DataSpecer Journal

Štěpán Stenchlák, Marek Čermák, Jan Janda, Tereza Miklóšová, Petr Škoda, Martin Nečaský and Jakub Klímek *

Department of Software Engineering, Charles University, Faculty of Mathematics and Physics, Malostranské náměstí 25, 118 00 Praha 1, Czechia

E-mails: stepan.stenchlak@matfyz.cuni.cz, ..., ..., petr.skoda@matfyz.cuni.cz, martin.necasky@matfyz.cuni.cz, jakub.klimek@matfyz.cuni.cz

Abstract. Abstract text.

Keywords: Keyword one, keyword two

1. Introduction

Martin Nečaský/Jakub Klímek

2. Model-driven approach to generation of technical artifacts

Martin Nečaský/Jakub Klímek

3. Ontology, structural model, data in rdf

Štěpán Stenchlák/Petr Škoda

4. UI: Specification manager

Štěpán Stenchlák

5. Documentation in Respec

Štěpán Stenchlák

*Corresponding author. E-mail: jakub.klimek@matfyz.cuni.cz.

6. JSON, JSON Schema, JSON-LD context

Petr Škoda

7. XML, XSD, XSLT

Marek Čermák

8. CSV, CSVW

Jan Janda

9. SHACL, ShEx

In previous chapters we have seen the Dataspecer tool architecture, how to create data structures in user-friendly environment, how to represent said data structure in different representations or how to convert data with this structure from various data formats to RDF format. For various formats we have standard validation artifacts, but there was none for validating RDF data before these two artifact generators - SHACL and ShEx - were implemented. Both SHACL and ShEx can be used to validate RDF data. SHACL [1] is a standard published by W3C. ShEx [] is not a standard, but is also being used by the community, e.g. in the Wikidata.

This chapter focuses on validating data against the data structure created by the user in Dataspecer. The first part of the chapter describes how mapping of structural model features to SHACL was made, following with a similar section focusing on translating of structural model features to ShEx. Both sections will present an overview of SHACL and ShEx respectively and how they work. Then they will describe the implementations in greater detail.

9.1. SHACL

Shapes Constraint Language (SHACL) [1] is a language for validating RDF graphs given some conditions. A validator that is compatible with SHACL takes two inputs: a Shapes Graph and a Data Graph. Shape Graph is an RDF graph that sets a set of constraints that are required of the validated data. Data Graph is an RDF graph containing data that needs to be validated by the Shapes Graph. After the validator is finished with the data validation, it produces Validation Report. This report describes in greater depth whether the Data Graph satisfies the conditions given by the Shapes Graph.

As of now, SHACL is a W3C Recommendation that serves to help augment the quality of data by setting structural and value constraints to an RDF graph. The conditions are set by using classes and attributes from not only the SHACL namespace http://www.w3.org/ns/shacl#.

SHACL can be also used for describing data that satisfy conditions set by the Shapes Graph and can be used in more ways than pure validation purposes. The data description characteristics is beyond the use in this project and will not be further discussed.

SHACL describes the desired data structure with the help of two main building blocks - sh:NodeShape and sh:PropertyShape. These two classes form a base that contains more attributes and characteristics demanded of the validated Data Graph. Node Shape describes a data node without the information how to get to that node. It specifies whether the node is IRI or blank node, whether it is strictly typed, which other properties it has and how to get to them. Property Shape on the other hand always contain a path that describes how to get to a certain property in the RDF graph and sets constraints on the given property. Property Shape is usually referenced as a sh:property from a Node Shape.

9.1.1. Mapping of structural model features to SHACL shapes

In this section we are going to see how each data-validating feature of Dataspecer has been integrated into the generation of the SHACL shape. Each feature subsection will provide information about the mapping and then an example for the reader to grasp the precise impact on the generated shape.

For the purpose of the increased readability of the examples presented, the TURTLE outputs have given prefixes to show relative IRIs to:

```
@base <https://myexample.com/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

The prefixes will not be shown again at each snippet presenting an example of the output, but apply to each and every one of them.

9.1.2. Common properties for both classes and attributes

There are some properties that are available for both classes and attributes in Dataspecer. All of them are there for semantic purposes to make the data structure more readable to the people that will be working with it in the future. That means that their occurrence in the generator's methods is purely for better readability and does not have an impact on the overall ability to validate the data graph with the generated SHACL shape.

Translations To open the workplace for other contributors to the creation and editing of the data structures, Dataspecer allows labels and descriptions of the data structure parts to be named and described in different languages. During this process the user specifies the IETF language tag[2] for the target language and then provides translation for either the label or the description of the Dataspecer data structure part. These language tagged strings are recognized as a core part of the RDF data types and are therefore supported in SHACL as well.

In the SHACL shape generator the translations are considered when generating semantic, non-validating properties, namely sh:name. Examples of the use of language-tagged strings are provided in the label 9.1.2 section down below.

Label Labels name the data structure entity. They can be edited and new translations can be created.

Example: The user edited the translations of the label to contain czech and english versions of the data structure part name. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<61ade27ffd9f512a2aaffd35576d769epoštovní_směrovací_čísloShape> a
 ↪  sh:PropertyShape;
    sh:name "poštovní směrovací číslo"@cs, "zip code"@en;
# The rest of the PropertyShape and TURTLE document
```

Description Descriptions carry semantically described details of the data structure entity. As labels they can be edited and new translations can be created.

Example: The user chose the address to be the root class of the data structure. The class has already defined description in czech language. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<5dee1c907cbf6a57a3b04e3bbafbf51cadresaShape> a sh:NodeShape;
    sh:description "Adresou se rozumí kombinace názvu okresu, názvu obce
     ↪  nebo vojenského újezdu, názvu části obce nebo v případě hlavního
     ↪  města Prahy názvu katastrálního území a názvu městského
     ↪  obvodu..."@cs;
# The rest of the NodeShape and TURTLE document
```

Technical label Technical label is also editable by user. It is generally used for naming structures that cannot be defined with regular language that contains white spaces that occur in labels and descriptions. In SHACL generator technical label is used along MD5 to generate the IRI for the SHACL shape, both **sh:NodeShape** and **sh:PropertyShape**.

Example: The user chose the address to be the root class of the data structure. The automatically provided technical label is adresa. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<f9d79044d74c490f3a023cbc5c243a33adresaShape> a sh:NodeShape;
    sh:nodeKind sh:BlankNodeOrIRI;
# The rest of the NodeShape and TURTLE document
```

The example contains IRI https://myexample.com/f9d79044d74c490f3a023cbc5c243a33adresaShape where https://myexample.com/ is set as the base of the IRI, f9d79044d74c490f3a023cbc5c243a33 is the generated MD5 hash and adresaShape is concatenation of the technical label and the word "Shape" to signal, that the IRI is an IRI of a shape.

9.1.3. Class properties

Some attributes are only available when working with classes of the data structure. In this subsection we will go through the SHACL generator mapping of these attributes.

Class type Class type is a unique attribute in a sense that it is already filled in by Dataspecer and cannot be editted by the user. It is therefore hidden from the Dataspecer class editing UI. Nevertheless the IRI of the class type is in the Dataspecer data structure data and is used to create the validating SHACL shape.

Class type can be used in the SHACL shape in two ways: either it server as a class type constraint or it helps with targetting the whole shape for correct validation process.

The former case is mapped to constraint on an instance type by SHACL predicate **sh:class** which ensures that instances of this class in data graph are of this type. This class conformance is only checked if the user has checked "ALWAYS" at the instances typing option in class editing UI or generally in the artifact configuration settings.

The latter case is used for SHACL targetting mechanism that is discussed in detail further in the chapter in subsection 9.1.6.

Example: The user chose "ALWAYS" for the instance typing for the address class. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<e4f13be068d7454a3ad17a1846005859adresaShape> a sh:NodeShape;
    sh:class
     ↪  <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/adresa>;
# The rest of the NodeShape and TURTLE document
```

Instances IRI regular expression Apart from aforementioned class properties, the only one that concerns generating SHACL shape is regular expression for IRIs of its instances. When the user sets this regular expression, SHACL validator will verify, whether the nodes in the data graph representing this class have the IRI in the desired form. This IRI matching is ensured by SHACL predicate **sh:pattern**.

Example: The user has filled in the regular expression for IRIs of the class instances to be in the domain of example.com. The generated SHACL output is as follows:

```
# Start of the TURTLE document
@base <https://myexample.com/>.

<f25745939ada5172f537c3177edd827eosobaShape> a sh:NodeShape;
    sh:pattern "^https?:\\/\\/(www\\.)?example\\.com(?:\\/.*)?$";
# The rest of the NodeShape and TURTLE document
```

9.1.4. Attribute properties

The attributes can be divided into two categories - primitive types attributes and associations. As associations are a combination of attribute and class properties using both definitions to define an association, this section will focus only on the pure attribute properties part of the definitions. Associations mapping combining both class and attribute properties can be found further in the text 9.1.4.

Data type Dataspecer offers a choice of basic primitive data types that can define the data type of a class attribute from a data structure. SHACL shape maps this choice to the predicate **sh:datatype** followed by an IRI of the data type. The user can also input their own data type, which is also going to be processed by the SHACL generator.

Example: The user has chosen for the attribute to be of type xsd:integer. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<ed38f9fff3e90dc09202d12b08a4b4famá_číslo_domovníShape> a sh:PropertyShape;
    sh:datatype <http://www.w3.org/2001/XMLSchema#integer>.
# The rest of the PropertyShape and TURTLE document
```

Regular expressions for selected data types For selected data types (namely xsd:string and xsd:anyURI) there is an option for the user to specify a pattern for the given input, that the data specification is going to be expecting. The pattern is in the form of regular expression and is applied in SHACL by the predicate sh:pattern. The object following the predicate representing the regular expression is in double quotes. The SHACL validator can then verify, whether the given data type adheres to the regular expression pattern. The predicate checking the pattern of data type is always present only in sh:PropertyShape, as it concerns an attribute specification.

