Semantic Web 0 (0) 1
IOS Press

XMLSchema2ShEx: Converting XML validation to RDF validation

Herminio Garcia-Gonzalez [a,*] Jose Emilio Labra-Gayo [a,**]
[a] *Department of Computer Science, University of Oviedo, C/ Federico García Lorca S/N 33007*

Abstract. RDF validation is a new topic where the Semantic Web community is focusing attention while in other communities, like XML or databases, data validation and quality was considered a key part of their ecosystem. On the other hand, there is a recent trend to migrate data from different sources to semantic web formats. These transformations and mappings between different technologies come at a price. In order to facilitate this transformation, we propose a set of mappings that can be used to convert from XML Schema to Shape Expressions (ShEx)—one of the recent RDF validation languages—. We also present a prototype that implements a subset of the mappings proposed, and an example application to obtain a ShEx schema from an XML Schema one. We consider that this work and the development of other format mappings could drive to a new era of semantic-aware and interoperable data.

Keywords: ShEx, XML Schema, Shape Expressions, formats mapping, data validation

1. Introduction

Data validation is one of the key areas when normalisation and reliance are desired. Normalisation is desired as a way of making a dataset more trustworthy and even more useful to possible consumers because of its predictable schema. Validation can excel data cleansing, querying and standardisation. In words of P.N. Fox et.al.: "Procedures for data validation increase the value of data and the users' confidence in predictions made from them. Well-designed data management systems may strengthen data validation itself, by providing better estimates of expected values than were available previously." [12]. Therefore, validation is a key field of data management.

XML Schema [4] was designed as a language to make XML validation possible and with more features than DTDs [3]. With XML Schema, developers can define the structure, constraints and documentation of an XML vocabulary. Alongside the appearance of DTDs and XML Schema, other alternatives (such as Relax NG [7] and Schematron [13]) were proposed.

Unlike XML, RDF lacked a standard schema language. Some alternatives were OWL and RDF Schema, however, they do not cover completely what XML Schema does with XML [29]. For this purpose, Shape Expressions (ShEx) [25] was proposed to fulfill the requirement of a validation language for RDF [26], and SHACL has recently become a W3C recommendation [16].

As many documents and data are persisted in XML, migration and interoperability needs are nowadays more pressing than before, many authors have proposed conversions from XML to RDF [20,8,1,5], which have the goal of transforming XML data to Semantic Web formats.

Although these conversions enable users to migrate their data to Semantic Web technologies, a lacking process when converting XML to RDF is validation. How to be ensure that the conversion has been done correctly and that both versions—in different languages—are defining the same type, i.e. how to migrate all the effort put in validation mark-up and preserve this functionality in the new platform.

Conversions between XML and RDF and between XML Schema and ShEx are necessary to alleviate the gap between semantic technologies and more tradi-

*Email: herminiogg@gmail.com
**Email: labra@uniovi.es

tional ones. With that in mind, providing migrations from in-use technologies to semantic technologies can enhance the migration possibilities. Although we consider that generic approaches for some of these conversions are not going to be valid in all cases, in other cases like small companies or low budget projects they can make their point as initial or by-default transformations. Taking TEI [10] as an example, digital humanities can take the benefit of Semantic Web approaches [28,27]. There are a lot of manuscripts transcribed to XML that can be converted to RDF. But transcribers are not going to deal with the underlying technology despite they can benefit from it [19]. Those are the cases where generic approaches can offer a solution and, therefore, automatic conversion of schemata has its space when transformations can be checked.

With that problem in mind, the questions that we want to address in the present work are the following:

- Is a mapping between XML Schema and ShEx reachable?
- In case this mapping is possible, how can one be sure that both schemata are defining the same meaning?
- How to ensure that both schemata are equivalent and, moreover, backwards conversion can be performed?
- What are the conditions to ensure a valid conversion?

Therefore, a solution on how to make the conversion from XML Schema to ShEx is described in this paper. Detailing how each element in XML Schema can be translated into ShEx. Moreover, a prototype that can convert a subset of what is defined in the following sections is also presented.

The rest of the paper is structured as follows: Section 2 presents the background, Section 3 gives a brief introduction to ShEx, Section 4 describes a possible set of mappings between XML Schema and ShEx, Section 5 presents a prototype used to validate a subset of previously presented mappings and how this conversion works against existing RDF validators. Finally, Section 6 draws some conclusions and future lines of work and improvement.

2. Background

Conversion to Semantic Web formats is a field that presents several previous works. In the XML community, many conversions to RDF—and backwards—have been proposed using different techniques. In [20] authors describe their experience on developing this transformation for business to business industry. In [8] an ontology based transformation is described. In [1] they try to solve the lift problem (the problem of how to map heterogeneous data sources in the same representational framework) from XML to RDF and backwards by using the Gloze mapping approach on top of Apache Jena. In [5] authors describe a transformation supported on SPARQL and in [2] a transformation from RDF to other kind of formats, including XML, is proposed using embedded SPARQL into XSLT stylesheets which, by means of these extensions, could retrieve, query, merge and transform data from the Semantic Web.

Data validation is also a key question [12] as it has been previously stated in this paper. In [23] a dictionary of transformations is defined based on similarities between XML and JSON schemas. In [14] authors patented a mechanism to convert XML Schema components to Java components. In [24] an algorithm that converts from XML Schemata to ER diagrams is proposed. And in [22], the authors propose the conversion from XML Schema to XText to bring more functionalities to domain specific languages based on XML Schema.

Another approach for transformation between schemas is to take a domain model as the main representation and then transform between that model and other schema formats like XML Schema, JSON Schema or ShEx. This has been the approach followed by FHIR [1].

More focused on Semantic Web technologies, other approaches have been taken to transform XML Schema to OWL [11] or RDF Schema [20].

RDF Schema and OWL were not designed as RDF validation languages. Their use of Open World and Non-Unique Name Assumptions can pose some difficulties to define the integrity constraints that RDF validation languages require [29]. Various languages have recently been developed for RDF validation. On one hand Shapes Constraint Language (SHACL) [16] has been developed by the W3C Data Shapes Working Group and Shape Expressions (ShEx) [26] is being developed by the W3C Shape Expressions Community Group. In this paper, ShEx is used to describe the mappings due to its compact syntax and its support for recursion whereas in SHACL recursion depends on

[1] https://www.hl7.org/fhir/

the implementation. However, we consider that converting the mappings proposed in this paper to SHACL is feasible and can be an interesting line of future work given that it has already been accepted as a W3C recommendation and that there are some ways to simulate recursion by target declarations or property paths.

