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 field where researchers among the Semantic Web are putting their effort. However, migration to new formats and standards comes at a price. In order to facilitate and alleviate this transformation, this paper proposes a set of mappings that can be used to convert between XML Schema and ShEx—one of the new RDF validation languages—. Moreover, this paper presents a prototype that supports a small subset of the mappings proposed in it and a example of a XML Schema converted to ShEx with this prototype. This work and the development of other formats mappings could drive to a new era of semantic-aware and interoperable data.

Keywords: ShEx, XML Schema, RDF, XML, RDF validation, Shape Expressions, Semantic Web, data conversion, format mapping, semantic-aware data

1. Introduction

Validation is one of the key areas when normalisation and confidence are desired. Moreover, normalisation is desired as a way of making a dataset more trustworthy and even more useful to possible users because of its predictable schema. Validation can excel data cleansing, querying and standardisation. Therefore, validation is a key field of data management.

XML Schema was born as a language to make validation of XML possible and more convenient than using DTDs [3]. With XML Schema, developers can define the structure, constraints and documentation of a XML vocabulary. Alongside the appearance of DTDs, XML Schema and other alternatives (such as Relax NG [6] and Schematron [9]), new possibilities come to scene such as conversion to OWL [8] in order to convert this information into Semantic Web compatible version.

As XML has its own schema language—or languages—it was some that RDF lacked. Some alternatives were OWL and RDF Schema; however, they do not cover completely what XML Schema does with XML [16]. For this purpose, Shape Expressions was proposed to fulfil this requirement.

As many documents and data are persisted in XML and need of migration or interoperability is nowadays more pressing, many authors have proposed conversions from XML to RDF [11] [7] [1] [4]. These conversions enable users to migrate their data to newer and more modern technologies. However, a key and lacking process when converting XML to RDF is the validation process. How to be sure that the conversion has been done effectively and that both versions—in different languages—are defining the same nature. How to migrate all the effort put in validation mark-up and preserve this functionality in the new platform. These questions are some of which we desire to answer in the present work.

Therefore, an alternative 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 describes how to convert each element from XML Schema to ShEx notation, section 3 describes a possible set of mappings between XML Schema and RDF,

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

section 4 presents a prototype used to validate a subset of previously presented mappings and how this conversion works against existing RDF validators and section 5 draws some conclusions and future lines of work and improvement.

2. Background

Conversions between two formats is an active field where researchers put their effort. In XML community, many conversions to RDF, and backwards, have been proposed using different techniques: in [11] authors describe their experience on developing this transformation for business to business industry, in [7] an ontology based transformation is described, in [1] they use this approach to solve the lift problem, in [4] authors describe a transformation supported on SPARQL and in [2] a transformation from RDF to other kind of formats, including XML, is proposed using XSLT.

However, data validation is a key question as it has been previously stated in this paper. Therefore, transformation between schemas is another important and active field: in [14] a dictionary of transformations is defined based on similarities between XML and JSON schemas, in [10] authors patented a mechanism to convert XML Schema components to Java components, in [15] an algorithm which converts from XML schemas to ER diagrams is proposed and in [13] authors propose conversion from XML Schema to xtext to bring more functionalities to XML Schema based DSLs.

Another approach is to take a domain model as the main representation and then make the needed transformations between the model and other formats (e.g., XML Schema, ShEx, JSON Schema and so on) as it has been made on FHIR specification[1].

Based in the Semantic Web various approaches could be taken: transforming XML Schema to ontologies: in [8] authors define a set of mapping to transform XML Schema to OWL and transforming XML Schema to RDF Schema [11].

However, neither OWL nor RDF Schema capture all the constraints that are supported by XML Schema and that RDF needs as it is stated by [16]. Therefore, various languages were proposed to perform this operation. On one hand Shapes Constraint Language (SHACL)[2] that is proposed by the W3C Data Shapes Working Group and Shape Expressions (ShEx) [17] proposed by the W3C Shape Expressions Community Group. In this paper, ShEx is used to describe the transformations due to its support for recursion whereas in SHACL recursion depends on the implementation being used. However, transformations to SHACL are feasible and interesting for future work because it has been proposed as a W3C recommendation and it can be simulated by target declarations.

To our knowledge, due to its recent appearance, no XML Schema to ShEx conversion has been proposed. In this paper, a transformation from XML Schema to ShEx is proposed and how each element could be handled on its individual translation.

