ShEx-Lite: Automatic Generation of Domain Object Models from a Shape Expressions Subset Language *

Guillermo Facundo Colunga[2][0000−0003−1283−2763], Alejandro González Hevia[2][0000−0003−1394−5073], Emilio Rubiera Azcona[2][0000−0002−0292−9177], and Jose Emilio Labra Gayo[1][0000−0001−8907−5348]

[1] Dpt. of Computer Science, University of Oviedo, Spain labra@uniovi.es
[2] WESO Research Group, University of Oviedo, Spain
{thewilly.work alejgh.weso emilio.rubiera}@gmail.com

Abstract. Shape Expressions (ShEx) was defined as a human-readable and concise language to describe and validate RDF. In the last years, the usage of ShEx has grown and more functionalities are being demanded. One such functionality is to ensure interoperability between ShEx schemas and domain models in programming languages. In this paper, we present ShEx-Lite, a tabular based subset of ShEx that allows to generate domain object models in different object-oriented languages. Although the current system generates Java and Python, it offers a public interface so anyone can implement code generation in other programming languages. The system has been employed in a workflow where the shape expressions are used both to define constraints over an ontology and to generate domain objects that will be part of a clean architecture style.

Keywords: Linked Data · RDF · Shape Expressions · Validation.

1 Introduction

Since the appearance of Shape Expressions Language [4] (`ShEx`) the demands of the community on new tools based on `ShEx` have grown. One of those demands, born during the development of the Hercules ASIO European Project[3], was the creation of a tool that can automatically transform Shape Expressions into object domain models represented by means of an Object Oriented Programing Language. The object domain model generated will be part of a clean architecture based solution [2]. `ShEx-Lite`[4] was created as a tabular based subset of `ShEx` which enabled the automatic generation of domain object models from the schemas expressed with it. This paper describes how the domain object models are generated along with the architecture of the software that implements it.

[3] https://www.um.es/web/hercules
[4] https://www.weso.es/shex-lite/

2 Domain Object Model Generation

The ShEx language is quite powerful and allows for a high degree of expressiveness like disjunctions, negations, etc. which make generating domain object models from shapes a non-trivial task. However the ShEx-lite language is simpler and it permits mainly tabular based schemas based on constraints of type PROPERTY TYPE CARDINALITY referred to as simple triple constraints. For instance, Fig. 1 shows an schema example that defines the properties of a Person model. In this particular case a Person is defined as a property :name of type xsd:string and cardinality 1 *(Default one)* as well as a second property :knows of type @:Person and cardinality 0-n, which represents the set of people known by the current Person.

Person.shexl

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

Person.java

```
// Imports...
public class Person {
  private String name;
  private List<Person> knows;
  // Constructor...
  // Getters and Setters...
}
```

Fig. 1: Schema modeling a Person in ShExC syntax to the left. And the ShEx-Lite generated code in Java to the right.

Once defined the input of ShEx-Lite it is easier to explain how the system generated domain object models. Basically, the comunication with the system is done through a CLI tool that is provided. In this tool the users can define several options but the one that is in the scope of this paper is --java-pkg=STRING which triggers the java code generation and generates the target object in the specified package.

For example, for the input java -jar shexlc.jar --java-pkg=demo person.shexl where the person.shexl file corresponds to the schema defined at Fig. 1 ShEx-Lite generates a single java class with the code that appears at the Person.java file, also in Fig. 1.

From this process a number of questions raise, such as how the mapping process between the constraint types in the schemas and the object-oriented programming language is done, or what happens if the schema expresses something that the programing language is not able to represent as fixed cardinalities or repeated properties. The way ShEx-Lite solves this issue is by delegation [1]. It does not implicity check anything, it delegates to the specific code generators the ability to inform about any incompatibility and perform the corresponding mappings.

For example, by default, Java and Python code generation is built in with ShEx-Lite but the JavaCodeGenerator runs some checks that the PythonCodeGenerator does not and viceversa. If any incompatibility between the schema

```
error[E014]: feature not available
--> input_incorrect_schema_big_schema_2.shexl:15:24
    |
15  | schema:name            asdf:string
    |                        ^ this prefix has no mapping in java
```

Fig. 2: `ShEx-Lite` example error caused by a prefix with no mapping in java.

and the target language is found by the corresponding code generator, ShEx-Lite will let the user know by means of error or warning messages such as the once shown in Fig. 2.