To the best of our knowledge, no conversion between XML Schema and ShEx has been proposed to date. This might be due to the recent introduction of ShEx. In this paper, a transformation from XML Schema to ShEx is proposed, indicating how each element could be translated.

3. Brief introduction to ShEx

ShEx was proposed as a language for RDF validation in 2014 [26]. It was one of the inputs for the W3C data shapes working group which developed the Shapes Constraint Language (SHACL) for the same purpose. SHACL was also inspired by SPIN [15] and although both languages can perform RDF validation there are some differences between them like the support of recursion or the emphasis on validation vs constraint checking (see chapter 7 of [17] for more details). In this paper we will focus on ShEx because it has a well-defined semantics for recursion [6] and its semantics are more inspired by grammar-based formalisms like RelaxNG.

ShEx syntax was inspired by Turtle, SPARQL and RelaxNG with the aim to offer a concise and easy to use syntax. Nowadays, version 2.0 was released with a primer and the working group is developing the 2.1 version.

ShEx uses shapes to group different validations associated with the same node 'type'. That is, a shape can define how a node and its triples should be in order to be valid. In Listing 1 there is an example of a ShEx shape:

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

:PurchaseOrder {
  :orderId           /Order\\d{2}/ ;
  schema:customer    @:User ;
  schema:orderDate   xsd:date ? ;
  schema:orderedItem @:Item +
}
:Item {
  schema:name xsd:string ;
  :quantity xsd:positiveInteger OR
            xsd:integer MININCLUSIVE 1
}
:User {
  a [ schema:Person ] ;
  :purchaseOrder @:PurchaseOrder*
}
```

Listing 1: ShEx shape example

Listing 1 defines a shape with a `:PurchaseOrder` type. Prefixes are defined at the beginning of the snippet and use the same similar syntax as in Turtle. Triple constraints are defined inside the shape where a purchase order must have an orderId of type that matches the regular expression `Order\d{2}`, it must have a `schema:customer` which must be a node that conforms to shape `:User`, a `schema:orderDate` whose value must be an `xsd:date` and can have one or more (represented by the plus sign) `schema:orderedItem` whose values must conform to the `:Item` shape.

The `:Item` shape must have a `schema:name` of value string and a `schema:orderQuantity` of value `xsd:positiveInteger`, while the `:User` shape declares that the values must have type `schema:Person`, and can contain zero or more values of `:purchaseOrder` which must conform to the `:PurchaseOrder` shape.

```
### Pass validation as :PurchaseOrder
:order1 :orderId "Order23" ;
 schema:customer      :alice ;
 schema:orderDate     "2017-03-02"^^xsd:date;
 schema:orderedItem   :item1 .
:alice a              schema:Person ;
       :purchaseOrder :order1 .
:item1 schema:name    "Lawn" ;
      :quantity       2 .

### Fails validation as :PurchaseOrder
:order2 :orderId    "MyOrder" ;
 schema:customer    :bob;
 schema:orderDate   2017;
 schema:orderedItem :item1.
:bob a              schema:Person ;
    :purchaseOrder  :unknown.
```

Listing 2: RDF validation example

In Listing 2 there is an example of two purchase orders defined in RDF. The first of them passes validation and conforms to the shapes declaration whereas `:order2` fails for several reasons: the value of `:orderId` does not conform to the required regular expression, the value of `schema:customer` does not con-

form to shape `:User` and the value of `schema:orderDate` does not have datatype `xsd:date`.

ShEx supports different serialization formats:

- ShExC: a concise human readable compact syntax which is the one presented in previous example.
- ShExJ: a JSON-LD syntax which is used as an abstract syntax in ShEx specification.
- ShExR: an RDF representation syntax based on ShExJ.

In this paper ShExC syntax was selected because it is intended for humans and it is more easy to read and understand. The goal of this introduction was to provide a basic understanding of ShEx. For more examples and a longer comparison between ShEx and SHACL technologies readers can consult [17].

4. Mappings between XML Schema and ShEx

XML Schema defines a set of elements and datatypes for doing the validation that need to be converted to ShEx. In this section, we describe different XML Schema elements and what a possible conversion to ShEx can be. All examples use the default prefix : for URIs. It is intended to be replaced by different prefixes depending on the required namespaces. For XML Schema elements and datatypes `xs` prefix is used in the examples.

4.1. Element

Elements are treated as a triple predicate and object, i.e., we convert them to a triple constraint whose predicate is the element's name:

```
### XML Schema
<xs:element name="birthday" type="xs:date"/>

### ShEx
:birthday xs:date ;
```

Listing 3: Element mapping

The `name` attribute is used as the fragment of the URI in the predicate and the type is transcribed directly, as ShEx has built-in support for XML Schema datatypes, there is a direct match between them. If the `ref` attribute is present, the type should be defined somewhere to link the corresponding type or shape. When an `element` type is a `xs:complexType`, the type should be referenced to a new shape where the `xs:complexType` is converted (see Section 4.3 where we explain how to convert `xs:complexType` to a shape).

```
### XML Schema
<xs:element name="purchaseOrder"
            type="PurchaseOrderType"/>

<xs:complexType name="PurchaseOrderType">
    ...
</xs:complexType>

### ShEX
:purchaseOrder @<PurchaseOrderType> ;
```

Listing 4: Element mapping with linked type

```
### XML Schema
<xs:element name="item"
            minOccurs="0"
            maxOccurs="unbounded">
    <xs:complexType>
        ...
    </xs:complexType>
</xs:element>

### ShEx
:item @<item> * ;
```

Listing 5: Element mapping with nested type

4.1.1. Cardinality

Cardinality in ShEx is defined with the following symbols: '*' for 0 or more repetitions, '+' for 1 or more repetitions, '?' for 0 or 1 repetitions (optional element) or '{m, n}' for m to n repetitions where m is `minOccurs` and n `maxOccurs`. As in XML Schema, the default cardinality in ShEx is 1 for lower and upper bounds. Therefore, transformation of `minOccurs` and `maxOccurs` in the previously defined cardinality marks is done as showed in Listing 6.