Example: The user has chosen for the data type to be of kind xsd:string. The expected input for this attribute is a string consisting of at least one ASCII letter beginning with a capital letter. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<e3aaee5c5366ca72328b0b9964a522afsurnameShape> a sh:PropertyShape;
    sh:datatype xsd:string;
    sh:pattern "^[A-Z][a-z]*$".
# The rest of the PropertyShape and TURTLE document
```

Cardinality Cardinality represents the number of values the given attribute can have. Attributes can be mandatory, optional, some may have a range of acceptable number of values and some can have only a precise number of values. To verify whether the attribute has expected number of values, SHACL has two predicates: sh:minCount and sh:maxCount. They take an integer as their object and create inclusive range of possible number of values.

SHACL implicitly uses minimal cardinality of 0, so that even though the generated SHACL artifact does not explicitly state it, SHACL validators expect sh:minCount 0. When the user wants an attribute to be optional, meaning its minimal cardinality is zero, there is no matching predicate generated for it in SHACL generator.

Example: The user sets an attribute to be optional, but also allows at maximum 2 values to be present in the data. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<e3aaee5c5366ca72328b0b9964a522afsurnameShape> a sh:PropertyShape;
    sh:maxCount 2;
    sh:datatype <http://www.w3.org/2001/XMLSchema#anyURI>;
# The rest of the PropertyShape and TURTLE document
```

The implicit sh:maxCOunt by SHACL is infinite, meaning that if user does not set an uppper bound to the number of allowed values, SHACL validators will allow any number of values bigger than sh:minCount.

Example: The user sets an attribute value to be present at least three times, but does not set the upper bound. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<e3aaee5c5366ca72328b0b9964a522afscoreShape> a sh:PropertyShape;
    sh:minCount 3;
    sh:datatype <http://www.w3.org/2001/XMLSchema#integer>;
# The rest of the PropertyShape and TURTLE document
```

When the user needs other values than zero for sh:minCOunt and infinity for sh:maxCount, the SHACL generator will print both bounds to the resulting generated artifact.

Example: The user sets an attribute value to be present at least once and at maximum twice in the data. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<e3aaee5c5366ca72328b0b9964a522afscoreShape> a sh:PropertyShape;
    sh:minCount 1;
    sh:maxCount 2;
    sh:datatype <http://www.w3.org/2001/XMLSchema#string>;
# The rest of the PropertyShape and TURTLE document
```

9.1.5. Associations properties

Associations provide only one extra property that has not been covered by either class or attribute properties and that is the fact, that we are connecting a class to another class as its attribute. The rest of the definition for class or attribute properties stays the same and the reader is advised to review two previous sections if interested in these general ideas.

Attaching association to a class as an attribute When the data specification contains a class with an attribute in the form of another class, we call that an association. The association is usually another non-trivial class, that contains its own attributes, but is also an attribute itself.

The generator recognizes two cases: an otherwise empty association and a non-empty association. Both cases are attached to the parent class by **sh:property**, that points to appropriate sh:PropertyShape describing the attribute predicate binding it to the subject which is always sh:NodeShape for classes. The object of the association relation is a new sh:NodeShape if the associated class is not empty (does contain at least one attribute).

The empty association does not contain any attributes of its own. It is expected that this association will be represented in the data as an IRI and nothing else is required from the data of this association definition. This means that although the association points to a class, the NodeShape representing it will not be present in the generated SHACL artifact. This class will only be represented by a property shape which enumerates all of its important configurations such as cardinality, predicate path and node kind (IRI in this case).

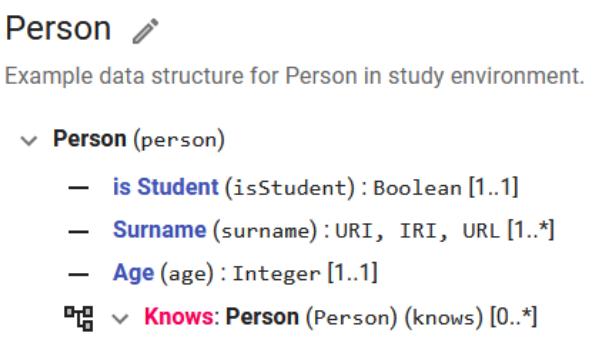

Fig. 1. Example of a data structure of a person who among other attributes has an attribute "knows" which is an association to other instances of class Person.

Example: The user created a data specification of a person, who may know other people. The association "knows" does not describe any attributes for this known Person. There are no set precise cardinalities for this data structure. Implicit cardinality values are used. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<762e033a25817341942243a1ffb6180apersonShape> sh:property
 ↪  <7e05bf39bc8be8b065244493b79a3d26knowsShape>.
<7e05bf39bc8be8b065244493b79a3d26knowsShape> a sh:PropertyShape;
    sh:path <knows>;
    sh:nodeKind sh:IRI.
# The rest of the PropertyShape and TURTLE document
```

In case the association specifies object class that contains attributes connected to it, the sh:PropertyShape also points to the class constraints written in sh:NodeShape for the associated class. That kind of pointing from a predicate to an object class is done with SHACL predicate **sh:node**.

Example: The user created a data specification of a person, who may know other people. The association "knows" specifies a Person as its class which also contains an attribute "name". There are no set precise cardinalities for the assocation. Implicit cardinality values are used. The generated SHACL output is as follows:

```
# Start of the TURTLE document containing prefix declarations
<a9c4ce4c480b4b1fec4facf305811af0personShape> a sh:NodeShape;
    sh:targetClass <Person>;
    sh:nodeKind sh:IRI;
<ec787f21ab18763447b1da9f3f7e3fe0knowsShape> a sh:PropertyShape;
    sh:name "Knows"@en;
    sh:path <knows>;
    sh:nodeKind sh:IRI.
<dd32e61da8feba969b75f57f0f7ffc72anotherPersonShape> a sh:NodeShape;
    sh:class
     ↪  <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/území-městského-obvod
    sh:nodeKind sh:IRI.
<dc2dadc88a4129901c5b9e9eca5d6c9enameShape> a sh:PropertyShape;
    sh:description "First name of a person"@en;
    sh:name "First name"@en;
    sh:path <name>;
    sh:datatype xsd:string.
<dd32e61da8feba969b75f57f0f7ffc72anotherPersonShape> sh:property
 ↪  <dc2dadc88a4129901c5b9e9eca5d6c9enameShape>.
<ec787f21ab18763447b1da9f3f7e3fe0knowsShape> sh:node
 ↪  <dd32e61da8feba969b75f57f0f7ffc72anotherPersonShape>.
<a9c4ce4c480b4b1fec4facf305811af0personShape> sh:property
 ↪  <ec787f21ab18763447b1da9f3f7e3fe0knowsShape>.
# The end of the TURTLE document
```

9.1.6. *Configure artifacts for data specification - Base URL and Instance identification and typing*

In the package selection of data structures to be edited, there is a Generation of artifacts section lower on the page layout where the user can either Configue artifacts or Generate the artifacts to a .zip file. In the Configure artifacts pop-up window there are four inputs that can be made that interest the validation generators.

Right on top is optional Base URL input window where the user can specify the base URL to which all of the artifacts create relative addresses and names to. Following the base URL section are three inputs concerning the default behavior for instances of classes corresponding to created data structures. The first option dictates whether the instances of classes have to be identified by IRI or whether blank node is also feasible. The second input sets whether instances of classes have to be typed by the class identifier or not. The last input option tells in general whether classes allow more properties than stated in the data structure.

All three inputs under Instance identification and typing header can be also set individually for each class while creating and editing the data structure.

Base URL The base URL that user fills in the artifact configuration window is translated into the TURTLE format that SHACL uses for output.

Example: Base URL is set to https://myexample.com/. The SHACL generator produces this line in the TURTLE output prefix part:

```
# Other prefixes
@base <https://myexample.com/> .
# The rest of the TURTLE document
```

This base URL is then displayed as follows in the TURTLE body:

```
# Start of the TURTLE document
@prefix sh: <http://www.w3.org/ns/shacl#>.
@base <https://myexample.com/>.
```

```
<d856f855ee8af546dbfcafbe9b1fde32adresaShape> a sh:NodeShape;
# The rest of the TURTLE document
```

When there is no base URL provided by the user in the artifact configuration, the IRIs in the generator output are relative to no base URL. There is no @base in the prefixes part in the TURTLE format and the IRIs are encapsulated in < and > respectively from the left and the right side of the IRI in the TURTLE body.

Example: Base URL is not set. The SHACL generator produces no line in the TURTLE output prefix part. The body with relative IRIs is as follows:

```
# Start of the TURTLE document
@prefix sh: <http://www.w3.org/ns/shacl#>.

