Node Server - External DSL - Individual Project

University of Southern Denmark

(F18 2018)

Report

Christos Anestis Georgiadis
chgeo17@student.sdu.dk

Supervisor: Ulrik Pagh Schultz

February, 2018 - May, 2018

Contents

1 Overview

The goal of the present individual assignment, is to expand the potential of the previous group project. Based on the knowledge gained in regard to the project and acknowledging the deficiencies of the DSL, crucial improvements were made. The main focus, was on increasing syntactic support and making the development easier for the user, by providing more specific errors during the program's development. Some extra features were also added, so as to make the recognition among the ServerEntity and MemberEntity more efficient. In addition, the possibility of creating more than one ServerEntity in different parts of the code was accomplished.

2 Metamodel and Example

Figure 1 shows the Metamodel for the Node Server DSL with the improvements that were made on it. As it can be easily noticed, "ServerEntity" and "MemberEntity" are both belonging to the Entity, making the separation of them more difficult.

More specifically, the problem was in recognizing whether an Entity was "Server-Entity" or "MemberEntity". On the previous edition of the project, this problem was solved by requesting from the developer to write the "ServerEntity" on the top of the program, when starting the development. Thus, in the generator file just checking the first Entity of the program was enough in order to recognize it.

In this version of the DSL, an extra field in the "ServerEntity" with the name "is" and type of "IsServer" was added. The last mentioned type, contains the value "SERVER". By importing the above features in the Metamodel, a simplification on the detection of the "ServerEntity" was achieved, by just checking if “is” is equal to "null" or not. In the first case the Entity is a "MemberEntity" and in the second case it is a "ServerEntity".

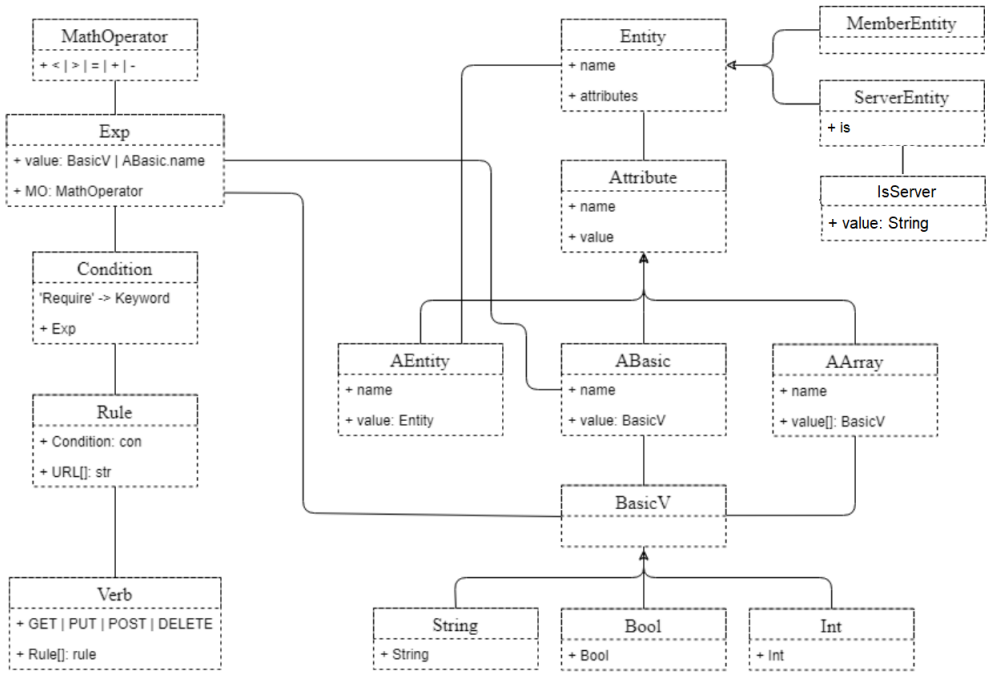

Figure 1: Improved Metamodel for the Node Server External DSL

Moreover, the most important and functional improvement, is that the developers could be able to create more than one "ServerEntity" among the code. In order to achieve that, the grammar and the generator file were changed. In the current version, the DSL produces as many ".js" files as the number of "ServerEntity". Below, a small example of the DSL code is presented.

```
#mydb
-> DBDomain: "www.sdu.dk"

#Server SERVER
-> DOMAIN:"www.elamou.com"
-> PORT: 9999
-> Running: true
-> DB: mydb
-> URL:
  - "Career.html"
  - "Map.html"
  - "Home.html"

#Server2 SERVER
-> DOMAIN:"www.blah.com"
-> PORT: 1234
-> Running: true
-> DB: mydb
-> URL:
  - "Index.html"
  - "About.html"
  - "Contact.html"

@get Server
REQUIRE (5 * 34 *23 * 23 || 12 > 10 && (False != True))
-"Index.html"

@post Server2
REQUIRE(Server.PORT != 1234 && mydb.DBDomain == "www.sdu.dk")
- "About.html"
- "Contact.html"
```