```
### XML Schema
<xs:element name="nameZeroUnbounded"
            type="xs:string"
            minOccurs="0"
            maxOccurs="unbounded">
<xs:element name="nameOneUnbounded"
            type="xs:string"
            minOccurs="1"
            maxOccurs="unbounded">
<xs:element name="nameOptional"
```

```
            type="xs:string"
            minOccurs="0"
            maxOccurs="1">
<xs:element name="nameFourToTen"
            type="xs:string"
            minOccurs="4"
            maxOccurs="10">

### ShEx
:nameZeroUnbounded xs:string * ;
:nameOneUnbounded xs:string + ;
:nameOptional xs:string ? ;
:nameFourToTen xs:string {4, 10} ;
```

Listing 6: Cardinality mapping

As presented in the previous examples, when an element has its complex type nested the shape name will be the `name` of the element.

4.2. Attribute

Attributes are treated as elements in ShEx. ShEx makes no difference between an attribute and an element because this difference is part of XML data model and the RDF data model does not have the concept of attributes. One possibility to transform attributes is to use their `name` and `type` as performed with elements (see Section 4.1). This allows better readability of the corresponding RDF data, but limits roundtrip conversions between XML to RDF and back.

4.3. ComplexType

Complex types are translated directly to ShEx shapes. The `name` of the `complexType` will be the name of the shape to which elements can refer to. Complex types can be compound of different statements so we provide a detailed transformation of each possibility below.

```
### XML Schema
<xs:complexType name="PurchaseOrderType">
    ...
</xs:complexType>

### ShEx
<PurchaseOrderType> {
    ...
}
```

Listing 7: Complex type mapping

4.3.1. Sequence

While sequences in XML Schema define sequential order of elements, in ShEx this is more complex due to the RDF graph structure. There are several ways to represent order in RDF, the most obvious is using RDF lists. However, this is one of the possible ways of doing that and there can be other ways to represent it [9,18].

The following example shows how the mapping is done for a `sequence` using RDF lists:

```
### XML Schema
<xs:complexType name="Address">
    <xs:sequence>
        <xs:element name="street"
            type="xs:string"/>
        <xs:element name="city"
            type="xs:string"/>
        <xs:element name="state"
            type="xs:string"/>
        <xs:element name="zip"
            type="xs:decimal"/>
    </xs:sequence>
</xs:complexType>

### ShEx
<address> {
    rdf:first @<street> ;
    rdf:rest @<i1> ;
}
<i1> {
    rdf:first @<city> ;
    rdf:rest @<i2> ;
}
<i2> {
    rdf:first @<state> ;
    rdf:rest @<i3> ;
}
<i3> {
    rdf:first @<zip> ;
    rdf:rest [ rdf:nil ] ;
}
<street> {
    :street xs:string ;
}
<city> {
    :city xs:string ;
}
<state> {
    :state xs:string ;
}
<zip> {
    :zip xs:decimal ;
}
```

Listing 8: Sequence mapping

4.3.2. Choice

Choices in XML Schema are the disjunction operator to select between two options, for instance: choice between two elements. This operator is supported in ShEx using the *oneOf* operator ('|'). The object and predicate of the RDF statement must be one of the enclosed ones. Therefore, translation is performed as shown in the following snippet:

```
### XML Schema
<xs:choice>
  <xs:element name="name"
              type="xs:string"/>
  <xs:sequence>
    <xs:element name="givenName"
                type="xs:string"
                maxOccurs="unbounded"/>
    <xs:element name="familyName"
                type="xs:string" />
  </xs:sequence>
</xs:choice>

### ShEx
( :name xs:string |
  :givenName xs:string + ;
  :familyName xs:string
) ;
```

Listing 9: Choice mapping

4.3.3. All

While sequences are an ordered set of elements, `all` is instead a set of unordered elements. Indeed, `all` has a better representation using ShEx elements and the transformation is simpler than the `sequence` one as there is no need to keep track of the order of elements.

```
### XML Schema
<xs:all>
    <xs:element name="street"
                type="xs:string"/>
    <xs:element name="city"
                type="xs:string"/>
    <xs:element name="state"
                type="xs:string"/>
    <xs:element name="zip"
                type="xs:decimal"/>
</xs:all>

### ShEx
:street  xs:string  ;
:city    xs:string  ;
:state   xs:string  ;
:zip     xs:decimal ;
```

Listing 10: All mapping

4.4. XSD Types

XSD Types can be used on ShEx as they are used on XML Schema, e.g. whenever a string type is desired we can use `xs:string`. Therefore, translation is done directly using the same types that are defined in the XML Schema document.

4.4.1. Enumerations

Enumerations in XML Schema can be used to declare the possible values that an element can have. In ShEx, this is supported using the symbols '[' and ']'. The enclosed values are the possible values that the RDF object can take.

```
### XML Schema
<xs:simpleType name="PublicationType">
   <xs:restriction base="xs:NMTOKEN">
     <xs:enumeration value="Book"/>
     <xs:enumeration value="Magazine"/>
     <xs:enumeration value="Journal"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="pubType"
            ref="PublicationType"/>
<xs:attribute name="country"
              type="xs:NMTOKEN"
              fixed="US"/>

### ShEx
:pubType ["Book" "Magazine" "Journal"] ;
:country ["US"] ;
```

Listing 11: NMTokens mapping

4.4.2. Pattern

`Pattern` is used in XML Schema to define how a string value should be or what type of format is allowed. `Pattern` in ShEx uses a syntax similar to the JavaScript language except that backslash is required to be escaped, i.e., double backslash have to be used to be correctly escaped. Therefore, the conversion is a transformation between XML Schema and JavaScript Regular Expression syntaxes.

```
### XML Schema
<xs:simpleType name="SKU">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3}-[A-Z]{2}"/>
  </xs:restriction>
</xs:simpleType>
<xs:attribute name="partNum"
              type="SKU"
              use="required"/>
```

```
### ShEx
:partNum /\\d{3}-[A-Z]{2}/ ;
```

Listing 12: Pattern mapping

4.5. SimpleType

Simple types in XML Schema are based in XSD Types (see Section 4.4) and allow some enhancements like: restrictions, lists and unions. Translation into ShEx will use the same XSD Types, as ShEx supports them. Depending on the content, translation is performed following a different criteria which we detail bellow. For translation of restrictions, see Section 4.7.

4.5.1. List

Lists inside simple types define a way of creating collections of a base XSD type in XML Schema. These lists are supported in RDF using RDF Collections[2]. As previously discussed, there can be several approaches to represent ordered lists in RDF (see Section 4.3.1). A commonly accepted approach is the use of RDF lists: an edge point to the first element and another to the rest of the list which recursively follows the same structure until the `rdf:nil` element is declared to represent the end of the list. In this way, it is possible to create the desired list and preserve the order. Figure 1 shows how an RDF list is constructed for a better understanding of this section. Hence, translation into ShEx is made by using RDF lists and the use of recursion that defines a type with a pointer to itself in the `rdf:rest` edge.