3. 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. Along this section, different XML Schema elements and what a possible conversion to ShEx could be are described. Starting from the example and then expanding with advanced XML Schema and ShEx features. All examples are using the default prefix : for URIs. It is intended to be replaced by different prefixes depending on the required namespace.

3.1. Element

Elements should be treated as a triple terminal, i.e., we should convert it as a terminal expression of a shape one.

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

```
:birthday xs:date ;
```

Name attribute is used as the end of the URI in the predicate and the type is transcribed directly, as ShEx use the XSD types. If ref attribute is present, type should be defined somewhere to link the corresponding type or shape. When an element type is a complex type, the type should be referenced to a new shape where the complex type was converted (see 3.3 section to expand information on how to convert a complex type to a shape).

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

[1] https://www.hl7.org/fhir/
[2] https://www.w3.org/TR/shacl/

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

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

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

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

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

3.2. Attribute

Attributes should be treated as elements in ShEx. ShEx makes no difference between an attribute and an element because this is only part of the XML semantics. Therefore, transformation of an attribute could be done using its name and type as performed with an element (see 3.1 section).

3.3. ComplexType

Complex types are translated directly to ShEx's shapes. The name of the complex type will be the name of the shape to which elements can refer to. Complex types can be compound of different statements. Therefore, detailed transformation of each possibility is presented below.

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

```
<PurchaseOrderType> {
    ...
}
```

3.3.1. Sequence

While sequences in XML Schema define sequential order of elements, in ShEx this is more complicated due to the RDF graph schema. Therefore, there are two main forms of converting a sequence: considering a sequence as an unordered set of elements—useful as many users use sequence like all clause [5]—or an ordered set of elements preserving the semantics of XML.

The following example shows two different conversions covering each of the possibilities that were mentioned before.

```
<xs:complexType name="address">
    <xs:sequence>
        <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:decimal"/>
    </xs:sequence>
</xs:complexType>
```

```
<address> {
    :name   xs:string ;
    :street   xs:string   ;
    :city   xs:string   ;
    :state   xs:string ;
    :zip   xs:decimal ;
}
```

```
<address> {
    rdf:first @<name> ;
    rdf:rest @<i1> ;
}

<i1> {
    rdf:first @<street> ;
    rdf:rest @<i2> ;
}

<i2> {
    rdf:first @<city> ;
    rdf:rest @<i3> ;
}

<i3> {
    rdf:first @<state> ;
    rdf:rest @<i4> ;
}
```

```
<i4> {
    rdf:first @<zip> ;
    rdf:rest [ rdf:nil ] ;
}

<name> {
    :name xs:string ;
}

<street> {
    :street xs:string ;
}

<city> {
    :city xs:string ;
}

<state> {
    :state xs:string ;
}

<zip> {
    :zip xs:decimal ;
}
```

3.3.2. Choice

Choices in XML Schema are the OR logic operator to select between a value or another one. This operator is supported in ShEx using the brackets '(' and ')' and the '|' operator. The object and predicate of the RDF statement must be one of the enclosed ones. Therefore, translation is performed as showed in the following snippet:

```
<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>
```

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

3.3.3. All

While sequences were an ordered set of elements, all is instead a set of unordered elements. Indeed, all is a better representation of ShEx elements of a shape. Therefore, transformation is done following the same procedure used for the unordered option sequence (see 3.3.1 section for more details).

```
<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:decimal"/>
</xs:all>
```

```
:name  xs:string ;
:street  xs:string  ;
:city  xs:string  ;
:state  xs:string ;
:zip  xs:decimal ;
```

3.4. SimpleType

Simple types in XML Schema are based in XSD Types (see 3.7 section) 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. For translation of restrictions see 3.6 section.

3.4.1. 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[3]. Hence, translation into ShEx is made by using RDF Collections vocabulary.

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

```
<IntegerList> {
    rdf:first xs:integer ;
    rdf:rest @<IntegerList> ;
}
```

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

3.4.2. Union

Unions are the way that XML Schema offers to make new types that are the conjunction of two simple types. With this kind of conjunction, a new type which allows any value admitted by each one of the members of the union is created. Therefore, for the translation into ShEx a new type that is the combination of the shapes involved into the union is defined.

```
<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>
```

```
(:fontsize xs:positiveInteger MaxExclusive 72
|
:fontsize ["Small" "Medium" "Large"] ) ;
```

3.5. ComplexContent and SimpleContent

Complex contents and simple contents, in XML Schema, are a way to define a new type from a base type using restrictions and extensions. Complex contents for complex types and simple contents for simple types. Hence, for the translation into ShEx, the respective restriction or extension should be taken into account to define the new type.

