behavior

```
:User a sh:NodeShape ;
  sh:and (
   [ sh:property [
      sh:path      schema:name;
      sh:minCount 1;
     ]
   ]
   [ sh:property [
      sh:path      schema:affiliation;
      sh:minCount 1;
     ]
   ]
  ) .
```

≡

```
:User a sh:Shape ;
   [ sh:property [
      sh:path      schema:name;
      sh:minCount 1;
     ]
   ]
   [ sh:property [
      sh:path      schema:affiliation;
      sh:minCount 1;
     ]
   ]
.
```

not

```
:NotFoaf a sh:NodeShape ;
 sh:not [ a sh:Shape ;
  sh:property [
   sh:predicate foaf:name ;
   sh:minCount 1 ;
 ] ;
 ] .
```

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

:bob   foaf:name "Robert" .   ☹

:carol rdfs:label "Carol" .
```

Exactly One

```
:UserShape a sh:NodeShape ;
  sh:targetClass :User ;
  sh:xone (
   [ sh:property [
      sh:path     foaf:name;
      sh:minCount 1;
     ]
   ]
   [ sh:property [
      sh:path     schema:name;
      sh:minCount 1;
     ]
   ]
  ) .
```

```
:alice a :User ;                   #Passes as :User
       schema:name "Alice" .

:bob   a :User ;                   #Passes as :User
       foaf:name   "Robert" .

:carol a :User ;                   #Fails as :User
       foaf:name   "Carol";
       schema:name "Carol" .

:dave  a :User ;                   #Fails as :User
       rdfs:label  "Dave" .
```

Value Ranges

Constraint	Description
minInclusive	<=
maxInclusive	>=
minExclusive	<
maxExclusive	>

```
:Rating a sh:NodeShape ;
 sh:property [
   sh:path          schema:ratingValue ;
   sh:minInclusive 1 ;
   sh:maxInclusive 5 ;
   sh:datatype      xsd:integer
 ] .
```

```
:bad       schema:ratingValue 1 .

:average   schema:ratingValue 3 .

:veryGood  schema:ratingValue 5 .

:zero      schema:ratingValue 0 .   ☹
```

String based constraints

Constraint	Description
`minLength`	Restricts the minimum string length on value nodes
`maxLength`	Restricts the maximum string length on value nodes
`pattern`	Checks if the string value matches a regular expression

Language based constraints

Constraint	Description
`languageIn`	Declares the allowed languages of a literal
`uniqueLang`	Specifies that no pair of nodes can have the same language tag

Closed shapes

Constraint	Description
`closed`	Valid resources must only have values for properties that appear in `sh:property`
`ignoredProperties`	Optional list of properties that are also permitted

```
:User a sh:NodeShape ;
  sh:closed true ;
  sh:ignoredProperties ( rdf:type ) ;
  sh:property [
    sh:path schema:givenName ;
  ];
  sh:property [
    sh:path schema:lastName ;
  ] .
```

```
:alice schema:givenName "Alice";
       schema:lastName "Cooper" .

:bob   a :Employee ;
       schema:givenName "Bob";
       schema:lastName "Smith" .

:carol schema:givenName "Carol";
       schema:lastName "King" ;         ☹
       rdfs:label "Carol" .
```

SPARQL constraints

- Constraints based on SPARQL code
  - When the SPARQL query returns validation errors, a violation is reported
  - SPARQL constraints have type sh:SPARQLConstraint

Constraint	Description
`message`	Message in case of error
`sparql`	SPARQL code that is run
`prefixes`	Points to namespace prefix declarations defined by sh:declare: Each one has: sh:prefix: Prefix alias sh:namespace: namespace IRI

SPARQL constraints

Example: Name must be the concatenation of singleName and familyName