```
### XML Schema
<xs:simpleType name="IntegerList">
  <xs:list itemType="xs:integer" />
</xs:simpleType>

### ShEx
<IntegerList> {
 rdf:first xs:integer ;
 rdf:rest @<IntegerList> OR [rdf:nil];
}
```

Listing 13: List mapping

[2]https://www.w3.org/TR/rdf11-mt/#rdf-collections

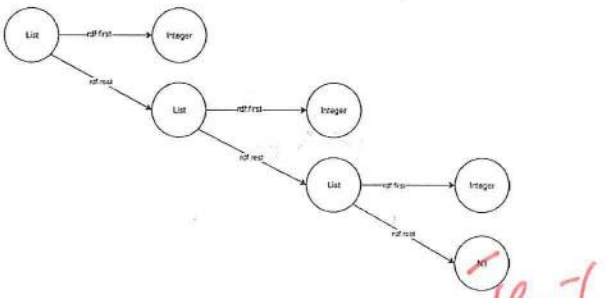

Fig. 1. Example of a RDF list construction

4.5.2. Union

Unions are the mechanism that XML Schema offers to make new types that are the union of two simple types. With this kind of disjunction, a new type which allows any value admitted by any of the members of the `union` is created. For the translation into ShEx we create a new type that is the combination of the types involved in the `union`.

```
### XML Schema
<xs:attribute name="fontsize">
 <xs:simpleType>
   <xs:union memberTypes="fontbynumber
                          fontbystringname"
   />
    </xs:simpleType>
</xs:attribute>

<xs:simpleType name="fontbynumber">
    <xs:restriction
        base="xs:positiveInteger">
        <xs:maxInclusive value="72"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="fontbystringname">
    <xs:restriction base="xs:string">
        <xs:enumeration value="small"/>
        <xs:enumeration value="medium"/>
        <xs:enumeration value="large"/>
    </xs:restriction>
</xs:simpleType>

### ShEx
:fontsize
  @:fontbynumber OR @:fontbystringname

:fontbynumber
   xs:positiveInteger MAXINCLUSIVE 72

:fontbystringname ["small"
                   "medium"
                   "large"
                  ]
```

Listing 14: Union mapping

4.6. ComplexContent and SimpleContent

Complex contents and simple contents are a way to define a new type from a base type using restrictions or extensions. The base type is the one that is used as a base for the restriction (or extension) clause and the new type is the one that is been restricted (or extended). Complex content allows to extend or restrict a base `complexType` with mixed content or elements only. Simple content allows to extend or restrict a `complexType` with character data or with a `simpleType`. For the translation into ShEx, the respective `restriction` or `extension` have to be taken into account to define the new type.

4.6.1. Restriction

Restrictions are used in XML Schema to restrict possible values of a base type. A new type can be defined using restrictions applied to a base type. Depending on how the type and the restrictions are defined, the translation strategies vary.

- Simple Content: If `simpleContent` is present XSD Facets/Restrictions must be used (see Section 4.7 for more information). When restricting using a `simpleType`, the transformation is done using the known base type (see Section 4.4) and putting some format restrictions depending on the base type. Translation into ShEx will be performed using the base type—ShEx supports the built-in XSD Types defined for XML Schema, therefore translation is done directly—and translating the XSD Facets as they are defined in every specific case, see Section 4.7.
- Complex Content: If `complexContent` is present, the base `complexType` is restricted using `group`, `all`, `choice`, `sequence`, `attributeGroup` or `attribute`. Complex content restriction will restrict allowed values and `element` type restrictions. This is a case of inheritance by restriction. For translation into ShEx, the `restriction` elements must be taken and transformed directly into a new shape that defines the resulting child shape[3].

[3] Future versions of ShEx are planning to include inheritance. See: https://github.com/shexSpec/shex/issues/50

4.6.2. Extension

With extensions in XML Schema, it is possible to define a new type as an extension of a previously defined one. This is a case of classic inheritance, where the child inherits its parent elements that are added to its own defined elements. Depending on the content, i.e., `complexContent` or `simpleContent`, different translation strategies can be used.

- Simple content: If `simpleContent` is present extension of the base type is performed by adding more attributes or attribute groups to the new type. Therefore, the translation into ShEx is made by the concatenation of both the type and its `extension` to create the new shape.
- Complex content: If `complexContent` is present extension of base type is performed by adding more attributes and elements to a new base one. Therefore, translation is done by combining the base type and its `extension` to create a new shape.

Restrictions and extensions in ShEx are not supported directly in the current version (i.e., ShEx has no support for extensions, restriction or inheritance) with the same semantics as XML Schema. Therefore, we use the normal syntax provided by ShEx and create the two resulting shapes from the respective `restriction` or `extension` as can be seen in Listing 15.

```
### XML Schema
<xs:simpleType name="mountainBikeSize">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small" />
    <xs:enumeration value="medium" />
    <xs:enumeration value="large" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="FamilyMountainBikes">
  <xs:simpleContent>
    <xs:extension base="mountainBikeSize">
      <xs:attribute name="familyMember">
       <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="child" />
          <xs:enumeration value="male" />
          <xs:enumeration value="female" />
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

### ShEx
:MountainBikeSize ["small" "medium" "large"]
```

```
:FamilyMountainBikes {
 :mountainBikeSize @:MountainBikeSize ;
 :familyMember ["child" "male" "female"];
}
```

Listing 15: Restrictions and extensions mapping, where extensions and restrictions are directly transformed into the equivalent shape

4.7. XSD Types Restrictions/Facets

4.7.1. Enumeration

`Enumeration` restrictions use a base type to restrict the possible values of a type. It is declared using a set of possible values. In ShEx this is defined using the '[' and ']' operators. The values that are allowed are enclosed inside the square brackets.

```
### XML Schema
<xs:simpleType name="Mountainbikesize">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType
  name="FamilyMountainBikeSizes">
  <xs:simpleContent>
    <xs:extension base="mountainbikesize">
       <xs:attribute name="familyMember"
                     type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType
    name="ChildMountainBikeSizes">
  <xs:simpleContent>
     <xs:restriction
          base="FamilyMountainBikeSizes" >
        <xs:enumeration value="small"/>
        <xs:enumeration value="medium"/>
     </xs:restriction>
  </xs:simpleContent>
</xs:complexType>

### ShEx
<MountainBikeSize> ["small" "medium" "large"]

<FamilyMountainBikes> {
 :mountainBikeSize @:MountainBikeSize ;
 :familyMember ["child" "male" "female"];
}

<ChildMountainBikeSizes>
   @:FamilyMountainBikes AND {
   :mountainBikeSize ["small" "medium"]
}
```

Listing 16: Enumeration mapping

4.7.2. Fraction digits

`FractionDigits` are used in XML Schema when a decimal type is defined (e.g., `xs:decimal`) and the number of decimal digits is desired to be restricted in the representation. ShEx supports this feature in a similar way as XML Schema. Hence, `FRACTIONDIGITS` keyword is used followed by the integer number of fraction digits that should be allowed.