3.5.1. Restriction

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

- Simple content: If simple content is present XSD Facets/Restrictions must be used (see 3.6 section for more information). When a simple type is restricted transformation is done using the known base type (see 3.7) and putting some format restrictions depending on the base type. Translation into ShEx will be performed using the base type—ShEx uses the same XSD Types that are defined for XML Schema, therefore translation is done directly—and translating the XSD Facets as they are defined in every specific case, see 3.6.
- Complex content: If complex content is present, base complex type is restricted using group, all, choice, sequence, attribute groups or attribute. Complex content restriction will restrict allowable values and element type restrictions. This 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 child shape. [4].

3.5.2. Extension

With extensions in XML Schema it is possible to define a new type as an extension of other one previously defined. This is a case of extending inheritance, where the child inherits its parent elements plus its own defined elements. Depending on the content, i.e., complex content or simple content, different translation strategies are used:

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

Restrictions and extensions in ShEx are not supported directly (i.e., ShEx has no support for extensions, restriction or inheritance in any way) 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.

[4] In future versions of ShEx a way of inheritance will be supported. See: https://github.com/shexSpec/shex/issues/50

```
<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:restriction base="xs:string">
                <xs:enumeration
                    value="child" />
                <xs:enumeration
                    value="male" />
                <xs:enumeration
                    value="female" />
            </xs:restriction>
        </xs:attribute>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

```
FamilyMountainBikes {
    :mountainBikeSize [Small Medium Large] ;
    :familyMember [child male female] ;
}
```

3.6. XSD Types Restrictions/Facets

3.6.1. Enumeration

Enumeration restrictions use a base type to restrict the possible values. Normally it is a set of possible values. In ShEx this is defined using the '[' and ']' operators. Inside the square brackets are the values that are allowed.

```
<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>
```

```
FamilyMountainBikeSizes {
    :mountainBikeSize [Small Medium Large] ;
    :familyMember xs:string ;
}

ChildMountainBikeSizes {
    :mountainBikeSize [Small Medium] ;
    :familyMember xs:string ;
}
```

3.6.2. Fraction digits

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

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

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

3.6.3. Length

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

```
<xs:element name="group">
```

```
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="1"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

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

3.6.4. Max Length and Min Length

Max length and min 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 permitted. In ShEx definition of min and max length is made by using the MINLENGTH and MAXLENGTH keywords.

```
<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>
```

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

3.6.5. Max exclusive, min exclusive, min inclusive and max inclusive

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

```
<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>
```

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

3.6.6. Total digits

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

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

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

3.6.7. Whitespace

Whitespaces allow 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 white spaces options are not supported. In order to simulate the same behaviour it is possible to use semantic actions.

3.6.8. Unique

Unique is used in XML Schema to define what element of a type is unique, i.e., they cannot have same values among them. This is very useful for cases like IDs where a unique ID is the way to identify an ele-

ment. 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.

3.6.9. *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. Therefore, transformation of minOccurs and maxOccurs in the previous defined cardinality marks is done as showed in the following example.

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

```
:name xs:string * ;
:name xs:string + ;
:name xs:string ? ;
:name xs:string {4, 10} ;
```

3.7. *XSDTypes*

XSD types are used directly on ShEx. Therefore, translation is done directly using the same types that are defined in the XML Schema document.

3.7.1. *XS:NMTOKEN*

NMTokens on XML Schema are used to define possible values that a type could take. In ShEx this is supported using the symbols '[' and ']'. Enclosed values are the possible values that the RDF object could take.

```
<xs:simpleType name="PublicationType">
    <xs:restriction base="xsd:NMTOKEN">
        <xs:enumeration value="Book"/>
        <xs:enumeration value="Magazine"/>
        <xs:enumeration value="Journal"/>
        <xs:enumeration value="Online"/>
    </xs:restriction>
</xs:simpleType>

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

Table 1

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

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

```
:pubType [Book Magazine Journal Online] ;
:country [US] ;
```

3.7.2. *Pattern*

Patterns are used in XML Schema to define how a string value should be or what type of format is allowed. Patterns in ShEx use the same expressions except that backslash is required to be doubled, i.e., double backslash to be correctly escaped.