```
:UserShape a sh:NodeShape ;
 sh:targetClass :User ;
 sh:sparql [ a sh:SPARQLConstraint ;
  sh:message "schema:name must equal schema:givenName+schema:familyName";
  sh:prefixes [ sh:declare [
    sh:prefix "schema" ;
    sh:namespace "http://schema.org/"^^xsd:anyURI ;
   ]] ;
 sh:select
  """SELECT $this (schema:name AS ?path) (?name as ?value)
     WHERE {
      $this schema:name ?name .
      $this schema:givenName ?givenName .
      $this schema:familyName ?familyName .
      FILTER (!isLiteral(?value) ||
              !isLiteral(?givenName) || !isLiteral(?familyName) ||
              concat(str(?givenName), ' ', str(?familyName))!=?name )
     }""" ;
] .
```

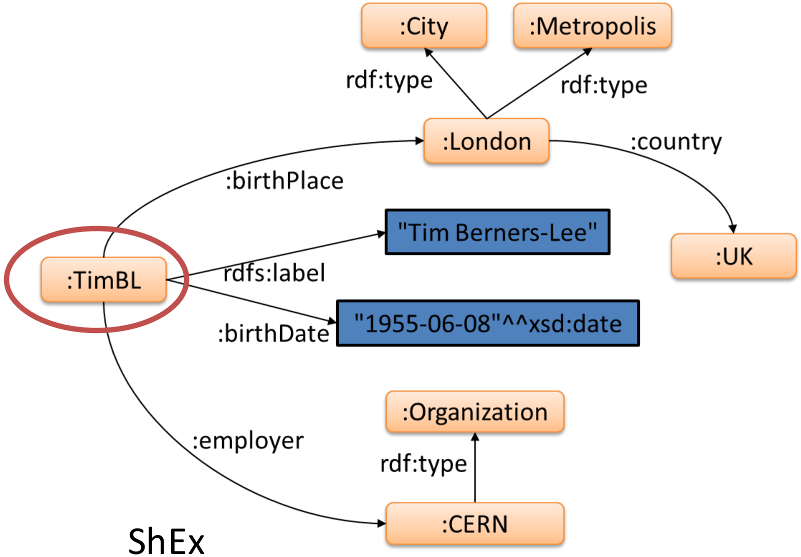
```
:alice a :User ;
  schema:givenName "Alice" ;
  schema:familyName "Cooper" ;
  schema:name "Alice Cooper" .

:bob a :User ;
  schema:givenName "Bob" ;
  schema:familyName "Smith" ;
  schema:name "Robert Smith" .
```

☹

ShEx

Example of a shape

- A shape describes
  - The form of a node (node constraint)
  - Incoming/outgoing arcs from a node
  - Possible values associated with those arcs



RDF Node

```
:timbl rdfs:label   "Tim Berners-Lee" ;
       :birthPlace :london ;
       :birthDate  "1955-06-08"^^xsd:date ;
       :employer   :CERN .
```

ShEx

```
<Researcher> {
 rdfs:label  xsd:string          ;
 :birthPlace @<Place>          ? ;
 :birthDate  xsd:date          ? ;
 :employer   @<Organization> * ;
}
```

Like regular expressions: * (zero or more), + (one or more), ? (zero or one)

Shape Expressions - ShEx

- Goal: Concise and human-readable language
- 3 syntaxes:
  - ShExC: Compact syntax, similar to Turtle or SPARQL
  - ShExJ: JSON(-LD), for the spec
  - ShExR: RDF, based on JSON-LD

    Note: Round tripping is possible, convert from one to the other
- Semantics inspired by regular expressions

ShEx libraries and demos

Implementations & libraries:

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

Online demos & playgrounds

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

More info: http://shex.io

Simple example

Prefix declarations as in Turtle/SPARQL

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

:Book IRI AND {
 schema:name   xsd:string   ;
 :related      @:Book     *
}
```

Nodes conforming to `:Book` must

- Be `IRI`s and
- Have property `schema:name` with a value of type `xsd:string` (exactly one)
- Have property `:related` with values conforming to `:Book` (zero or more)

RDF Validation using ShEx

Schema

```
:Book IRI AND {
 :name    xsd:string   ;
 :related @:Book      *
}
```

Shape map

```
:a@:Book   ✓
:b@:Book,  ✓
:c@:Book,  ✗
:d@:Book,  ✗
:e@:Book,  ✗
:f@:Book   ✗
```