```
### XML Schema
<xs:element name="itemValue">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:fractionDigits value="2"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

### ShEx
:itemValue xs:decimal FRACTIONDIGITS 2 ;
```

Listing 17: Fraction digits mapping

4.7.3. Length

`Length` is used to restrict the number of characters allowed in a string type. In ShEx this is supported with the `LENGTH` keyword, followed by the integer number that defines the desired length.

```
### XML Schema
<xs:element name="group">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

### ShEx
:group xs:string LENGTH 1 ;
```

Listing 18: Length mapping

4.7.4. Max Length and Min Length

Maximum and minimum length are used to restrict the number of characters allowed in a text type. But instead of restricting to a fixed number of characters, with these features restriction to a length interval is possible. In ShEx, the definitions of minimum and maximum length are made by using the MINLENGTH and MAXLENGTH keywords.

```
### XML Schema
<xs:element name="comments">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="1000"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

### ShEx
:comment xs:string
         MINLENGTH 1
         MAXLENGTH 1000;
```

Listing 19: Max length and min length mapping

4.7.5. Max-min exclusive and max-min inclusive

These features allow to restrict number types to an interval of desired values. EXCLUSIVE restricts the use of the given value and INCLUSIVE does not restrict the use of given value. This is the same notion as in open and closed intervals. In ShEx, these features are supported directly.

```
### XML Schema
<xs:element name="cores">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minExclusive value="0"/>
      <xs:maxExclusive value="9"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="coresOpenInterval">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

### ShEx
:cores xs:integer
       MINEXCLUSIVE 0
       MAXEXCLUSIVE 9 ;
:coresOpenInterval xs:integer
                   MININCLUSIVE 1
                   MAXINCLUSIVE 8 ;
```

Listing 20: Max exclusive, min exclusive, min inclusive and max inclusive mapping

4.7.6. Total digits

This feature allows to restrict the total number of digits permitted in a numeric type. In ShEx this is possible using TOTALDIGITS keyword.

```
### XML Schema
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:totalDigits value="3"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

### ShEx
:age xs:integer
     TOTALDIGITS 3 ;
```

Listing 21: Total digits mapping

4.7.7. Whitespace

WhiteSpace allows to specify how white spaces on strings are handled. In XML Schema, there are three options:

- Preserve: This option will not remove any white space character from the given string.
- Replace: This option will replace all white space characters (line feeds, tabs, spaces and carriage returns) with spaces.
- Collapse: This option will remove all white spaces characters:
  - Line feeds, tabs, spaces and carriage returns are replaced with spaces.
  - Leading and trailing spaces are removed.
  - Multiple spaces are reduced to a single space.

In ShEx, whiteSpace options are not supported. Their behaviour could be simulated using semantic actions (see Listing 22).

```
### XML Schema
<xs:complexType name="whiteSpaces">
 <xs:all>
  <xs:element name="preserve">
    <xs:simpleType>
```

```
      <xs:restriction base="xs:string">
        <xs:whiteSpace
           value="preserve"/>
        </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="replace">
    <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:whiteSpace
            value="replace"/>
         </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="collapse">
   <xs:simpleType>
     <xs:restriction base="xs:string">
        <xs:whiteSpace
          value="collapse"/>
     </xs:restriction>
   </xs:simpleType>
 </xs:element>
</xs:all>
</xs:complexType>

### ShEx
<whiteSpaces> {
  :preserve xs:string ;
  :replace  xs:string
   %js{
     _.o.lex = _.o.lex
        .replace("/\r|\n|\r\n|\s/g", " ");
    return true;
   }
   % ;
   :collapse xs:string
   %js{
     var replacedText = _.o.lex
        .replace("/\r|\n|\r\n|\s/g", " ");
     _.o.lex = replacedText.trim();
     return true;
   }
  %
}
```

Listing 22: WhiteSpace mapping

```
### XML Schema
<xs:element name="Person"
            maxOccurs="unbounded">
  <xs:complexType>
    <xs:all>
       <xs:element name="name"
                   type="xs:string" />
         <xs:element name="surname"
                     type="xs:string" />
           <xs:element name="id"
                       type="xs:integer" />
    </xs:all>
  </xs:complexType>
  <xs:unique name="onePersonPerID">
     <xs:selector xpath="."/>
     <xs:field xpath="id"/>
  </xs:unique>
</xs:element>

### ShEx
%js{
    var ids = [];
    return true;
}
%
<Person> {
  :name xs:string ;
  :surname xs:string ;
  :id xs:integer
  %js{ if(ids.indexOf(_.o.lex) >= 0)
         return false;
       ids.push(_.o.lex);
       return true;
  }%
}
```

Listing 23: Unique mapping

4.7.8. *Unique*

`Unique` is used in XML Schema to define that an element of some type is unique, i.e., there can not be the same values among elements defined in the constraint. This is useful for cases like IDs, where a unique ID is the way to identify an element. Nowadays, ShEx does not support `Unique` function but it is expected to be supported in future versions. As a temporal solution, semantic actions could be used to implement this kind of constraint.

5. XMLSchema2ShEx prototype

In addition to the proposed mappings from XML Schema to Shape Expressions, for the sake of hypothesis demonstration, a prototype has been developed that uses a subset of the presented mappings and converts from a given XML Schema input to a ShEx output.

The prototype has been developed in Scala and it is available online[4]. It is a work-in-progress implementation, so not all the mappings are supported yet (see Table 1).

The tool is built on top of Scala parser combinators [21]. Once the XML Schema input is analysed

[4] https://github.com/herminiogg/XMLSchema2ShEx

Table 1

Supported and pending of implementation features in XMLSchema2ShEx prototype. * Not supported in ShEx 2.0.

Supported features	Complex type, Simple type, All, Attributes, Restriction, Element, Max exclusive, Min exclusive, Max inclusive, Min inclusive, Enumeration, Pattern, Cardinality
Pending implementation	Choice, List, Union, Extension, Fraction Digits, Length, Max Length, Min Length, Total digits, Whitespace*, Unique*

and verified, it is converted to ShEx based on different elements and types declared on it. These conversions are made recursively and printed to the ouput in ShEx Compact Format (ShExC).

The example presented in Listing 24 is used to ensure that the prototype can work and do the transformation as expected. This example includes complex types, attributes, elements, simple types and patterns among others. Therefore, complex types are converted to shapes, elements and attributes to triple predicates and objects, restrictions (max/minExclusive and max/minInclusive) to numeric intervals, cardinality attributes to ShEx cardinality and so on. Although it is a small example, it has the structure of typical XML Schemas used nowadays and the prototype can convert it properly as it is stated in the example conversion below.