Listing 1: DSL example

As it can be assumed, the developer has to specify the name of the "ServerEntity" every time that "Verb" is utilized, in order to perform the proper action on the right Server. So as to achieve the above, the field "qa" in the "Verb", that saves the name of the "ServerEntity", was added. Furthermore, the above code generates the files "Main.js", "mydb.js", "Server.js" and "Server2.js".

3 Validation

In this chapter, the improvements of the validation in the DSL are presented and explained. Firstly, a part of the validation from the generator file, was deleted, since it was not providing detailed information regarding the errors. In order to increase the syntactic support, the validation file was developed.

Firstly, it should be checked, whether all the requirement fields (DOMAIN, PORT, URL, amountOfRequests) exist inside of each "ServerEntity" and then if they are of the right type. The pre-mentioned fields, could be attributed with the names "DOMAIN", "PORT", "URL", "amountOfRequests" in the "MemberEnity", but neither their existence is a requirement nor their type should be specific. More specifically, for the “DOMAIN” attribute, it is possible to check the type and the pattern of "StringType". When checking the type, an "error" is indicated if it returns false, while a "warning" is indicated for the pattern. Regarding the “PORT” and the "amountOfRequests", the possibility of checking the type of "IntType" is also possible. Lastly, a checking associated with the URL is taking place, regarding on the type of "ArrayType" and each value of the type of "StringType".

Moreover, there is a check about the return value of a Verb in the validation file being ArrayType of StringType and a check in the generator file about the name of the ServerEntity side by the Verb. If the last check is not true then throws an error. Below, the code regarding the return type and the name check are presented respectively in Figure 2 and Figure 3.

```
@Check
def checkVerb(Verb v){
    try{
        var i = 0;
        for(i = 0; i <v.rules.get(v.rules.size-1).url.arrayElements.size; i++){
            val array = v.rules.get(v.rules.size-1).url.arrayElements.get(i).value
            switch(array){
                StringType: array.value
                default: error("Return Array of URL should be a String.", MyDslPackage.Literals.VERB__VERB)
            }
        }
    }catch(Exception e){
        new Exception(e.toString)
    }
}
```

Figure 2: Checking the return type.

```
//Checking about ServerEntity name on Verb
var serverNames = new ArrayList()
for(e : serverEntities){
    serverNames.add(e.name)
}
for(v: verbs){
    if (!serverNames.contains(v.qa)) {
        throw new Error("One or more Verbs have wrong ServerEntity name.")
    }
}
```

Figure 3: Checking the name of the ServerEntity.

4 Conclusion

To conclude, this external DSL is developed in order to help developers create their own custom node server template with custom configuration parameters, in less time. New features have implemented in order to expand the functionalities of it and make it more useful. Obviously there are many improvements that could be implemented if we want to create a real useful DSL.

A Appendix

Node Server - External DSL

University of Southern Denmark

(F18 2018)

Report

Nermin Sehovic
neseh13@student.sdu.dk

Tom Darboux
todar18@student.sdu.dk

Christos Anestis Georgiadis
chgeo17@student.sdu.dk

Supervisor: Ulrik Pagh Schultz

February, 2018 - May, 2018

Contents

1 The Domain

This external DSL is built to fit inside the web application development domain. More specifically, the node.js technology development domain. The goal of the DSL is to automate the creation of a node server and its belonging configuration parameters.The DSL is meant to be used by a developer to create his own custom node server template with custom configuration parameters. This will save considerable time, effort and potential error search and debugging later thanks to the code generation.This can also have the potential to save financial resources. Figure 1 shows the intended flow of a software project originating from this DSL. The template will be generated by the generator from the grammar specifications, and the developer can then use it to make various web applications.