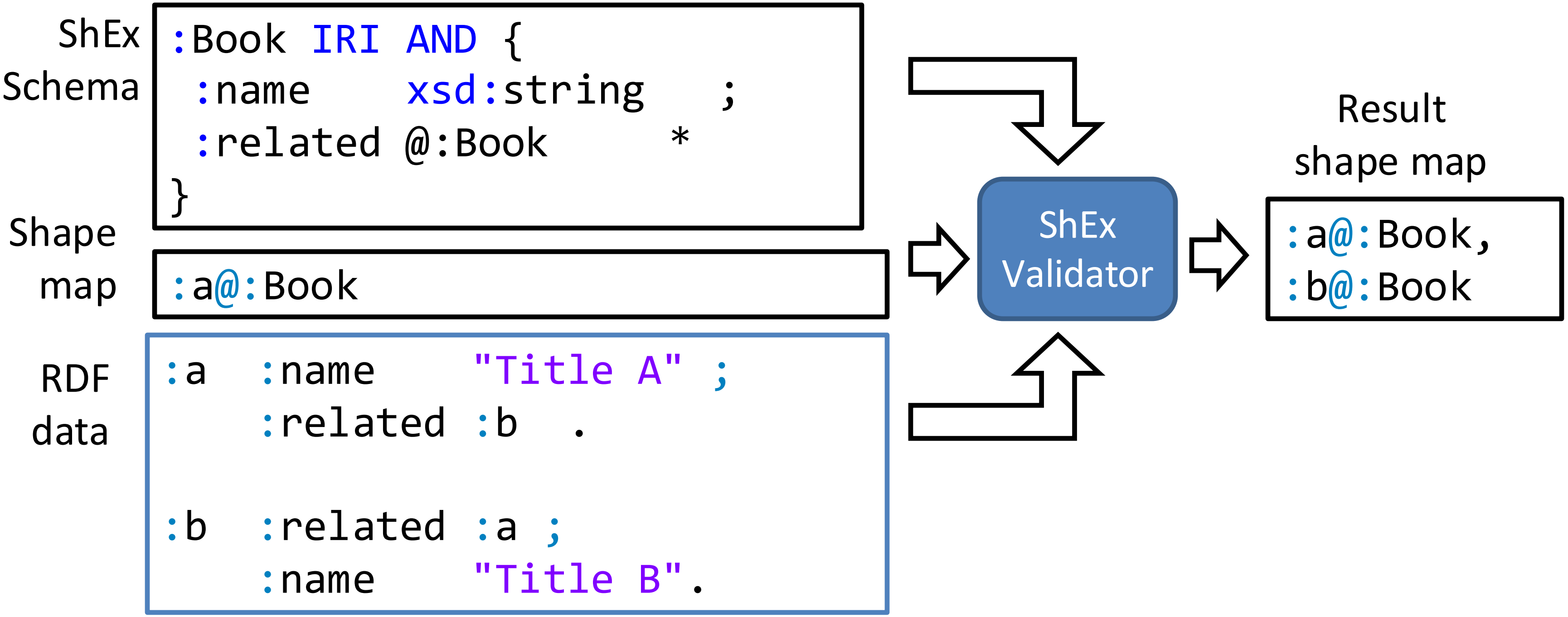
RDF Data

```
:a  :name    "Title A" ;
    :related :b  .
:b  :related :a ;
    :name    "Title B".
:c  :name    "Title C1", "Title C2" .
:d  :name    234 .
:e  :namme   "Title E" .
:f  :name    "Title F" ;
    :related :a, _:1 .
_:1 :name    "Unknown title" .
```

Validation process

Input: RDF data, ShEx schema, Shape map

Output: Result shape map



ShEx Schema

```
:Book IRI AND {
 :name    xsd:string   ;
 :related @:Book      *
}
```

Shape map

```
:a@:Book
```

RDF data

```
:a  :name    "Title A" ;
    :related :b   .

:b  :related :a ;
    :name    "Title B".
```

Result shape map

```
:a@:Book,
:b@:Book
```

Node constraints

Constraints over a node (without considering its neighborhood)

```
:Book {
 :name          xsd:string
 :datePublished xsd:date
 :numberOfPages MinInclusive 1
 :author        @:Person
 :genre         [:Fiction :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                             .
```

Cardinalities

Inspired by regular expressions: +, ?, *, {m,n}

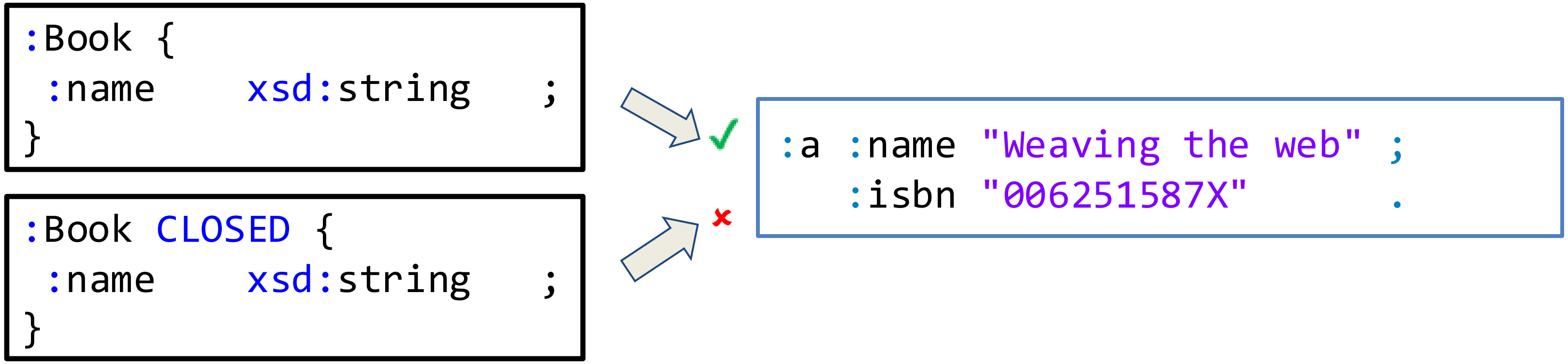
By default {1,1}

```
: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             .
```