```
### XML Schema
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://tempuri.org/po.xsd"
 xmlns="http://tempuri.org/po.xsd"
 elementFormDefault="qualified">

 <xs:element name="purchaseOrder"
             type="PurchaseOrderType"/>

  <xs:element name="comment"
              type="xs:string"/>

  <xs:complexType name="PurchaseOrderType">
    <xs:all>
       <xs:element name="shipTo"
                   type="USAddress"/>
       <xs:element name="billTo"
                   type="USAddress"/>
       <xs:element ref="comment"
                   minOccurs="0"/>
       <xs:element name="items"
                   type="Items"/>
    </xs:all>
    <xs:attribute name="orderDate"
                  type="xs:date"/>
  </xs:complexType>

  <xs:complexType name="USAddress">
    <xs:all>
       <xs:element name="name"
                   type="xs:string"/>
       <xs:element name="street"
                   type="xs:string"/>
       <xs:element name="city"
                   type="xs:string"/>
       <xs:element name="state"
                   type="xs:string"/>
       <xs:element name="zip"
                   type="xs:integer"/>
    </xs:all>
    <xs:attribute name="country"
                  type="xs:NMTOKEN"
                  fixed="US"/>
</xs:complexType>

<xs:complexType name="Items">
  <xs:all>
   <xs:element name="item"
               minOccurs="0"
               maxOccurs="unbounded">
     <xs:complexType>
      <xs:all>
       <xs:element
               name="productName"
               type="xs:string"/>
       <xs:element
               name="quantity">
         <xs:simpleType>
            <xs:restriction
               base="xs:positiveInteger">
              <xs:maxExclusive
                   value="100"/>
            </xs:restriction>
         </xs:simpleType>
         </xs:element>
           <xs:element name="USPrice"
             type="xs:decimal"/>
         <xs:element ref="comment"
              minOccurs="0"/>
         <xs:element name="shipDate"
              type="xs:date" minOccurs="0"/>
     </xs:all>
   <xs:attribute name="partNum" type="SKU"
        use="required"/>
   </xs:complexType>
   </xs:element>
  </xs:all>
</xs:complexType>

<xs:simpleType name="SKU">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3}-[A-Z]{2}"/>
  </xs:restriction>
</xs:simpleType>
```

```
</xs:schema>

### ShEx
PREFIX : <http://www.example.com/>
PREFIX
    xsd: <http://www.w3.org/2001/XMLSchema#>

<Items> {
  :item          @<item> *  ;
}
<item> {
  :productName   xsd:string  ;
  :quantity      xsd:positiveInteger
                 MAXEXCLUSIVE 100 ;
  :USPrice       xsd:decimal  ;
  :comment       xsd:string ?  ;
  :shipDate      xsd:date ?  ;
  :partNum       /\\d{3}-[A-Z]{2}/ ;
}
<PurchaseOrderType> {
  :shipTo      @<USAddress>  ;
  :billTo      @<USAddress> ;
  :comment     xsd:string ?  ;
  :items       @<Items>  ;
  :orderDate   xsd:date ;
}
<USAddress> {
  :name        xsd:string ;
  :street      xsd:string  ;
  :city        xsd:string  ;
  :state       xsd:string ;
  :zip         xsd:integer ;
  :country     ["US"]  ;
}
```

Listing 24: XML Schema to ShEx example

5.1. Validation example

```
### XML
<?xml version="1.0"?>
<purchaseOrder
  xmlns="http://tempuri.org/po.xsd"
  orderDate="1999-10-20">
  <shipTo country="US">
     <name>Alice Smith</name>
     <street>123 Maple Street</street>
     <city>Mill Valley</city>
     <state>CA</state>
     <zip>90952</zip>
  </shipTo>
  <billTo country="US">
     <name>Robert Smith</name>
     <street>8 Oak Avenue</street>
     <city>Old Town</city>
     <state>PA</state>
     <zip>95819</zip>
  </billTo>
  <comment>
     Hurry, my lawn is going wild!
  </comment>
  <items>
  <item partNum="872-AA">
     <productName>
       Lawnmower
     </productName>
     <quantity>1</quantity>
     <USPrice>148.95</USPrice>
      <comment>
               Confirm this is electric
      </comment>
      </item>
  <item partNum="926-AA">
    <productName>
        Baby Monitor
    </productName>
    <quantity>1</quantity>
    <USPrice>39.98</USPrice>
    <shipDate>1999-05-21</shipDate>
  </item>
 </items>
</purchaseOrder>

### RDF
:order1
  :shipTo [
  :name "Alice Smith" ;
  :street "123 Maple Street" ;
  :city "Mall Valley" ;
  :state "CA" ;
  :zip 90952 ;
  :country "US"
  ] ;
 :billTo [
  :name "Robert Smith" ;
  :street "8 Oak Avenue" ;
  :city "Old Town" ;
  :state "PA" ;
  :zip 95819 ;
  :country "US"
 ] ;
 :comment "Hurry, my lawn is going wild!";
 :items [
    :item [
      :productName "Lawnmower" ;
      :quantity "1"^^xsd:positiveInteger ;
      :USPrice 148.95 ;
      :comment "Confirm this is electric";
      :partNum "872-AA"
    ] ;
    :item [
      :productName "Baby Monitor" ;
      :quantity "1"^^xsd:positiveInteger ;
      :USPrice 39.98 ;
      :shipDate "1999-05-21"^^xsd:date ;
      :partNum "926-AA"
    ] ;
 ];
```