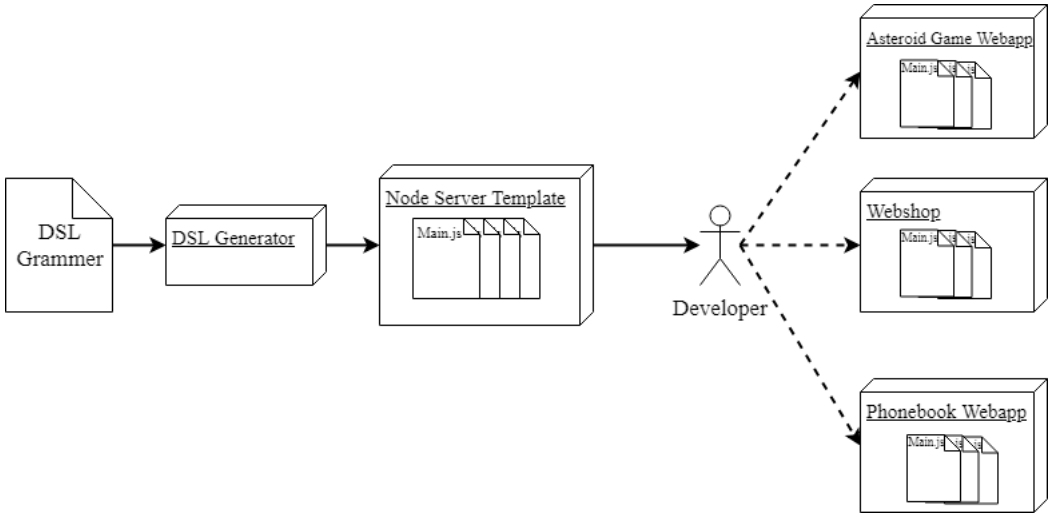

Figure 1: The DSL generates a custom template to the developer that can be used to create various web applications

2 Metamodel

Figure 2 shows the Metamodel for the Node Server DSL. The two top elements in the Metamodel are the 'Entity' and the 'VERBS'.

An entity has a name and must contain at least one attribute. Entity could be a 'ServerEntity' which will be the entity that will contain server properties or can be a 'MemberEntity' which represents all other aspects of the server, like DB connections, GUI framework etc.

An attribute has also a name, but also a value. This value can be different types which are the 'ABasic' that goes further down to 'BasicV' that finally takes the shape of either a 'String', 'Bool' or an 'Int' type. An attribute can also be an 'AArray' which is just an array of basic values 'BasicV'. The last thing an attribute can be is another existing entity that the attribute can reference to.

A 'VERB' can be one of the four verbs 'GET', 'PUT', 'POST' and 'DELETE'. Each 'VERB' must contain at least one 'Rule'. This rule will contain a 'Condition' and an array of URLs that the condition will apply to. A 'Condititon' requires the 'REQUIRE' keyword together with an expression. An expression 'Exp' can contain basic values 'BasicV', attribute values from entities 'ABasic.name' and math operators like +, -, =, etc.