```
<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"/>
```

```
:partNum xs:string PATTERN \\d{3}-[A-Z]{2} ;
```

4. XMLSchema2ShEx prototype

Although proposed mappings between XML Schema and Shape Expressions in the previous section, 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 equivalent output. This prototype is developed in Scala and it is available online[5].

The tool is built on top of Scala parser combinators [12] which grammar could be seen in Supplementary

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

Material. Once the XML Schema input is analysed and verified it is converted to ShEx ouput based on the different outputs and linked elements declared in the file. These conversions are made recursively and printed to the ouput in ShEx Compact Format (ShExC) which is the output format supported by this tool.

The example presented below is used to prove that the prototype could 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 terminals, restrictions and cardinality attributes to triple 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.

```
<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:sequence>
            <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:sequence>
        <xs:attribute name="orderDate"
            type="xs:date"/>
    </xs:complexType>

    <xs:complexType name="USAddress">
        <xs:sequence>
            <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:decimal"/>
        </xs:sequence>
        <xs:attribute name="country"
            type="xs:NMTOKEN"
            fixed="US"/>
    </xs:complexType>

    <xs:complexType name="Items">
    <xs:sequence>
    <xs:element name="item"
        minOccurs="0"
        maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
            <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:sequence>
    <xs:attribute name="partNum" type="SKU"
        use="required"/>
    </xs:complexType>
    </xs:element>
    </xs:sequence>
    </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>
```

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

<Items> {
    :item  @<item> *  ;
}
<item> {
    :productName  xsd:string  ;
    :quantity  xsd:positiveInteger  [1, 99]  ;
    :USPrice  xsd:decimal  ;
    :comment  xsd:string ? ;
}
```

```
    :shipDate  xsd:date ? ;
    :partNum xsd:string
        PATTERN "\\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"]  ;
}
```

4.1. Validation example

```
<?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>
```

```
: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 ;
                :USPrice 148.95 ;
                :comment "Confirm␣this␣is␣electric" ;
                :partNum "872-AA"
            ] ;
            :item [
                :productName "Baby␣Monitor" ;
                :quantity 1 ;
                :USPrice 39.98 ;
                :shipDate "1999-05-21"^^xsd:date ;
                :partNum "926-AA"
            ] ;
        ];
        :orderDate "1999-10-20"^^xsd:date .
```

Once conversion from XML Schema input to ShEx output 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 properly. Therefore, translation of a valid XML to RDF is executed which is presented in the above snippet. The conversion presented in the snippet is a possible conversion that uses bnodes to represent the nested types for the example simplicity.

For RDF validation using ShEx there are various implementations in different programming languages that are being developed by other researchers in the community. 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 input and the ShEx input in various formats and then the option to validate the RDF against the ShEx or SHACL schema. As seen in Figure 1, validation is performed by trying to match the shapes with the existing graphs, whenever the tool matches a pattern it shows the coincidence in green and a short explanation of why this graph has matched.

5. 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.

One future line 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 Schema semantics and it would involve some sort of modifications and additions in ShEx semantics (like the inheritance case).

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

[2] Diego Berrueta, Jose E Labra, 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.

[6] http://shaclex.herokuapp.com/validate?schemaEmbedded=false

[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] 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.

[5] Iovka Boneva, Radu Ciucanu, and Slawomir Staworko. Simple schemas for unordered xml. In *16th International Workshop on the Web and Databases (WebDB)*, 2013.

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

[7] D V Deursen, C Poppe, G Martens, E Mannens, and R V d. 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, nov 2008.

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

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

[10] S.P. Kaipa, A. Rahurkar, P.C. Bollineni, and A. Arora. Mapping xml schema components to qualified java components, March 20 2007. US Patent 7,194,485.

[11] I Miletic, M Vujasinovic, N Ivezic, and Z 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.

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

[13] Patrick Neubauer, Alexander Bergmayr, Tanja Mayerhofer, Javier Troya, and Manuel Wimmer. Xmltext: from xml schema to xtext. In *SLE*, 2015.