Open/Closed content models

- RDF semantics mostly presume open content models
- Shape expressions are open by default
    - Enable extensibility
- But…some use cases require closed content models
    - Added CLOSED keyword



```
:Book {
 :name     xsd:string   ;
}
```

✓

```
:Book CLOSED {
 :name     xsd:string   ;
}
```

✗

```
:a :name "Weaving the web" ;
   :isbn "006251587X"      .
```

Open/Closed properties

Property values are closed by default (closed properties)

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

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

✗
```
:item23  :code 23                .
```

Properties can be repeated

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

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

EXTRA declares properties as open

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

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

Logical operators

Shape Expressions can be combined with AND, OR, NOT

```
: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            .
```

Inheritance model for ShEx

`extends` allows to reuse existing shapes adding new content

- Handles closed properties and shapes

```
: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"        .
```

Other features

- Multiple inheritance
- Abstract shapes

3 syntaxes: ShExC, ShExJ, ShExR

ShExC

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

:Book {
 schema:name   xsd:string   ;
 :related      @:Book     *
}
```

ShExR (RDF, Turtle)

```
:Book a sx:ShapeDecl ;
  sx:shapeExpr [ a sx:Shape ;
   sx:expression [ a sx:EachOf ;
    sx:expressions (
      [ a sx:TripleConstraint ;
        sx:predicate schema:name ;
        sx:valueExpr  [ a sx:NodeConstraint ;
                        sx:datatype xsd:string
                      ] ]
      [ a sx:TripleConstraint ;
        sx:predicate :related ;
        sx:valueExpr  [ a sx:NodeConstraint ;
        sx:valueExpr :Book ] ] ) ] ] .
```

It's possible to roundtrip from each one