<d856f855ee8af546dbfcafbe9b1fde32adresaShape> a sh:NodeShape;
# The rest of the TURTLE document
```

Instance identification The user can choose whether the instances of the data specification are always going to have the IRIs identifying the nodes in the RDF format, if the IRIs are optional or they are banned altogether. This choice has an impact on which **sh:nodeKind** the SHACL shape is going to expect from the respective class instance.

The sh:nodeKind restriction has 6 possible inputs out of which only 3 are used to differentiate between the 3 possible user inputs for identifying the instances of the classes. The inputs used for the Dataspecer needs are sh:IRI, sh:BlankNodeOrIRI and sh:BlankNode, each of them corresponding to the possible user inputs mentioned earlier.

Example: The user has chosen the identification of the class instances to be mandatory. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<bfe55dedc512e9e184b5194b632c1c03adresaShape> a sh:NodeShape;
    sh:nodeKind sh:IRI;
# The rest of the NodeShape and TURTLE document
```

Example: The user has chosen the identification of the class instances to be optional. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<bfe55dedc512e9e184b5194b632c1c03adresaShape> a sh:NodeShape;
    sh:nodeKind sh:BlankNodeOrIRI;
# The rest of the NodeShape and TURTLE document
```

Example: The user has chosen the identification of the class instances to not be used. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<bfe55dedc512e9e184b5194b632c1c03adresaShape> a sh:NodeShape;
    sh:nodeKind sh:BlankNode;
# The rest of the NodeShape and TURTLE document
```

Instance typing The Dataspecer tool allows users to specify whether the class in the data structure will demand its instances to be typed. In other words, whether the data node in RDF graph will require the instances of this class to have predicate http://www.w3.org/1999/02/22-rdf-syntax-ns#type and a corresponding class type as an object of this triple.

Along with the attributes of the root of the data structure, this choice creates conditions for SHACL targetting, that is necessary for the generated shapes to work properly when validating data. How the setting of this artifact configuration affects the final shape is described in section 9.1.6.

As in instance identification, there are three possible user inputs. Instance typing mandatory, optional and banned.

When the instance typing is mandatory, the shape demands the instance to have given class type with the help of SHACL predicate **sh:class**. This construct ensures, that the SHACL validator checks, whether the target data node contains predicate rdf:type (abbreviated as a) and a corresponding class IRI as its object. *Example: The user has chosen the typing of the class instances to be mandatory. The generated SHACL output is as follows:*

```
# Start of the TURTLE document
@prefix slov: <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/>.

<47650012853ed01edab3415a461b176cprávní_formaShape> a sh:NodeShape;
    sh:class slov:právní-forma;
# The rest of the NodeShape and TURTLE document
```

If the instance typing is optional, the SHACL generator does not generate any specific triple for the shape as the class may or may not exist in the data graph and both possibilities are correct.

If the instance typing of the class is banned by the user, the rdf:type cannot appear in the data graph. That condition in SHACL is represented by negation of a property, that has path of the predicate rdf:type and is in the data node at least once. That condition means not allowing rdf:type to appear even once in the data graph relevant node. *Example: The user has chosen the typing of the class instances to not be used. The generated SHACL output is as follows:*

```
# Start of the TURTLE document
<bfe55dedc512e9e184b5194b632c1c03adresaShape> a sh:NodeShape;
    sh:not [
        a sh:PropertyShape;
        sh:path <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>;
        sh:minCount 1
    ];
# The rest of the NodeShape and TURTLE document
```

Additional class properties The last configuration property for artifacts that concerns SHACL is whether the data structure class allows properties other than the explicitly specified ones. The possible inputs are that additional properties are allowed or are not allowed. SHACL implicitly allows other properties, so there is no action done for this case. The case where additional properties are not allowed is handled by **sh:closed**. This predicate ultimately also bans the usage of the rdf:type property. Since we do not explicitly create this property separately in Dataspecer, it is assumed, that the user normally wants his instances typed. If the user wishes to not have the instances of the data structure typed, there is another choice box to cater to that need. Hence the rdf:type is protected from the sh:closed predicate in the SHACL shape.

Example: The user has chosen that other properties of the class Osoba are not allowed. The generated SHACL output is as follows:

```
# Start of the TURTLE document
<f25745939ada5172f537c3177edd827eosobaShape> a sh:NodeShape;
    sh:closed true;
    sh:ignoredProperties
     ↪  (<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>);
# The rest of the NodeShape and TURTLE document
```

9.1.7. SHACL targets

For the SHACL validator to work and verify data against the generated SHACL artifact, there has to be a directive telling the validator which data to verify against which SHACL shapes. That is the goal of SHACL targets.

In Dataspecer the goal was to define as general SHACL targets as possible, knowing that the user might feed the SHACL validator data with any kind of node names and that the data specification in Dataspecer can be very general.

This part describes the main cases of targetting where the common tools of SHACL language can be used directly.

1. **The root of the structure always has unique type** If the rdf:type of the structure is unique, it can be targetted by sh:targetClass. This can only be used when "Explicitní určení typu instancí" is set to vyžadováno, because if the type determination is missing in the data, the data won't be checked against created shape. Each value of sh:targetClass in a shape is an IRI.
2. **The root of the structure contains a unique predicate leading from it** If the root has a unique predicate in attributes with cardinality [1..x], it can be used for targetting by sh:targetSubjectsOf. If the predicate is [0..x], there is a chance the predicate is not available in data, not allowing the shape to find a focusNode. The values of sh:targetSubjectsOf in a shape are IRIs.
3. **The root has an attribute, which has always unique type** If the rdf:type of the structure is unique, it can be targetted by sh:targetClass. This can only be used when "Explicitní určení typu instancí" is set to vyžadováno for the targetted attribute class, because if the type determination is missing in the data, the data won't be checked against created shape. The only difference to 1) is, that the structure above the targetted class has to be found using reverse paths (using sh:path: [sh:inversePath ex:parent]). Each value of sh:targetClass in a shape is an IRI.
4. **The root has an attribute that contains a unique predicate leading from it** If the targetted attribute has a unique predicate in attributes with cardinality [1..x], it can be used for targetting by sh:targetSubjectsOf. The only difference to 2) is, that the structure above the targetted node has to be found using reverse paths (using sh:path: [sh:inversePath ex:parent]). The values of sh:targetSubjectsOf in a shape are IRIs.

Below follows the discussion of cases where usual SHACL defined tools do not work.

1. **There are no unique types** In the whole structure, there are no unique types of any class. In that case, the rdf:type will be the same for different nodes in different levels of the structure and both will be targetted with sh:targetClass. But the aim is to check the structure from only one of them and the other focusNodes will give false Negative results while being checked. In that case let's check for second option, checking for unique predicates.
2. **There are also no unique predicates** In the whole structure there is no unique predicate with the cardinality [1..x] that would ensure it always occures in the data.
3. **There are unique unique types but the explicit typing of instances is not mandatory** If the unique class has Explicitní určení typu instancí is set to other than vyžadováno, there is no guarantee the data will have the rdf:type specified. In that case there is no certainly focusable node in the data.
4. **There are unique predicates but are OPTIONAL ... [0..x]** In case there are unique predicates but they are only optional with cardinality [0..x], there is no guarantee they will be present in the to-be-validated data.

If all outliers 1)-4) are true in the structure, there are no basic SHACL means how to target the focus node from where to validate the data.

If all techniques in the first section of this discussion fail, the user is informed about this issue in an informative manner and asked them to SET the conditions in whichever manner they feel the most comfortable having their data structure usage in mind. Then the user may fix one of the problems at hand by:

- giving one class a unique type
- changing one predicate for a unique predicate not in the data yet
- setting a unique class type instance typing mandatory
- setting a unique predicate cardinality to [1..x]

After the analysis has been done, the implementation for SHACL targets in Dataspecer is as follows with the notion that the cases are considered in this order. If the condition for the first use case does not hold, the generator tries to use the second use case etc.

1. **The root class of the data specification has a unique type and is set to instance typing "ALWAYS".** In this case, the data node is being targetted with the SHACL predicate **sh:targetClass**, because it is unique so there is not going to be any unexpected matching for verification, that the user would not want.

2. **The root class of the data specification has a predicate with a unique type of cardinality [1..x].** Since the cardinality assures that the predicate is always present and its type is unique, we can use SHACL predicate **sh:targetSubjectsOf** to point at the root of the data specification by its child.
3. **The root class of the data specification has an association with cardinality [1..x] which has a unique type and is set to instance typing "ALWAYS".** In this case, the data node is being targetted with the SHACL predicate **sh:targetClass** same as in the first use case but with the difference that the node being targetted by SHACL is a child of the root class of the data specification. The root of the data specification is going to be validated by an inverse path reference from this child node. The inverse path predicate for SHACL is done with **sh:path [sh:inversePath <predicateComingFromParent>]**.
4. **The root class of the data specification has an association of cardinality [1..x] which has an association with a unique type and cardinality of [1..x].** The mandatory presence of the predicates assures that the validator will always have a predicate to start at if it is to succeed. The SHACL language equivalents used for this condition are **sh:targetSubjectsOf** from use case 2 and the same technique of validating inverse path predicates leading to the root of the data structure from the use case 3: **sh:path [sh:inversePath <predicateComingFromParent>]**.
5. **None of the conditions above hold.** In this not very common case the user is informed that the SHACL generator cannot generate SHACL artifact for this kind of data structure and the thrown error suggests possible edits that can be done to the structure to make it targettable.

Following examples depict different approaches to SHACL targets in generated artifacts.