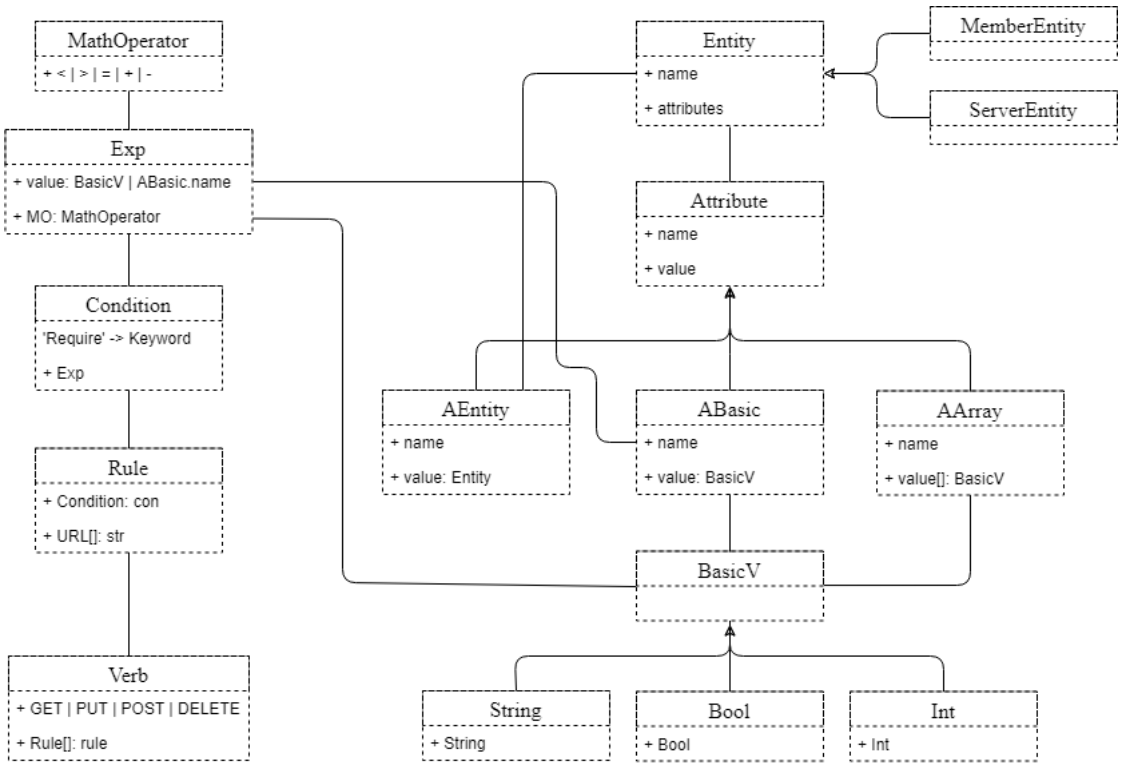

Figure 2: Metamodel for the Node Server External DSL

3 Example Program

The DSL code in listing 1 shows an example of how to create a simple server in the DSL language. The DSL requires the user to first create a server entity which is different from other entities in that it has the 'SERVER' keyword and that it has mandatory attributes 'PORT', 'DOMAIN', 'URL' and 'amountOfRequest: 0'. The user is still able to add custom attributes. A member entity 'mydb' is created after the server entity. This 'mydb' entity is used as an attribute in the server entity.

A user firstly defines the entities and then the verbs follow. This way of setting up the DSL structure is the recommended way, but is not enforced. A '@get' verb is defined that sets the 'Index.html' page as the target for GET requests. And a condition is defined saying that the 'amountOfRequest' server attribute has to be below 100 and that the 'PORT' server attribute has to have a value less than or equal to 65535.

The same pattern is repeated for another verb '@post'. This structure means that a user only need to define a particular verb once and not multiple times even though he is allowed to do that. If a new page is added to the web application, the user only needs to append the name of the page to the existing verb.

```
#Server SERVER
-> DOMAIN:"127.0.0.1"
-> PORT:1234
-> Running: true
-> DB: mydb
-> amountOfRequests: 0
-> URL:
  - "Index.html"
  - "About.html"
  - "Contact.html"

#mydb
-> DBDomain: "www.sdu.dk"

@get
REQUIRE (Server.amountOfRequests < 100 && Server.PORT <= 65535)
-"Index.html"

@post
REQUIRE(Server.PORT != 1234 && mydb.DBDomain == "www.sdu.dk")
- "About.html"
- "Contact.html"
```

Listing 1: DSL example

4 Code Generation

The DSL generates javascript files. Each entity will have its corresponding javascript file which will contain a JS object that represents the entity. Beside these files, another javascript file called Main.js will always be generated. This is the file that will be run by Node to start the server. The 'Main.js' file contains URL definitions and the basic setup of the server.

Figure 3 shows the files that have been generated from the DSL specifications in listing 1. A 'Main.js' file together with a 'Server.js' and 'mydb.js' files representing the entities. The dotted 'Other.js' file is present to illustrate that the DSL is able to generate more files besides those in this example.

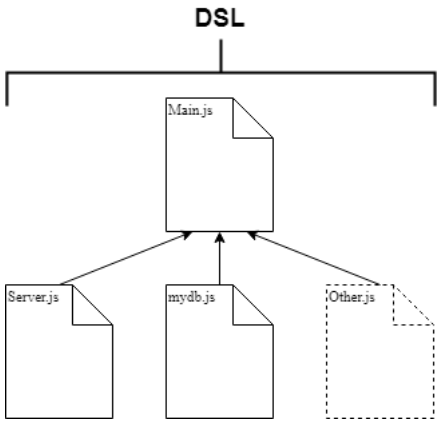

Figure 3: Code generated from the DSL Specifications in listing 1