```
:orderDate "1999-10-20"^^xsd:date .
```

Listing 25: XML to RDF example

Once conversion from XML Schema to ShEx is done, it must be verified that the same validation that was performed on XML data using XML Schema, but now on RDF data using ShEx, is working equivalently. Therefore, translation of a valid XML to RDF is executed which is presented in Listing 25. The conversion presented in the snippet is a possible one that uses blank nodes to represent the nested types. This is done for avoiding to create a fictional node every time a triple is pointing to another triple (in other words, every time it has a nested type). The conversion was performed following similar equivalences to those proposed in the mappings. That is, complex types to triple subjects or predicates, simple types to triple objects, cardinality translated directly and so on.

For RDF validation using ShEx there are various implementations in different programming languages that are being developed[5]. One of these implementations is made in Scala by one of the authors of this paper and it is available online[6].

Using the examples given above the validation can be performed with the mentioned tool which allows the RDF and the ShEx inputs in various formats and then the option to validate the RDF against ShEx or SHACL schema. As seen in Figure 2, validation is performed trying to match the shapes with the existing graphs, whenever the tool matches a pattern it shows the evidence in green and a short explanation of why this graph has matched.

This kind of transformations can work in most of the cases. However, there is a premise—which is in line with one of the defined research questions—that must be satisfied before generating a valid conversion. In case of XML files with ambiguous content models where some files can be transformed in different ways and correct validation of converted data cannot be guaranteed. This problem comes in two dimensions: from XML to RDF, trying to maintain the same semantics with different models; and for schema generation, trying to create a schema that describes all the possibilities. Nevertheless, if this ambiguity problem is previously solved or is not present, the conversion can be validated using the proposed techniques.

[5] A list of ShEx implementations is available at: https://shex.io

[6] http://shaclex.herokuapp.com

6. Conclusions and Future work

In this work, a possible set of mappings between XML Schema and ShEx has been presented. With this set of mappings, automation of XML Schema conversions to ShEx is a new possibility which is demonstrated by the prototype that has been developed and presented in this paper. Using an existing validator helped to demonstrate that an XML and its corresponding XML Schema are still valid when they are converted to RDF and ShEx, although some ambiguity premises must be satisfied.

One future line of work that should be tackled is the loss of semantics: with this kind of transformations some of the elements could not be converted back to their XML Schema origin. Nevertheless, it is a difficult problem due to the difference between ShEx and XML data models and it would involve some sort of modifications and additions to the ShEx semantics (like the inheritance case).

To cover all the business cases and make this solution more compatible, there is the need to create mappings for Schematron and Relax NG as a future work. This future line should be handled with structure in mind. Relax NG is grammar-based but Schematron is rule based, which will make conversion from Relax NG to ShEx more straightforward than from Schematron, as ShEx is also based in grammars. Another line of future work is to adapt the presented mappings to SHACL: most of the mappings follow a similar structure. Moreover, the rule-based Schematron conversion seems more plausible using the advanced SHACL-Sparql features.

With the present work, validation of existing transformations between XML and RDF is now possible and convenient. This kind of validations makes the transformed data more reliable and trustworthy and it also facilitates migrations from non-semantic data formats to semantic data formats.

However, a big path should be travelled. Conversions from other formats (such as JSON Schema, DDL, CSV Schema, etc.) should also be treated and encouraged to permit a migration to a new set of semantic-aware and interoperable data.

References

[1] Steve Battle. Gloze: XML to RDF and back again. In *Proceedings of the First Jena User Conference*, 2006.

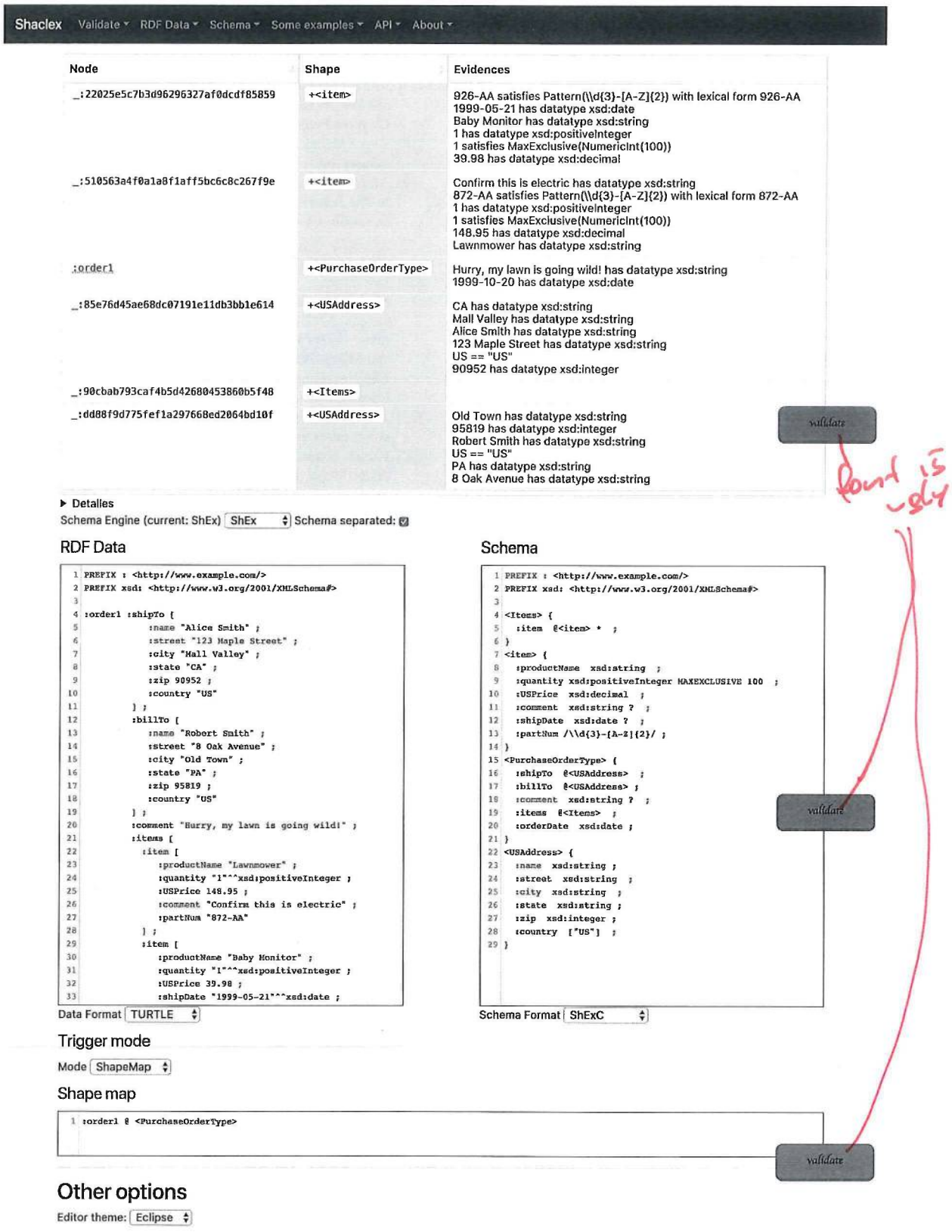

Fig. 2. Validation result using Shaclex validator. The RDF data is entered in the left text area whereas the ShEx schema is entered on the right text area. In the bottom, a ShapeMap is declared to make the validator know where and how to begin the validation, in this case we commanded to validate <http://www.example.com/order1> node with <PurchaseOrderType> shape. In the top of the page, the result is shown detailing how each node was validated and what are the evidences or failures for the validation. A link to the validation example can be found in Supplementary Material.