1. **The root class has a unique type**

Fig. 2. Example of a data structure of an address that fulfills the aforementioned conditions for invoking use case 1 for generation of the Query Shape Map

Example: The root class of address has a unique rdf:type that is not found anywhere else in the structure and explicit instance typing for the root class is required. Its describing shape is the address shape. The part of generated SHACL artifact is as follows:

```
<9d574a531084f922de1afced945f4d3eaddressShape> a sh:NodeShape;
    sh:targetClass
     ↪  <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/adresa>;
    sh:class
     ↪  <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/adresa>;
    sh:nodeKind sh:IRI;
    sh:description "Adresou se rozumí kombinace ..."@cs, "Address of a
     ↪  place"@en;
    sh:name "adresa"@cs, "Adresa"@en.
```

2. **The root class has a unique predicate**

Example: The root class of address has a unique attribute that is not found anywhere else in the structure and its cardinality is [1..x]. Its describing shape is the address shape. The part of generated SHACL artifact is as follows:

Fig. 3. Example of a data structure of an address that fulfills the aforementioned conditions for invoking use case 2 for generation of the Query Shape Map

```
<9d574a531084f922de1afced945f4d3eaddressShape> a sh:NodeShape;
    sh:targetSubjectsOf
     ↪  <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/je-adresou-stavebn
    sh:class
     ↪  <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/adresa>;
    sh:nodeKind sh:IRI;
    sh:description "Adresou se rozumí kombinace ..."@cs, "Address of a
     ↪  place"@en;
    sh:name "adresa"@cs, "Address"@en.
```

3. **The root class has an association that has a unique class type**

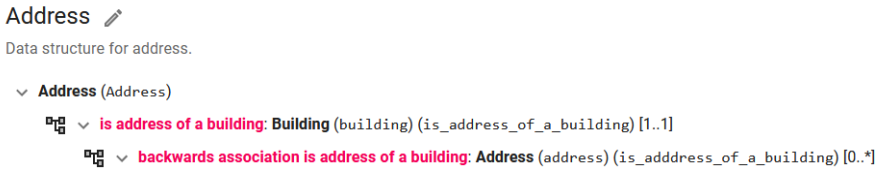

Fig. 4. Example of a data structure of an address that fulfills the aforementioned conditions for invoking use case 3 for generation of the Query Shape Map

Example: The root class of tourist destination has an association for a building with unique rdf:type that is not found anywhere else in the structure and explicit instance typing for the building class is required. The association has a cardinality of [1..x]. Its describing shape is the building shape. The part of generated SHACL artifact is as follows:

```
<81ec1dd3ebfb7f3f187a40f63ef50cbebuildingShape> a sh:NodeShape;
    sh:targetClass
     ↪  <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/stavební-objekt>;
    sh:class
     ↪  <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/stavební-objekt>;
    sh:nodeKind sh:IRI;
    sh:description "Stavebním objektem se rozumí budova..."@cs;
    sh:name "stavební objekt"@cs, "Building"@en.
```

4. **The root class has an association and there is a unique predicate going from it**

The root class of tourist destination has an association for a building with not unique rdf:type. But it has an attribute that is unique and has cardinality of [1..x]. Its describing shape is the building shape. The part of generated SHACL artifact is as follows:

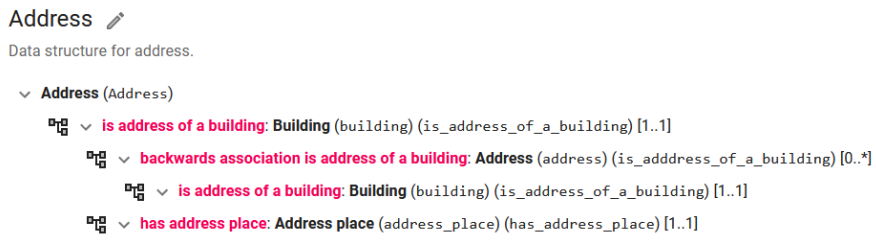

Fig. 5. Example of a data structure of an address that fulfills the aforementioned conditions for invoking use case 4 for generation of the Query Shape Map

```
<81ec1dd3ebfb7f3f187a40f63ef50cbebuildingShape> a sh:NodeShape;
    sh:targetSubjectsOf
     ↪  <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/má-adresní-místo>;
    sh:class
     ↪  <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/stavební-objekt>;
    sh:nodeKind sh:IRI;
    sh:description "Stavebním objektem se rozumí budova ..."@cs;
    sh:name "stavební objekt"@cs, "Building"@en.
```

9.1.8. *Conclusion*

This chapter described how SHACL artifact is generated in Dataspecer. To give one final complex and complete example, take a look at this data specification created in Dataspecer:

Fig. 6. Example of a data structure of a Tourist destination that has several attributes and associations defined.

This data specification produces SHACL artifact, that in its entirety is as follows:

```
@prefix sh: <http://www.w3.org/ns/shacl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@base <https://myexample.com/>.

<d7528e03c25e75bf0abc589ed5545ad5tourist_destinationShape> a sh:NodeShape;
    sh:targetClass
     ↪  <https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>;
```

```
    sh:class
     ↪  <https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>;
    sh:pattern "^https.+$";
    sh:closed true;
    sh:ignoredProperties
     ↪  (<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>);
    sh:nodeKind sh:IRI;
    sh:description "Samostatný turistický cíl."@cs, "A separate tourist
     ↪  destination"@en;
    sh:name "Tourist destination"@en, "Turistický cíl"@cs.
<ad078e41eecf74b8c1e6a29a453cb1f2capacityShape> a sh:PropertyShape;
    sh:name "kapacita"@cs, "capacity"@en;
    sh:path <https://slovník.gov.cz/datový/sportoviště/pojem/kapacita>.

<ad078e41eecf74b8c1e6a29a453cb1f2capacityShape> sh:datatype xsd:integer.
<d7528e03c25e75bf0abc589ed5545ad5tourist_destinationShape> sh:property
 ↪  <ad078e41eecf74b8c1e6a29a453cb1f2capacityShape>.
<1403f7adf349b6faf375941eb2c23517smoking_allowedShape> a sh:PropertyShape;
    sh:description "Determines whether it is possible to smoke tobacco
     ↪  products in the tourist destination."@en, "Určuje, zda je možné v
     ↪  turistickém cíli kouření tabákových výrobků."@cs;
    sh:name "smoking allowed"@en, "kouření povoleno"@cs;
    sh:maxCount 1;
    sh:path
     ↪  <https://slovník.gov.cz/datový/turistické-cíle/pojem/kouření-povoleno>.

<1403f7adf349b6faf375941eb2c23517smoking_allowedShape> sh:datatype
 ↪  xsd:boolean.
<d7528e03c25e75bf0abc589ed5545ad5tourist_destinationShape> sh:property
 ↪  <1403f7adf349b6faf375941eb2c23517smoking_allowedShape>.
<aa7de934a3ecd61f062989b47fe07237has_contactShape> a sh:PropertyShape;
    sh:name "kontakt"@cs, "has contact"@en;
    sh:minCount 1;
    sh:maxCount 1;
    sh:path <https://slovník.gov.cz/datový/turistické-cíle/pojem/kontakt>;
    sh:nodeKind sh:IRI;
    sh:pattern "^[A-Z]$".
<217547b71513ecc5b06d7da60879e65acontactShape> a sh:NodeShape;
    sh:class <https://slovník.gov.cz/generický/kontakty/pojem/kontakt>;
    sh:pattern "^[A-Z]$";
    sh:nodeKind sh:IRI;
    sh:description "Kontaktní údaje, např. na člověka, společnost,
     ↪  apod."@cs;
    sh:name "Contact"@en, "Kontakt"@cs.
<5cea7c188264a85165deeb269967e2c6has_urlShape> a sh:PropertyShape;
    sh:description "Webová kontaktní adresa: webová stránka či WebID."@cs;
    sh:name "má URL"@cs, "URL"@en;
    sh:minCount 2;
    sh:path <https://slovník.gov.cz/generický/kontakty/pojem/má-url>.

<5cea7c188264a85165deeb269967e2c6has_urlShape> sh:datatype xsd:anyURI.
```

```
<217547b71513ecc5b06d7da60879e65acontactShape> sh:property
 ↪  <5cea7c188264a85165deeb269967e2c6has_urlShape>.
<b74bd55ee12d3a2f7a5c37dc0f4eaad4facebook_accountShape> a sh:PropertyShape;
    sh:description "Uživatelské jméno na Facebooku."@cs;
    sh:name "má účet na Facebooku"@cs, "facebook"@en;
    sh:maxCount 1;
    sh:path
     ↪  <https://slovník.gov.cz/generický/kontakty/pojem/má-účet-na-facebooku>.

<b74bd55ee12d3a2f7a5c37dc0f4eaad4facebook_accountShape> sh:datatype
 ↪  xsd:string.
<217547b71513ecc5b06d7da60879e65acontactShape> sh:property
 ↪  <b74bd55ee12d3a2f7a5c37dc0f4eaad4facebook_accountShape>.
<d1b26941dcc54b5ed9cc7b2eb763a90ckind_of_contactShape> a sh:PropertyShape;
    sh:description "Druh kontaktu, například Oficiální, Neformální,
     ↪  apod."@cs;
    sh:name "má druh kontaktu"@cs, "kind of contact"@en;
    sh:maxCount 1;
    sh:path
     ↪  <https://slovník.gov.cz/generický/kontakty/pojem/má-druh-kontaktu>.

<d1b26941dcc54b5ed9cc7b2eb763a90ckind_of_contactShape> sh:datatype
 ↪  <http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>.
<217547b71513ecc5b06d7da60879e65acontactShape> sh:property
 ↪  <d1b26941dcc54b5ed9cc7b2eb763a90ckind_of_contactShape>.
<aa7de934a3ecd61f062989b47fe07237has_contactShape> sh:node
 ↪  <217547b71513ecc5b06d7da60879e65acontactShape>.
<d7528e03c25e75bf0abc589ed5545ad5tourist_destinationShape> sh:property
 ↪  <aa7de934a3ecd61f062989b47fe07237has_contactShape>.
<c84c57523a711a712805e1c039301eefhas_available_languageShape> a
 ↪  sh:PropertyShape;
    sh:description "Dostupný jazyk v místě turistického objektu."@cs,
     ↪  "Language available at the location of the tourist facility."@en;
    sh:name "má dostupný jazyk"@cs, "has available language"@en;
    sh:minCount 1;
    sh:path
     ↪  <https://slovník.gov.cz/datový/turistické-cíle/pojem/má-dostupný-jazyk>;
    sh:nodeKind sh:IRI;
    sh:pattern "^[A-Z]$".
<d7528e03c25e75bf0abc589ed5545ad5tourist_destinationShape> sh:property
 ↪  <c84c57523a711a712805e1c039301eefhas_available_languageShape>.
```

The artifact describes every aspect of the data specification, that is relevant for the data validation process based on this data specification. It is ready to be used in any SHACL compatible validator as the Shapes Graph part of the input, the user only needs to provide Data Graph that they wish to validate.