3 ShEx-Lite Architecture

`ShEx-Lite` is available as open source software at GitHub[5]. It has been designed in accordance with the concept *"compiler as an API"*, born with the Roslyn compiler [3]. The feature of code generation was designed with the goal of being flexible enough to work with different target programming languages like `Java` or `Python`. The main components of `ShEx-Lite` follow a traditional compiler architecture and are represented in Fig. 3.

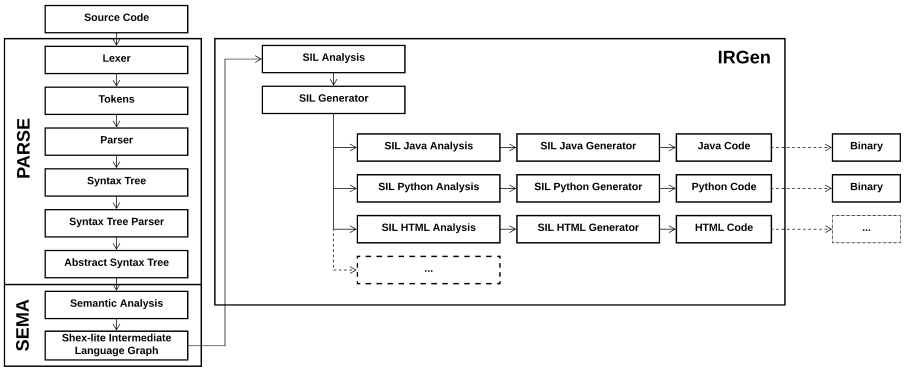

Fig. 3: `ShEx-Lite` internal architecture. SIL stands for ShEx-Lite Intermediate Language. IR stands for Intermediate representation.

– **Parse.** The components of this module aim at reading the source file and performing the syntax validation. The syntax validation of the schemas checks that the schemas defined follow the `ShEx-Lite` syntax. If this is not the case, the compiler lets the user know about the problem and possible solutions.

[5] https://github.com/weso/shex-lite

– **Sema.** At this stage the compiler checks that types are correct and that the invocations and references that occur in the schemas are defined. Also, during this process, if any error or warning is detected, the compiler will notify the user about the problem and possible solutions.
– **IRGen.** This module is the one actually generating target code (`Java`, `Python`, `Any...`). In order to allow adding other languages in the future, ShEx-Lite delegates the specific language checks and mappings and therefore it offers an interface that other language generators will implement. Each one of the code generators is responsible of checking that the constraints represented by the schemas are able to be expressed in the corresponding language. If the schema meets the requierements of the corresponding language, it is the specific code generator the one that produces the code.

4 Conclusions and future work

In this paper we have presented `ShEx-lite`, a subset of ShEx that allows to generate domain model objects in different Object-Oriented languages like Python and Java. The system proposed is being used in the Hércules project[6], and as a future work we're planning to incorporate the possiblity to read shape expressions from tabular formats like CSV[7].

Aknowledgements. The HÉRCULES Semantic University Research Data Project is backed by the Ministry of Economy, Industry and Competitiveness with a budget of 5.462.600,00 euros with an 80% of cofinancing from the 2014-2020 ERDF Program. This work has also been partially funded by the Spanish Ministry of Economy and Competitiveness (Society challenges: TIN2017-88877-R).

References

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: Abstraction and reuse of object-oriented design. In: European Conference on Object-Oriented Programming. pp. 406–431. Springer (1993)
2. Martin, R.C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson (2017)
3. McAllister, N.: Microsoft's roslyn: Reinventing the compiler as we know it. N. McAllister//InfoWorld from IDG.–2011 (2011)
4. Prud'hommeaux, E., Labra Gayo, J.E., Solbrig, H.: Shape expressions: an RDF validation and transformation language. In: Proceedings of the 10th International Conference on Semantic Systems. pp. 32–40 (2014)

[6] https://www.um.es/web/hercules
[7] https://github.com/dcmi/dcap