ShExJ (JSON LD)

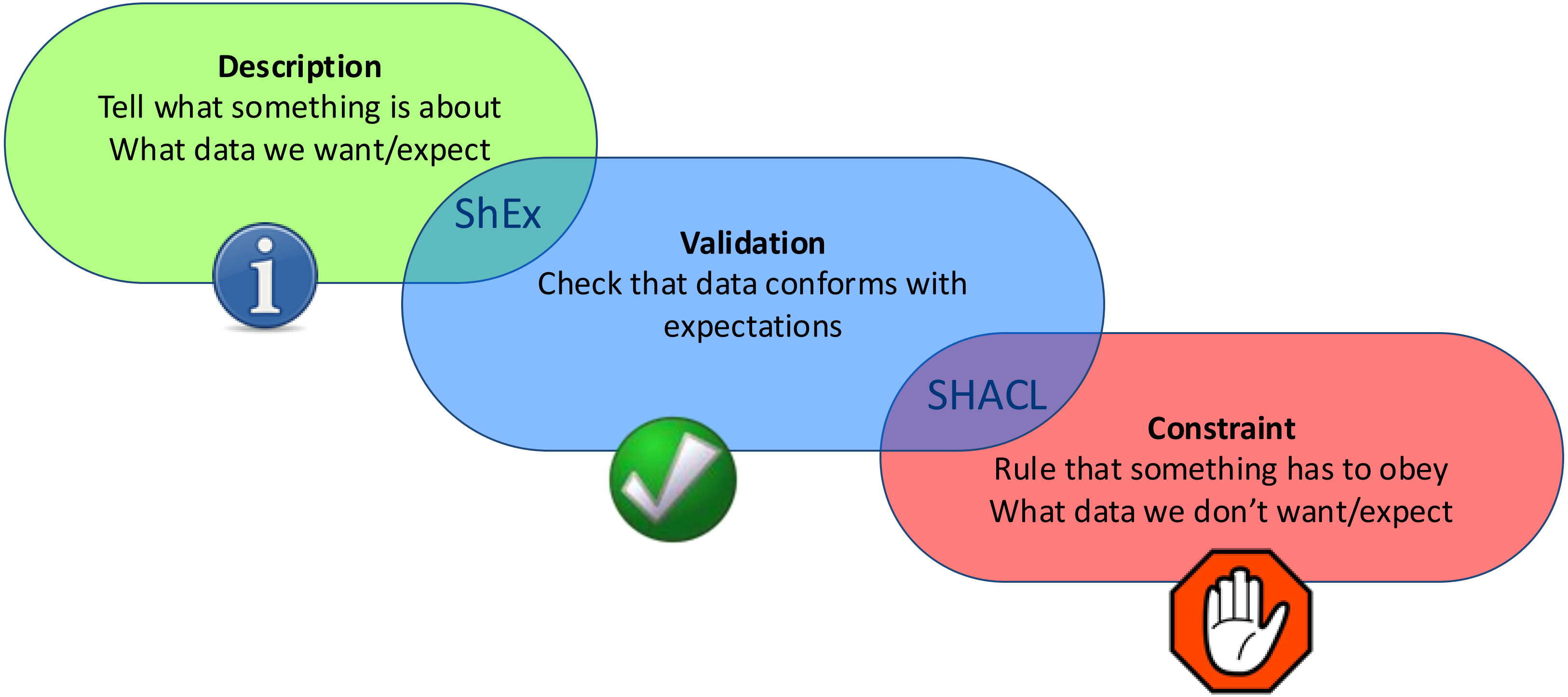
```
{ "type" : "Schema",
  "@context" : "http://www.w3.org/ns/shex.jsonld",
  "shapes" : [ {
      "type" : "Shape",
      "id" : "http://example.org/Book",
      "expression" : {
        "type" : "EachOf",
        "expressions" : [ {
            "type" : "TripleConstraint",
            "predicate" : "http://schema.org/name",
            "valueExpr" : {
              "type" : "NodeConstraint",
              "datatype" : "http://www.w3.org/2001/XMLSchema#string"
            }
          },
          { "predicate" : "http://example.org/related",
            "valueExpr" : "http://example.org/Book",
            "min" : 0,
            "max" : -1,
            "type" : "TripleConstraint"
          }
        ] } } ] }
```

Summary

- More ShEx features
    - Stems, named expressions, nested shapes, semantic actions, …
- ShEx tools
- ShEx and SHACL compared
    - Different underlying philosophy
        - ShEx more inspired by grammars than constraints
    - Separation of concerns
        - Structure definition (ShEx) ≠ Ontology (OWL)
        - Structure definition (ShEx) ≠ Node/shape selection (ShapeMaps)

SHACL vs. ShEx

Description - validation - constraints



Description
Tell what something is about
What data we want/expect

ShEx

Validation
Check that data conforms with expectations

SHACL

Constraint
Rule that something has to obey
What data we don't want/expect

Underlying philosophy

ShEx is more *schema* based

Shape ≈ grammar

More focus on validation results

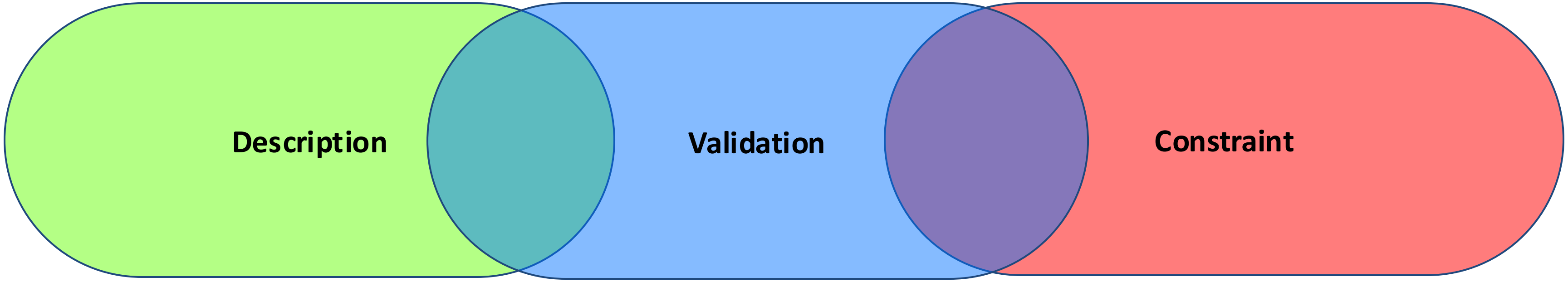
Result shape maps = Conforming and non-conforming nodes

SHACL is more constraint based

Shapes ≈ collections of constraints

More focus on validation errors

Validation report = set of violations



Summary

- Shape constraints can help improve the quality of knowledge graphs
- Shapes can be used to define a broad range of constraints
- Validation reports can be used to increase the quality of an RDF graph
- SHACL
- ShEx