9.2. ShEx

Shape Expressions (ShEx) [3] is a language for describing RDF graph structures. ShEx schema contains expressions constraining the structure in terms of relations, their directions, data types, node types, cardinalities and patterns for data. This description of the data structure can be then used for validating RDF data, generating user interfaces or transforming RDF data into other formats.

In this work, the focus of use for ShEx is on validating RDF data given the data specification from Dataspecer.

Validating RDF data with ShEx validator requires three inputs: ShEx schema, ShEx Map and RDF data. ShEx schema describes constraints that have to be fulfilled in order to claim that the RDF data is conformant. ShEx Map states which data nodes are to be validated against which ShEx shape from the ShEx schema. RDF data represent the data that need to be validated. After processing the inputs, the validator yields result shape map, which describes which data nodes succeeded or failed in the validation process.

ShEx generator for Dataspecer consists of two ShEx artifacts: ShEx and ShEx Query Map. ShEx in this context represents generated ShEx schema for validating given Dataspecer data specification and ShEx Query Map is a generated shape query map that generalizes the focusing nodes with the help of SPARQL structures. ShEx describes the constraints for the data specification whereas the ShEx Query Map only tells which data nodes are to be validated.

ShEx has several serialization formats: ShExC, ShExJ and ShExR. ShExC is human readable compact syntax, ShExJ is an abstract JSON-LD syntax and ShExR is an RDF representation derived from JSON-LD syntax. In this work, ShExC format has been chosen for the generator artifacts creation. As the artifacts in their most hands-on situation are directly generated on the screen for the user to see, it was desirable to choose a syntax that is the most human-readable. ShExC is also widely used in the ShEx documentation and in known ShEx validators, so the format was found well suited for the implementation.

In the next sections, ShEx schema generator mapping to Dataspecer data specification structures is presented followed by ShEx Query Map decisions on how the data nodes are targetted for validation.

For the sake of grasping how the generated artifacts look like, we give a full generated artifact example. The data specification from Dataspecer for this generated example is as follows:

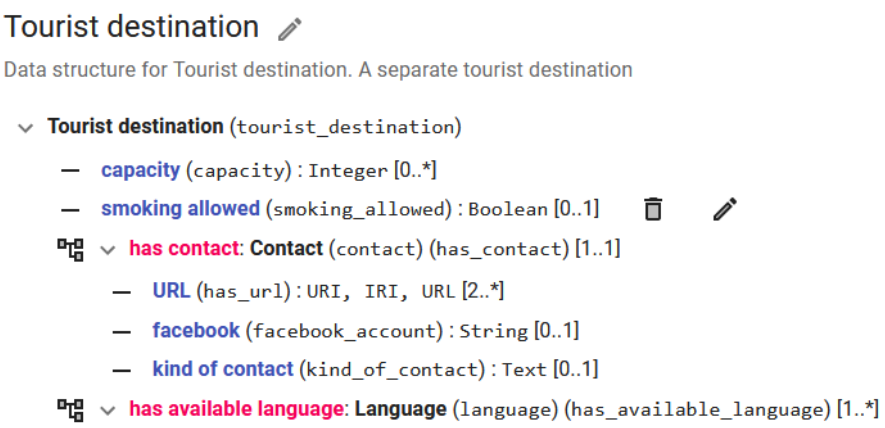

Fig. 7. Example of a data structure of a Tourist destination that has several attributes and associations defined.

The shown data specification generates following ShEx artifact:

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
base <https://myexample.com/>

<217547b71513ecc5b06d7da60879e65acontactShExShape> IRI /^[A-Z]$/{
        a [<https://slovník.gov.cz/generický/kontakty/pojem/kontakt>] ;
        <https://slovník.gov.cz/generický/kontakty/pojem/má-url> xsd:anyURI
        ↪  {2,}
                // rdfs:label        "má URL"
                // rdfs:comment        "Webová kontaktní adresa: webová
                ↪  stránka či WebID." ;
```

```
        <https://slovník.gov.cz/generický/kontakty/pojem/má-účet-na-facebooku>
        ↪ xsd:string ?
                // rdfs:label        "má účet na Facebooku"
                // rdfs:comment        "Uživatelské jméno na Facebooku." ;
        <https://slovník.gov.cz/generický/kontakty/pojem/má-druh-kontaktu>
        ↪ rdf:langString ?
                // rdfs:label        "má druh kontaktu"
                // rdfs:comment        "Druh kontaktu, například Oficiální,
                ↪ Neformální, apod."
}
<d7528e03c25e75bf0abc589ed5545ad5tourist_destinationShExShape> IRI
↪ /^https.+$/ CLOSED{
        a
        ↪ [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>]
        ↪ ;
        <https://slovník.gov.cz/datový/sportoviště/pojem/kapacita>
        ↪ xsd:integer *
                // rdfs:label        "kapacita" ;
        <https://slovník.gov.cz/datový/turistické-cíle/pojem/kouření-povoleno>
        ↪ xsd:boolean ?
                // rdfs:label        "smoking allowed"
                // rdfs:comment        "Determines whether it is possible to
                ↪ smoke tobacco products in the tourist destination." ;
        <https://slovník.gov.cz/datový/turistické-cíle/pojem/kontakt>
        ↪ @<217547b71513ecc5b06d7da60879e65acontactShExShape>
                // rdfs:label        "kontakt" ;
        <https://slovník.gov.cz/datový/turistické-cíle/pojem/má-dostupný-jazyk>
        ↪ IRI +
                // rdfs:label        "má dostupný jazyk"
                // rdfs:comment        "Dostupný jazyk v místě turistického
                ↪ objektu."
}
```

and the following ShEx Query Map artifact:

```
{ FOCUS rdf:type
↪ <https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>}
↪ @<https://myexample.com/d7528e03c25e75bf0abc589ed5545ad5tourist_destinationShExShape
```

The following sections will only show snippets of the generated schemas as to not copy parts of the schema that is not important for explaining of the mapping. Some snippets may be taken from the general example, but some may be generated from different data specifications to show different behaviors of the mapping system.

9.2.1. Mapping of structural model features to SHACL shapes

In this section we are going to see how each data-validating structures of Dataspecer has been integrated into the generation of the ShEx schema. Each data structure subsection will provide information about the mapping and then an example for the reader to grasp the precise impact on the generated part of the schema.

For the purpose of the increased readability of the examples presented, the ShEx outputs have given prefixes to show relative IRIs to:

```
base <https://myexample.com/>
prefix sh: <http://www.w3.org/ns/shacl#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
```

The prefixes will not be shown again at each snippet presenting an example of the output, but apply to each and every one of them.

9.2.2. Common properties for both classes and attributes

These properties are non-validating and are present for better readability and user experience. They help to discern between different data specifications and their particular data structures inside. The common properties for both classes and attributes are label, description and their respective translations. Both label and description fall in the category of ShEx annotations, which have special syntactic mark at the beginning of the line in the form of "//". It is notable that if more annotating lines follow each other, only the last one contains semicolon to close the annotating section. Also the line with constraining triple preceding the annotation does not end with semicolon.

Translations Dataspecer follows the RDF language-tagged strings feature for better international usability for labels and descriptions of the data structure parts to be named and described in different languages. The user specifies the IETF language tag[2] for the target language and then provides translation for either the label or the description of the Dataspecer data structure part. The support for the language-tagged strings in ShEx is limited in a way that neither labels or descriptions are part of ShEx syntax. ShEx does not define its ontology and borrows predicates for labels and descriptions from <http://www.w3.org/2000/01/rdf-schema#>. As the label and description are a part of a comment in the ShEx language they are not the main focus of the ShEx language.

In the ShEx schema generator the translations are not considered when generating semantic, non-validating properties, namely rdfs:label and rdfs:comment.

Label Label in ShEx can be represented with the use of annotation line beginning with //. ShEx does not have its own vocabulary to represent annotating triples. Instead rdfs:label can be used as the predicate which takes string as its object. To annotate the data structure, first language variant is used from the data store.

Example to this mapping and the mapping of description is found in the next paragraph about Description9.2.2.

Description Similar to label, description is an annotation that in ShExC format is done by prepending the line containing the descriptive predicate and object with //. For describing the ShEx constraint triple preceding the annotating section, rdfs:comment is used. The object after rdfs:comment predicate is a string.

Example: The data specification has class at its top level that has named and described attribute "smoking allowed". The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> IRI{
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
  <https://slovník.gov.cz/datový/turistické-cíle/pojem/kouření-povoleno>
   ↪  xsd:boolean ?
    // rdfs:label        "smoking allowed"
    // rdfs:comment        "Determines whether it is possible to smoke
     ↪  tobacco products in the tourist destination." ;
# The rest of the of the ShExC document
```

Technical label Technical label found in Dataspecer attributes is used for generating distinctive ShEx shape IRI. Along with md5 hashing and appending "ShExShape" at the end of the IRI it creates unique relative IRI. If base URL is set in the configuration, the shape IRI is relative to that base URL.