:order1 :—
be consistent!

[2] Diego Berrueta, Jose Emilio Labra Gayo, and Ivan Herman. XSLT + SPARQL: Scripting the semantic web with SPARQL embedded into XSLT stylesheets. In *4th Workshop on Scripting for the Semantic Web, Tenerife*, 2008.

[3] Geert Jan Bex, Frank Neven, and Jan den Bussche. DTDs versus XML schema: a practical study. In *Proceedings of the 7th international workshop on the web and databases: colocated with ACM SIGMOD/PODS 2004*, pages 79–84. ACM, 2004.

[4] Paul V Biron, Ashok Malhotra, World Wide Web Consortium, et al. XML Schema part 2: Datatypes, 2004.

[5] Stefan Bischof, Stefan Decker, Thomas Krennwallner, Nuno Lopes, and Axel Polleres. Mapping between RDF and XML with XSPARQL. *Journal on Data Semantics*, 1(3):147–185, 2012.

[6] Iovka Boneva, Jose Emilio Labra Gayo, and Eric Prud'hommeaux. Semantics and validation of shapes schemas for rdf. In Claudia d'Amato, Miriam Fernandez, Valentina Tamma, Freddy Lecue, Philippe Cudré-Mauroux, Juan Sequeda, Christoph Lange, and Jeff Heflin, editors, *International Semantic Web Conference*, volume 10587 of *Lecture Notes in Computer Science*, pages 104–120. Springer Verlag, October 2017.

[7] James Clark and Makoto Murata. Relax NG specification, 2001.

[8] Davy Van Deursen, Chris Poppe, Gäetan Martens, Erik Mannens, and Rik Van de Walle. XML to RDF Conversion: A Generic Approach. In *Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS '08. International Conference on*, pages 138–144, Washington, nov 2008.

[9] Nick Drummond, Alan L Rector, Robert Stevens, Georgina Moulton, Matthew Horridge, Hai Wang, and Julian Seidenberg. Putting OWL in order: Patterns for sequences in OWL. In *OWLED*, 2006.

[10] TEI Consortium, eds. TEI P5: Guidelines for Electronic Text Encoding and Interchange, 2017.

[11] Matthias Ferdinand, Christian Zirpins, and David Trastour. *Lifting XML Schema to OWL*, pages 354–358. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[12] P. N. Fox, R. Mead, M. Talbot, and J. D. Corbett. Data management and validation. In *Statistical Methods for Plant Variety Evaluation*, pages 19–39, Dordrecht, 1997. Springer Netherlands.

[13] Rick Jelliffe. The Schematron: An XML structure validation language using patterns in trees. 2001.

[14] Sam P. Kaipa, Ashish. Rahurkar, Pradeep C. Bollineni, and Ashutosh Arora. Mapping XML Schema components to qualified Java components, March 20 2007. US Patent 7,194,485.

[15] Holger Knublauch. SPIN - Modeling Vocabulary. http://www.w3.org/Submission/spin-modeling/, 2011.

[16] Holger Knublauch and Dimitris Kontokostas. *Shapes Constraint Language (SHACL)*. W3C Proposed Recommendation, June 2017.

[17] Jose Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, and Dimitri Kontokostas. *Validating RDF Data*. Morgan and Claypool Publishers, 2017.

[18] Sergei Melnik and Stefan Decker. Representing order in rdf. January 2001.

[19] Albert Meroño-Peñuela. Semantic web for the humanities. In *The Semantic Web: Semantics and Big Data: 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pages 645–649. Springer Berlin Heidelberg, 2013.

[20] Igor Miletic, Marko Vujasinovic, Nenad Ivezic, and Zoran Marjanovic. Enabling Semantic Mediation for Business Applications: XML-RDF, RDF-XML and XSD-RDFS transformations. In Ricardo J Gonçalves, Jörg P Müller, Kai Mertins, and Martin Zelm, editors, *Enterprise Interoperability II: New Challenges and Approaches*, pages 483–494. Springer London, London, 2007.

[21] Adriaan Moors, Frank Piessens, and Martin Odersky. Parser combinators in Scala. 2008.

[22] Patrick Neubauer, Alexander Bergmayr, Tanja Mayerhofer, Javier Troya, and Manuel Wimmer. XMLText: from XML Schema to Xtext. In *SLE*, 2015.

[23] Falco Nogatz and Thom Frühwirth. *From XML Schema to JSON Schema-Comparison and Translation with Constraint Handling Rules*. Ulm, Germany, 2013.

[24] Giuseppe Della Penna, Antinisca Di Marco, Benedetto Intrigila, Igor Melatti, and Alfonso Pierantonio. Interoperability mapping from XML Schemas to ER diagrams. *Data Knowl. Eng.*, 59:166–188, 2006.

[25] Eric Prud'hommeaux, Iovka Boneva, Jose Emilio Labra Gayo, and Gregg Kellogg. Shape expressions language 2.0, 2017.

[26] Eric Prud'hommeaux, Jose Emilio Labra Gayo, and Harold Solbrig. Shape expressions: an RDF validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems*, pages 32–40. ACM, 2014.

[27] John Simpson and Susan Brown. From XML to RDF in the Orlando Project. In *2013 International Conference on Culture and Computing, Kyoto, Japan*, pages 194–195, Sept 2013.

[28] Timo Sztyler, Jakob Huber, Jan Noessner, Jaimie Murdock, Colin Allen, and Mathias Niepert. LODE: Linking digital humanities content to the web of data. In *IEEE/ACM Joint Conference on Digital Libraries, London, UK*, pages 423–424, Sept 2014.

[29] Jiao Tao, Evren Sirin, Jie Bao, and Deborah L McGuinness. Integrity constraints in OWL. 2010.