

ShEx-Lite: Automatic generation of domain object models from Shape Expressions

Guillermo Facundo Colunga^{2[0000-0003-1283-2763]}, Alejandro González Hevia^{2[0000-0003-1394-5073]}, Jose Emilio Labra Gayo^{1[0000-0001-8907-5348]}, and Emilio Rubiera Azcona^{2[2222-3333-4444-5555]}

¹ Dpt. of Computer Science, University of Oviedo, Spain labra@uniovi.es

² WESO Research Group, University of Oviedo, Spain
{thewilly.work,alejgh,rubiera}@gmail.com

Abstract. The quality of RDF based solutions requires the possibility to describe and validate RDF graphs. In the last years the community behind one of the technologies for RDF validation, ShEx, has grown and more functionalities ensuring interoperability between ShEx and programming languages are demanded. **ShEx-Lite** is playground compiler for a syntax based on a subset of the Shape Expressions Compact Syntax that allows to generate domain object models in multiple object oriented languages from shape expressions. The system was mainly designed for **Java** and **Python** but it offers a public interface so anyone can implement code generation in any programming language. The system has been employed in production workflows and it is being extended to more fields of application.

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

1 Introduction

Since the appearance of Shape Expressions Language [2] (**ShEx**) the demands of the community on new tools based on **ShEx** have exponentially grown. One of those demands, born during the development of the Hercules ASIO European Project³, is the creation of a tool that allows to automatically transform Shape Expressions, which represent a domain model, into an **object** domain model represented by means of Object Oriented Programming Languages [1] (OOPL). **ShEx-Lite**⁴ was created as a playground compiler, based on a subset of the **ShEx** Compact Syntax⁵ (**ShExC**), in order to implement new functionalities before merging them in to the full **ShEx** ecosystem. One of those functionalities is the automatic generation of domain object models from the schemas expressed on the **ShExC** subset, called **shexl**. This paper elucidates how the domain object models are generated along with the architecture of this feature inside ShEx-Lite.

³ <https://www.um.es/web/hercules>

⁴ <https://github.com/weso/shex-lite>

⁵ <https://shex.io/shex-semantic/#shexc>

2 Domain Object Model Generation

The feature of generating domain object models from shapes is not trivial. The **ShEx** language is remarkably powerful and allows for a high degree of expressivity. However the **shexl** syntax is simpler. It permits merely simple schemas based on constraints of type **PROPERTY TYPE CARDINALITY** referred to as simple triple constraints. For instance, Fig. 1 shows an example of an schema that defines the properties of the **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	Person.java
<pre> 1 # Prefixes... 2 :Person { 3 :name xsd:string ; 4 :knows @:Person * 5 }</pre>	<pre> 1 // Imports... 2 public class Person { 3 private String name; 4 private List<Person> knows; 5 // Constructor... 6 // Getters and Setters... 7 }</pre>

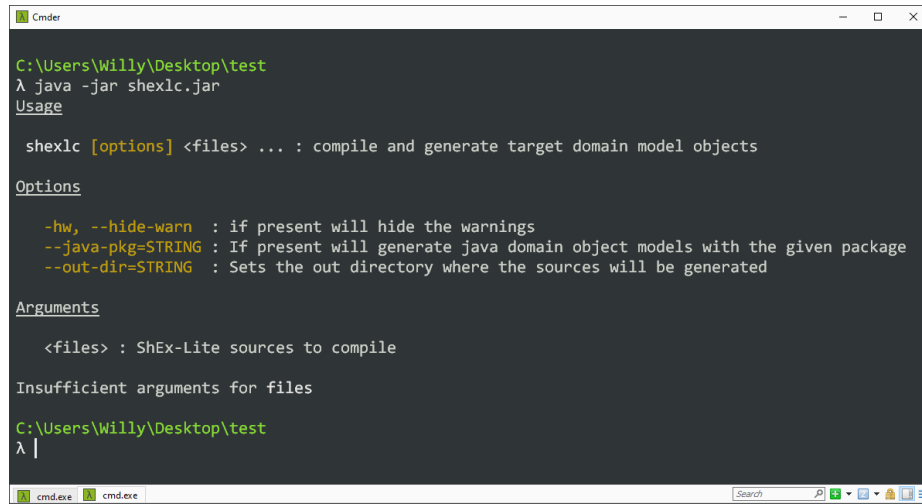
Fig. 1: Schema modeling a **Person** in **shexl** syntax to the left. And the **ShEx-Lite** generated code in **Java** to the right.