Example: The data specification has class at its top level with a technical label of ïuristický_cíl.. The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> IRI{
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
# The rest of the of the ShExC document
```

9.2.3. Class properties

Some attributes are only available when working with classes of the data structure. In this subsection we will go through the ShEx generator mapping of these attributes.

Class type Class type is described with the help of rdf:type predicate inside the ShEx shape. following the predicate is a value set containing only the IRI that represents the class type. That means the data node constrained by this shape has an rdf:type which has only one value possible: the one contained in the value set.

Example: The data specification has class with a class type https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl. The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> IRI{
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
# The rest of the of the ShExC document
```

Instances IRI regular expression User is able to set, whether they need the IRIs of the data nodes to adhere to some pattern when considering the IRI naming convention. This constraint on naming conventions for data nodes is done in ShEx by putting the regular expression between slashes after the shape IRI and after the node kind for the corresponding data node.

Example: The data specification has class with a class type https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl. The IRI of the data node should start with "https" followed by at least one character denoted by ĥttps.+$ regular expression in slashes. The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> IRI /^https.+$/{
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
  <https://slovník.gov.cz/datový/sportoviště/pojem/kapacita> xsd:integer *
          // rdfs:label        "kapacita" ;
# The rest of the of the ShExC document
```

9.2.4. Attribute properties

The attributes can be divided into two categories - primitive types attributes and associations. As associations are a combination of attribute and class properties using both definitions to define an association, this section will focus only on the pure attribute properties part of the definitions. Associations mapping combining both class and attribute properties can be found further in the text 9.2.4.

Data type Data type of object of given predicate is described in the ShExC after the predicate and space. If the data type of the object is not stated, the ShExC displays a dot in that place instead symbolizing that any type can follow given predicate.

Example: The data specification has a class that has an attribute https://slovník.gov.cz/datový/sportoviště/pojem/kapacita. The data type of object following this predicate is xsd:integer. The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> {
  <https://slovník.gov.cz/datový/sportoviště/pojem/kapacita> xsd:integer
        // rdfs:label        "kapacita" ;
# The rest of the of the ShExC document
```

Example: The data specification has a class that has an attribute https://slovník.gov.cz/generický/kontakty/pojem/má-účet-na-facebooku. The data type of the object following this predicate is undecided. The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<217547b71513ecc5b06d7da60879e65akontaktShExShape> {
  <https://slovník.gov.cz/generický/kontakty/pojem/má-účet-na-facebooku> .
}
# The rest of the of the ShExC document
```

Regular expressions for selected data types Dataspecer allows the users to set regular expressions to data types of xsd:anyURI and xsd:string. That means that the object values of those types have to adhere to that regular expression to be conformant. When the regular expression field is filled in, the ShEx generator inserts this regular expression pattern after the predicate and data type and before the cardinality mark.

Example: The data specification has a class that has an attribute https://slovník.gov.cz/generický/kontakty/pojem/má-druh-kontaktu. The data type of the object following this predicate is xsd:string. The regular expression expects the object of this predicate start with a capital letter followed by at least one any character. The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<217547b71513ecc5b06d7da60879e65akontaktShExShape> {
  a [<https://slovník.gov.cz/generický/kontakty/pojem/kontakt>] ;
  <https://slovník.gov.cz/generický/kontakty/pojem/má-druh-kontaktu>
   ↪  xsd:string /^[A-Z].+$/
        // rdfs:label        "má druh kontaktu"
        // rdfs:comment        "Druh kontaktu, například Oficiální,
         ↪  Neformální, apod."
# The rest of the of the ShExC document
```

Cardinality Cardinality or the number of data triples the data node can have with given predicate and its object is by default exactly one in ShEx. That means that if there is no cardinality mark after the data type of an object, ther has to be exactly one of this data relations in the data graph.

Example: The data specification for a contact has a class that has an attribute https://slovník.gov.cz/generický/kontakty/pojem/má-druh-kontaktu. The data type of the object following this predicate is xsd:string. This value should be present in the data graph exactly once. The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<217547b71513ecc5b06d7da60879e65akontaktShExShape> {
  <https://slovník.gov.cz/generický/kontakty/pojem/má-druh-kontaktu>
   ↪  xsd:string
        // rdfs:label        "má druh kontaktu" ;
# The rest of the of the ShExC document
```

Other cathegory of cardinality description is cardinalities denoted by regular expressions cardinalities. Namely they are special characters ?, * and +, exactly n times n and ranges in curly brackets from n to m occurances n,m. Question mark ? denotes either zero or one occurance, asterisk * denotes any number of times including zero, plus + denotes any number of times but at least once.

The curly brackets from regular expression can show that the value should be present exaclty n times n or between exactly n to m times n,m or at least n times expressed as open range at the upper limit n,.

Following is an example for all kinds of cardinalities except for exactly one occurance. The position of the cardinality is at the end of the triple constraint.

Example: The data specification for a contact has a class that has an attribute https://slovník.gov.cz/generický/kontakty/pojem/má-druh-kontaktu. The data type of the object following this predicate is xsd:string. This value should be present in the data graph in the inclusive range between two or three times. The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<217547b71513ecc5b06d7da60879e65akontaktShExShape> {
  <https://slovník.gov.cz/generický/kontakty/pojem/má-druh-kontaktu>
   ↪  xsd:string {2,3}
        // rdfs:label        "má druh kontaktu" ;
# The rest of the of the ShExC document
```

9.2.5. Associations properties

Associations provide only one extra property that has not been covered by either class or attribute properties and that is the fact, that we are connecting a class to another class as its attribute. The rest of the definition for class or attribute properties stays the same and the reader is advised to review two previous sections if interested in these general ideas.

Attaching association to a class as an attribute When the data specification contains a class with an attribute in the form of another class, we call that an association. The association is usually another non-trivial class, that contains its own attributes, but is also an attribute itself.

The generator recognizes two cases: an otherwise empty association and a non-empty association. Empty association is described only as a predicate that should have an IRI as an object. The non-empty association is attached to the parent class by the given predicate followed by a reference to the shape for given association class. Reference to another shape is done by at character @ followed by the IRI of referenced shape.

Example: The data specification for a tourist destination has a class that has one association to contact https://slovník.gov.cz/datový/turistické-cíle/pojem/kontakt. The referenced shape is the shape describing how contact's data structure should look like. The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<217547b71513ecc5b06d7da60879e65akontaktShExShape> IRI {
  a [<https://slovník.gov.cz/generický/kontakty/pojem/kontakt>] ;
  <https://slovník.gov.cz/generický/kontakty/pojem/má-url> xsd:anyURI
   ↪  /^https.+$/ +
        // rdfs:label        "má URL"
        // rdfs:comment        "Webová kontaktní adresa: webová stránka či
         ↪  WebID." ;
}
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> IRI /^https.+$/{
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
  <https://slovník.gov.cz/datový/turistické-cíle/pojem/kontakt>
   ↪  @<217547b71513ecc5b06d7da60879e65akontaktShExShape>
        // rdfs:label        "kontakt" ;
# The rest of the of the ShExC document
```

The empty association does not contain any attributes of its own. It is expected that this association will be represented in the data as an IRI and nothing else is required from the data of this association definition. This means that although the association points to a class, the shape representing it will not be present in the generated ShEx artifact. This class will only be represented by triple constraints which enumerates all of its important configurations such as predicate path, node kind (IRI in this case), optional IRI pattern and cardinality.

Example: The data specification for a tourist destination has a class that has arbitrary number of associations to available languages https://slovník.gov.cz/datový/turistické-cíle/pojem/. The only thing required of that association is for it to be an IRI. This behavior however can be changed in the class configuration settings. The generated ShEx schema part is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> {
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
  <https://slovník.gov.cz/datový/turistické-cíle/pojem/má-dostupný-jazyk>
   ↪  IRI *
        // rdfs:label        "má dostupný jazyk"
        // rdfs:comment        "Dostupný jazyk v místě turistického
         ↪  objektu."
# The rest of the of the ShExC document
```

9.2.6. Configure artifacts for data specification - Base URL and Instance identification and typing

In the package selection of data structures to be edited, there is a Generation of artifacts section lower on the page layout where the user can either Configure artifacts or Generate the artifacts to a .zip file. In the Configure artifacts pop-up window there are four inputs that can be made that interest the ShEx validation generator.

Right on top is optional Base URL input window where the user can specify the base URL to which all of the artifacts create relative addresses and names to. Following the base URL section are three inputs concerning the default behavior for instances of classes corresponding to created data structures. The first option dictates whether the instances of classes have to be identified by IRI or whether blank node is also feasible. The second input sets whether instances of classes have to be typed by the class identifier or not. The last input option tells in general whether classes allow more properties than stated in the data structure.

All inputs under Instance identification and typing header can be also set individually for each class while creating and editing the data structure. This setting is just used when no input at each class is manually overriding this general setting.

Base URL The base URL that user fills in the artifact configuration window is translated into the ShExC format that ShEx generator uses for output.