[14] Falco Nogatz and Thom Frühwirth. From xml schema to json schema-comparison and translation with constraint handling rules. 2013.

[15] 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.

[16] Eric Prud'hommeaux and Jose Emilio Labra Gayo. RDF ventures to boldly meet your most pedestrian needs. *Bulletin of the Association for Information Science and Technology*, 41(4):18–22, 2015.

[17] 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.

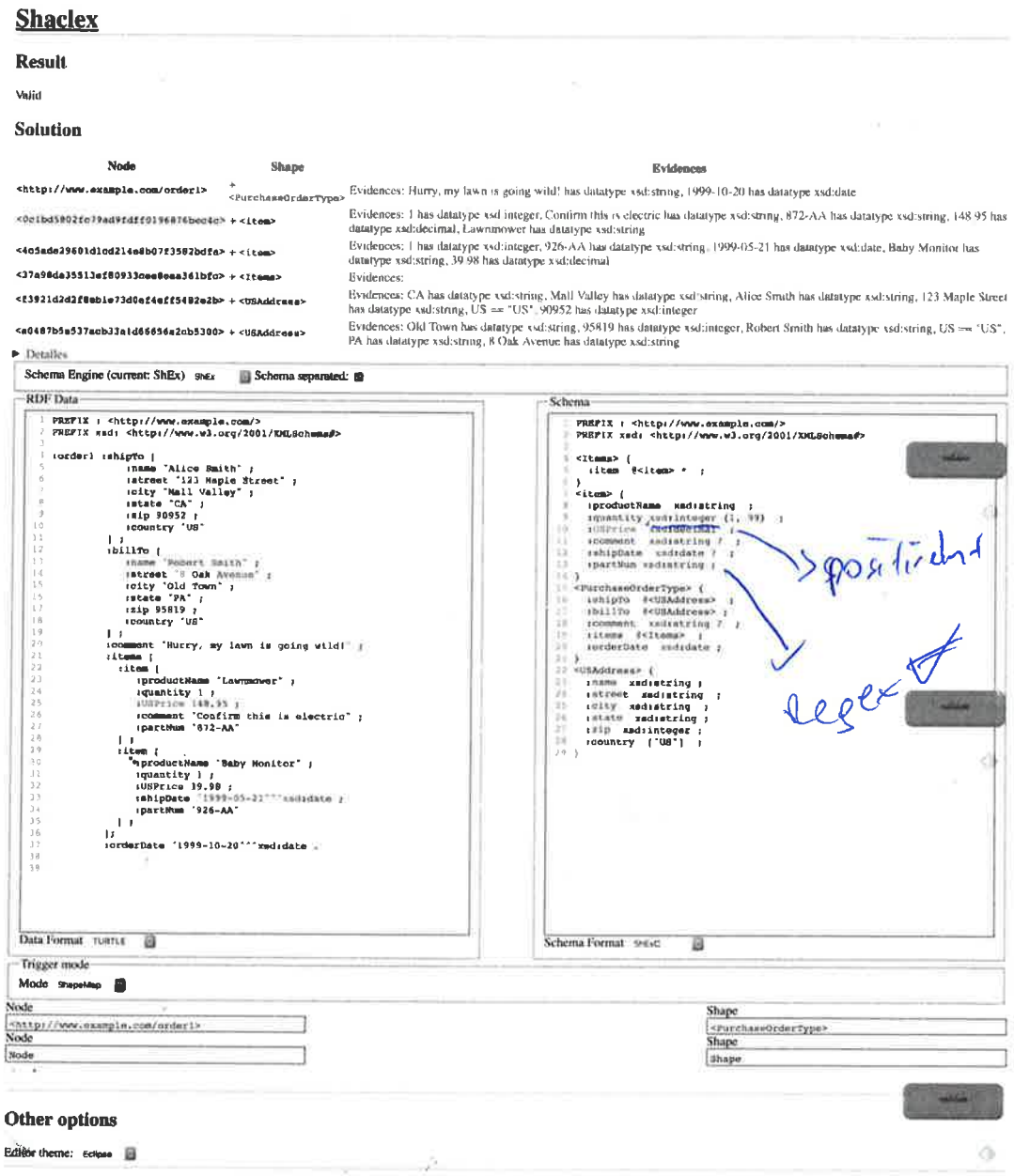

Fig. 1. Validation result using ShaclEx validator