Once defined the input of **ShEx-Lite** it is easier to explain how the system generated domain object models. Basically, the communication with the system is done through a CLI tool that is provided, illustrated by Fig. 2. In this tool the users can define multiple options but the one that is in the scope of this paper is **--java-pkg=STRING** which if present will trigger the java code generation and will generate 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** would generate 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 OOPL is done, or what happens if the schema expresses something that the programming language is not able to represent as fixed cardinalities or repeated properties. The way **ShEx-Lite** solves this issue is by delegation. It does not implicitly 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 build in with **ShEx-Lite** but the **JavaCodeGenerator** runs some checks that the **PythonCodeGenerator** does not and viceversa. If any incompatibility between the schema 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. 3.



```

C:\Users\Willy\Desktop\test
λ java -jar shexlc.jar
Usage

shexlc [options] <files> ... : compile and generate target domain model objects

Options

  -hw, --hide-warn : if present will hide the warnings
  --java-pkg=STRING : If present will generate java domain object models with the given package
  --out-dir=STRING : Sets the out directory where the sources will be generated

Arguments

  <files> : ShEx-Lite sources to compile

Insufficient arguments for files

C:\Users\Willy\Desktop\test
λ |

```

Fig. 2: CLI menu of ShEx-Lite CLI tool.

```

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

3 ShEx-Lite Architecture

ShEx-Lite is available as open source software under the MIT license at GitHub⁶. It has been designed in accordance with the concept “*compiler as an API*” and 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**. A detailed view of all the components of ShEx-Lite can be found in Fig. 4.

- **Parse.** The components of this module aim at reading the source file and performing the syntax validation. The syntax validation of the schemes checks that the schemas defined follow the **shexl** syntax. If this is not the case, the compiler let the user know about the problem and the possible solutions.
- **Sema.** At this stage the compiler checks that types are correct and that the invocations and references that occur in the schemas are defined. If during this process 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,

⁶ <https://github.com/weso/shex-lite>

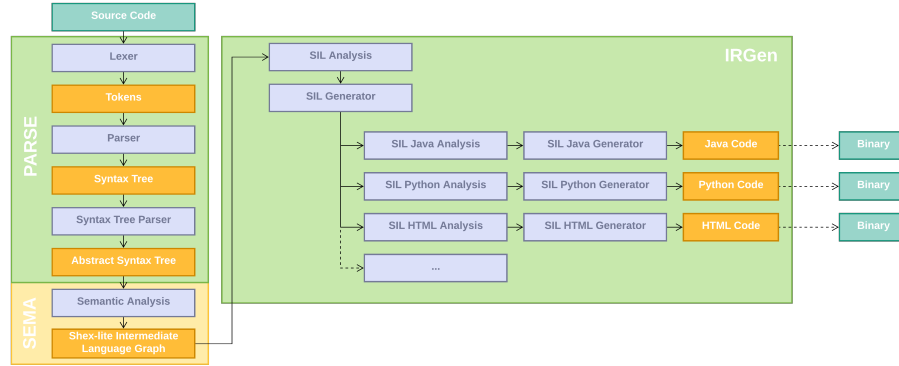


Fig. 4: ShEx-Lite internal architecture.

ShEx-Lite delegates the specific language checkings and mappings and therefore it offers an interface that specific 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 requirements of the corresponding language, it is the specific code generator the one that produces the code.

ShEx-Lite has already been used in several projects⁷, and as a future work we're planning to incorporate the possibility to read shape expressions from tabular formats⁸.

Acknowledgements. 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.

References

1. Ledgard, H.F.: The little book of object-oriented programming. Prentice-Hall, Inc. (1995)
2. 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)

⁷ <https://www.um.es/web/hercules>

⁸ <https://github.com/dcmi/dcap>