Example: Base URL is set to https://myexample.com/. The ShExL generator produces this line in the ShExC output prefix part:

```
# The rest of the prefixes
base <https://myexample.com/>
# The rest of the of the ShExC document
```

This base URL is then displayed as follows in the TURTLE body:

```
# Start of the ShExC document
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
base <https://myexample.com/>

<217547b71513ecc5b06d7da60879e65akontaktShExShape> IRI /^[A-Z]$/{
# The rest of the of the ShExC document
```

When there is no base URL provided by the user in the artifact configuration, the IRIs in the generator output are relative to no base URL. There is no base in the prefixes part in the ShExC format and the IRIs are encapsulated in < and > respectively from the left and the right side of the IRI in the ShExC format.

Example: Base URL is not set. The ShEx generator produces no line in the ShExC output prefix part. The body with relative IRIs is as follows:

```
# Start of the ShExC document
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape>{
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
  <https://slovník.gov.cz/datový/sportoviště/pojem/kapacita> xsd:integer *
        // rdfs:label        "kapacita" ;
# The rest of the of the ShExC document
```

Instance identification The user can choose whether the instances of the data specification are always going to have the IRIs identifying the nodes in the RDF format, if the IRIs are optional or they are banned altogether. This choice has an impact on which node kind the ShEx shape is going to expect from the respective class instance.

There are four node kinds in ShEx: Literal, IRI, BNode, and NonLiteral. Literal is not used in the Dataspecer environment as the instance identification only concerns the classes which are either IRIs or Blank Nodes. The NonLiteral includes both IRI and Blank Node node kinds. BNode is an abbreviation of Blank Node.

IRI setting is used when the instance identification is set as required, NonLiteral is used when the instance identification is optional and BNode is used when the instance identification is set to disabled.

Example: The user has chosen the identification of the class instances of tourist destination to be required. The generated ShEx output is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> IRI{
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
# The rest of the of the ShExC document
```

Example: The user has chosen the identification of the class instances of tourist destination to be optional. The generated ShEx output is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> NonLiteral{
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
# The rest of the of the ShExC document
```

Example: The user has chosen the identification of the class instances of tourist destination to be disabled. The generated ShEx output is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> BNode{
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
# The rest of the of the ShExC document
```

Instance typing The Dataspecer tool allows users to specify whether the class in the data structure will demand its instances to be typed. In other words, whether the data node in RDF graph will require the instances of this class to have predicate http://www.w3.org/1999/02/22-rdf-syntax-ns#type and a corresponding class type as an object of this triple.

Along with the attributes of the root of the data structure, this choice creates conditions for ShEx Query Map building, that is necessary for the generated ShEx schema to work properly when validating data. How the setting of this artifact configuration affects the final shape is described in section **??**.

As in instance identification, there are three possible user inputs. Instance typing required, optional and disabled.

When the instance typing is required, the shape demands the instance to have given class type with the help of RDF vocabulary predicate **rdf:type** or **a**. This construct ensures, that the ShEx validator checks, whether the target data node contains predicate rdf:type (abbreviated as a) and a corresponding class IRI as its object. The class type is inside a set of values, in this case only one value, that can be an object of the predicate rdf:type.

Example: The user has chosen the typing of the class of the tourist destination instances to be mandatory. The generated ShEx output is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> {
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
# The rest of the of the ShExC document
```

If the instance typing is optional, the ShEx generator does not generate any specific triple for the shape as the class may or may not exist in the data graph and both possibilities are correct.

Example: The user has chosen the typing of the class of the tourist destination instances to be optional. It does not contain predicate constraint for rdf:type. Instead other attributes are constrained in the shape. The generated ShEx output is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> {
  <https://slovník.gov.cz/datový/sportoviště/pojem/kapacita> xsd:integer *
        // rdfs:label        "kapacita" ;
# The rest of the of the ShExC document
```

If the instance typing of the class is banned by the user, the rdf:type cannot appear in the data graph. That condition in ShEx is represented by setting the cardinality of such property to zero. That condition means not allowing rdf:type to appear even once in the data graph relevant node.

Example: The user has chosen the typing of the class of the tourist destination instances to be disabled. The generated ShEx output is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> {
  a . {0} ;
# The rest of the of the ShExC document
```

Additional class properties The last configuration property for artifacts that concerns ShEx is whether the data structure class allows properties other than the explicitly specified ones. The possible inputs are that additional properties are allowed or are not allowed. ShEx implicitly allows other properties, so there is no action done for this case.

The case where additional properties are not allowed is handled by **CLOSED** modifier at the beginning of the shape. As ShEx has to specify if the data node describes its class type or not, there is no additional action to be taken to protect some base predicates such as rdf:type, which needs to be excluded from the closing of the shape in SHACL. If the user wants the data nodes to have rdf:type present, the predicate will be present in the ShEx shape.

Example: The user has chosen that other properties of the class of tourist destination are not allowed. The generated ShEx output is as follows:

```
# Start of the ShExC document
<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape> CLOSED{
  a [<https://slovník.gov.cz/datový/turistické-cíle/pojem/turistický-cíl>] ;
# The rest of the of the ShExC document
```

9.2.7. ShEx Query Maps

For the ShEx validator to know which data to validate against which ShEx shapes, it needs to have a mapping for this queue for validating. For ShEx that is done with the help of ShEx Shape Maps. There are two kinds of ShEx Shape maps: Fixed Shape Map and Query Shape Map. Fixed Shape Maps enumerate every data node that needs to be validated against a chosen shape in exhaustive way. As much as this approach is sufficient for one time validating or quick validation of known data parts against a known shape name, this approach has not been considered appropriate for ShEx map generator for Dataspecer where the generator does not have any knowledge about possible data nodes naming conventions and is trying to be as general as possible to allow automation and use for a big variety of possible data nodes.

Example: The user wants to validate two tourist destination data nodes against respective ShEx shapes and a notice board data node against the shape for contacts. The Fixed Shape Map would be as follows:

```
<destination-1>@<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape>,
<destination-2>@<d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape>,
<notice-board>@<217547b71513ecc5b06d7da60879e65akontaktShExShape>
```

Query Shape Map on the other hand uses pattern matching strategy to describe all the data nodes that fit the matched pattern to be validated against a chosen ShEx shape. It uses variable name FOCUS for the place where the matched data node fits the triple and underscore _ for any value. The second part of the map where the validating shape is referenced stays the same as for fixed shape map.

For better readability the following prefixes are used in the next examples:

```
prefix tur: <https://slovník.gov.cz/datový/turistické-cíle/pojem/>
prefix leg: <https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/>
prefix exm: <https://myexample.com/>
```

Example: The generated Query Shape Map is searching for data nodes that have rdf:type of tourist destination. The generated ShEx Query Map output is as follows:

```
{ FOCUS rdf:type tur:turistický-cíl}
↪ @exm:d7528e03c25e75bf0abc589ed5545ad5turistický_cílShExShape
```

This querying approach is fairly general and allows for various targetting possibilities, but for the sake of compatibility with the SHACL artifact generator, there were slight limitations put in place. The whole targetting process for SHACL is described earlier in this paper 9.1.6.

There are four main Query Shape Maps kinds generated for Dataspecer. In case the data specification does not fulfill any of those four options, an error is thrown for the user to see that an action needs to be taken to change the data specification in order to allow for seamless SHACL and ShEx artifacts generation.

Below are the cases where Query Shape Map is generated without problem:

1. **The root of the data structure always has unique type** If the rdf:type of the root class is unique, it can be used to find the data nodes that are the roots of the data graph according to the data specification. This can only be used when "Explicit instance typing" is set to "required", because if the type determination is missing in the data, the data node will not be found by the rdf:type predicate.

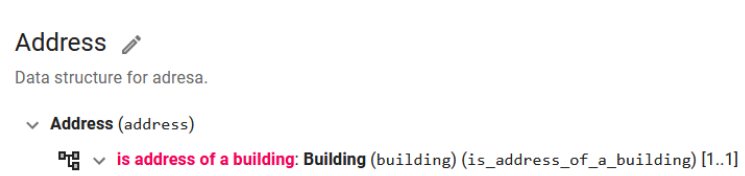

Fig. 8. Example of a data structure of an address that fulfills the aforementioned conditions for invoking use case 1 for generation of the Query Shape Map

Example: The root class of address has a unique rdf:type that is not found anywhere else in the structure and explicit instance typing for the root class is required. Its describing shape is the address shape. The generated Query Shape Map is as follows:

```
{ FOCUS rdf:type leg:adresa}
↪ @exm:09c5d2cfa6b4ca4b1a54034ae3a7ccc8adresaShExShape
```

2. **The root of the structure contains a unique predicate leading from it** If the root has a unique predicate in attributes with cardinality [1..x], it can be used for finding the data node at the root of the data structure. If the predicate's cardinality is [0..x], there is a chance the predicate is not available in data, not allowing the query shape map to find a FOCUS. The values of objects of this triple are pattern are not important for finding the subject and so the object part of the pattern is marked with _ signaling that the object can be arbitrary.
Example: The root class of address has a unique attribute that is not found anywhere else in the structure and its cardinality is [1..x]. Its describing shape is the address shape. The generated Query Shape Map is as follows:

```
{ FOCUS leg:je-adresou-stavebního-objektu _}
↪ @exm:09c5d2cfa6b4ca4b1a54034ae3a7ccc8adresaShExShape
```

3. **The root has an association, which has always unique type and the explicit instance typing of that association is required** If the rdf:type of the structure is unique, it can be used to find the data nodes that are the roots of the data graph according to the data specification. This can only be used when "Explicit instance typing" is set to "required" for the chosen association class, because if the type determination is missing in

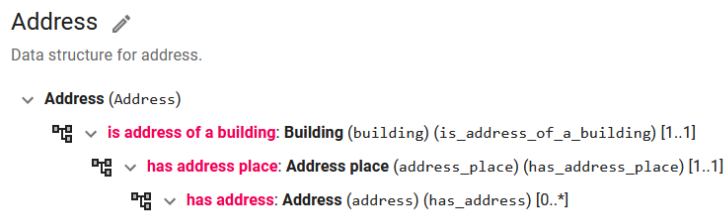

Fig. 9. Example of a data structure of an address that fulfills the aforementioned conditions for invoking use case 2 for generation of the Query Shape Map

the data, the data won't be checked against created shape. The only difference to 1) is, that the structure above the targetted class has to be found using reverse paths in the ShEx Shape (using ^<predicate-to-root> @<shape-for-root> inside the shape of the association with the unique type).

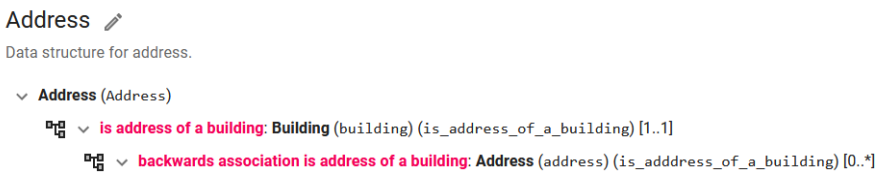

Fig. 10. Example of a data structure of an address that fulfills the aforementioned conditions for invoking use case 3 for generation of the Query Shape Map

Example: The root class of tourist destination has an association for a building with unique rdf:type that is not found anywhere else in the structure and explicit instance typing for the building class is required. The association has a cardinality of [1..x]. Its describing shape is the building shape. The generated Query Shape Map is as follows:

```
{ FOCUS rdf:type leg:stavební-objekt>}
↪  @exm:da1b1abf56e1b132c0073777da9af34stavební_objektShExShape
```

4. **The root has an association that contains a unique predicate leading from it** If the targetted attribute has a unique predicate in attributes with cardinality [1..x], it can be used for targetting by the unique predicate. The only difference to 2) is, that the structure above the targetted node has to be found using reverse paths (using (using ^<predicate-to-root> @<shape-for-root> inside the shape of the association with the unique type). The values of sh:targetSubjectsOf in a shape are IRIs. The values of objects of this triple are pattern are not important for finding the subject and so the object part of the pattern is marked with _ signaling that the object can be arbitrary.
Example: The root class of tourist destination has an association for a building with not unique rdf:type. But it has an attribute that is unique and has cardinality of [1..x]. Its describing shape is the building shape. The generated Query Shape Map is as follows:

```
{ FOCUS leg:má-adresní-místo> _}
↪  @exm:da1b1abf56e1b132c0073777da9af34stavební_objektShExShape
```

9.2.8. Conclusion

In this section we have discussed how the ShEx schema is generated containing various ShEx shapes and how the ShEx Query Map completes the necessary input for the ShEx validators to be able to target desired data nodes for validation with their descriptive shapes. For ShEx schema a mapping from Dataspecer structures to ShEx syntax for validation was shown. The reasoning behind generating ShEx Query Map was thoroughly described showing examples. The reader is encouraged to look at the final generated example data specification ShEx artifact with its ShEx Query Map to see the whole picture 9.2 again after going thorugh how each part of the schema is made.

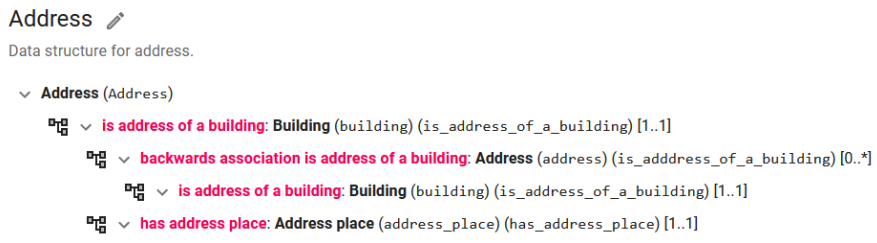

Fig. 11. Example of a data structure of an address that fulfills the aforementioned conditions for invoking use case 4 for generation of the Query Shape Map

9.3. Summary

In this chapter we have discussed how the SHACL and ShEx generators syntaxes and tools have been used in mapping onto Dataspecer data specification. We have gone through limitations imposed on the generators for validating schemas for RDF for Dataspecer's data specification and offered, how the user can solve this generation problem.

The semantics coded in both languages are the same, but they use different syntax. SHACL uses its own vocabulary from http://www.w3.org/ns/shacl# and defines its own terms for describing the data structure in the TURTLE format. ShEx does not have its own language and borrows terms from different already established vocabularies. SHACL generator has only one output, the SHACL shapes graph. ShEx generator produces two different outputs: the ShEx schema and the SheX Query Map.

To better understand the differences between SHACL and ShEx languages, the reader is encouraged to read a chapter called "Comparing ShEx and SHACL" from the book Validating RDF Data[4] which is also available for online reading.

10. Evaluation

10.1. Use case 1 - PPDF

Martin Nečaský

10.2. Use case 2 - OFN

Jakub Klímek

11. Conclusions and Future Work

Martin Nečaský/Jakub Klímek

References

[1] D. Kontokostas and H. Knublauch, Shapes Constraint Language (SHACL), W3C Recommendation, W3C, 2017, https://www.w3.org/TR/2017/REC-shacl-20170720/.

[2] A. Phillips and M. Davis, Information on BCP 47 » RFC Editor, Best current practice, RFC Production Center (RPC), 2009, https://www.rfc-editor.org/info/bcp47/.

[3] S.E.C. Group, Shape Expressions (ShEx) 2.1 Primer, Final Community Group Report, Shape Expressions Community Group, 2019, https://shex.io/shex-primer/.

[4] J.E. Labra Gayo, E. Prud'hommeaux, I. Boneva and D. Kontokostas, *Validating RDF Data*, Synthesis Lectures on the Semantic Web: Theory and Technology, Vol. 7, Morgan & Claypool Publishers LLC, 2017, pp. 1–328, http://book.validatingrdf.com/. doi:10.2200/s00786ed1v01y201707wbe016.

[5] J. Cowan and R. Tobin, XML Information Set (Second Edition), W3C Recommendation, W3C, 2004, https://www.w3.org/TR/2004/REC-xml-infoset-20040204/.

[6] H. Thompson, N. Mendelsohn, M. Maloney, M. Sperberg-McQueen, S. Gao and D. Beech, W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures, W3C Recommendation, W3C, 2012, https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/.

[7] M. Kay, XSL Transformations (XSLT) Version 3.0, W3C Recommendation, W3C, 2017, https://www.w3.org/TR/2017/REC-xslt-30-20170608/.

[8] H. Thompson, D. Hollander, T. Bray, A. Layman and R. Tobin, Namespaces in XML 1.0 (Third Edition), W3C Recommendation, W3C, 2009, https://www.w3.org/TR/2009/REC-xml-names-20091208/.

[9] S. Gao, D. Peterson, A. Malhotra, M. Sperberg-McQueen, H. Thompson and P.V. Biron, W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, W3C Recommendation, W3C, 2012, https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/.

[10] F. Yergeau, M. Sperberg-McQueen, T. Bray, J. Paoli and E. Maler, Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation, W3C, 2008, https://www.w3.org/TR/2008/REC-xml-20081126/.

[11] F. Gandon and G. Schreiber, RDF 1.1 XML Syntax, W3C Recommendation, W3C, 2014, https://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.

[12] D. Brickley and R. Guha, RDF Schema 1.1, W3C Recommendation, W3C, 2014, https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[13] J. Clark, S. Pieters and H. Thompson, Associating Style Sheets with XML documents 1.0 (Second Edition), W3C Recommendation, W3C, 2010, https://www.w3.org/TR/2010/REC-xml-stylesheet-20101028/.

[14] J. Broekstra and D. Beckett, SPARQL Query Results XML Format (Second Edition), W3C Recommendation, W3C, 2013, https://www.w3.org/TR/2013/REC-rdf-sparql-XMLres-20130321/.

[15] D. Wood, R. Cyganiak and M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, W3C, 2014, https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[16] A. Seaborne and S. Harris, SPARQL 1.1 Query Language, W3C Recommendation, W3C, 2013, https://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[17] H. Lausen and J. Farrell, Semantic Annotations for WSDL and XML Schema, W3C Recommendation, W3C, 2007, https://www.w3.org/TR/2007/REC-sawsdl-20070828/.

[18] J. Tennison, CSV on the Web: A Primer, W3C Note, W3C, 2016, https://www.w3.org/TR/2016/NOTE-tabular-data-primer-20160225/.

[19] G. Kellogg, J. Tandy and I. Herman, Generating RDF from Tabular Data on the Web, W3C Recommendation, W3C, 2015, https://www.w3.org/TR/2015/REC-csv2rdf-20151217/.

[20] G. Kellogg and J. Tennison, Model for Tabular Data and Metadata on the Web, W3C Recommendation, W3C, 2015, https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/.
[21] J. Tennison and G. Kellogg, Metadata Vocabulary for Tabular Data, W3C Recommendation, W3C, 2015, https://www.w3.org/TR/2015/REC-tabular-metadata-20151217/.
[22] H. Thompson, S. Gao, D. Peterson, M. Sperberg-McQueen, A. Malhotra and P.V. Biron, W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, W3C Recommendation, W3C, 2012, https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/.
[23] J. Tandy and I. Herman, Generating JSON from Tabular Data on the Web, W3C Recommendation, W3C, 2015, https://www.w3.org/TR/2015/REC-csv2json-20151217/.