UNIVERSAL LANGUAGE FOR FINITE STATE MACHINE DEFINITION

Vasile PĂPĂLUȚĂ[1*], Constantin CAZACU[1], Anastasia GHEORGHIȚĂ[1], Pavel CERLAT[1], Diana OLEDNIC[1]

[1]*Technical University of Moldova, Faculty of Computers, Informatics and Microelectronics, Department of Software Engineering and Automatics, FAF-192, Chișinău, Republic of Moldova*

*Corresonding author: Vasile Păpăluță, papaluta.vasile@isa.utm.md

Abstract. *This article is about a Domain Specific Language (DSL) called LOONA Language that has been developed to serve as a universal language for defining Finite State Machines (FSMs). This will serve as a universal guideline by which developers can create and implement their own state machines.*

Keywords: *Domain Specific Language, Finite State Machine, Syntax, Grammar, Python.*

Introduction

Finite State Machines these days have a lot of uses. They are used in the automotive industry, especially in autonomous vehicle development for parallel parking. In Fin-tech they are used for transitions and other types of financial operations for controlling the different steps of them. In game-development they describe non-player character (or NPC) behavior [1]. These are only some of the fields where FSMs are used, however all these fields like a universal language for defining them, like HTML or SQL. All of them are defined by the developers in their unique way that depends on their skills and experience. These facts are limiting the developing and replication of them on different projects if the developing leaves the company [2].

Solution

LOONA aims to be this universal language for defining FSMs, using simple writing rules. It allows to easily define the main states used by the automaton or FSM, without defining other variables that can disturb the FSM's running process. Also, it allows to easily describe the transition process, directly describing the change that must be performed in the program environment, skipping in this way the development of the special logic to change state of the FSM. Also, to make the process of state transition easier, LOONA allows to define the so named gateways - the gateways between the FSM and the user program. The (python) program must send to the FSM a dictionary with the values related to keys named as the gateways, and the FSM returns the new state (updated after the transition), which the program can use for its own scopes.

As it was said before, FSM has a lot of uses in a lot of different industries. Just to dive deeper in those enumerated before, here are some more examples. Interest of the automotive industry in designing intelligent systems capable of operating autonomously and safely beyond the linear region limits is constantly growing. Using the DSL would have its advantages: it would be clear in structure as it will be explicitly readable, so the behavior of automated controllers could be easy to predict; it will be reliable, because an FSM usually has a finite number of parameters, so that is easy to calibrate and optimize.

FinTech industry - the DSL can help this industry by facilitating the process of assisting companies, business owners, consumers etc. for better management of their financial operations, processes, and lives in general. The DSL can provide a better performance of computations, a greater accuracy of models, secure development, reliable results and more productivity.

Entertainment – usually, FSM is used in video games for the creation of rudimentary effective AI. A finite state can be used to define certain non-playable character (NPC) behavior, such as attacking, roaming or running. FSM could be used for mimicking a player strategy and

attack pattern in a fighting video game, by splitting the data into 2 tiers, strategic data and tactical data. Using the DSL for video games contains a high level of abstraction that allows to define all the elements that are necessary for the generation of a videogame, as well as the establishment of the required values to assign a behavior to the actors [3].

Legal aspect - using a DSL can provide more visibility for both sides of enterprises, improved cost savings, and a better performance.

The high use of FSMs in industry, made some software companies to develop their own frameworks and even languages, for defining these FSMs. Tinder - the famous dating app created Tinder StateMachine a DSL for defining Finite State Machines and Automata implemented in Kotlin. Voxa is another framework for FSM used by Alexa Skills, Google actions Facebook Messenger and Telegram bots implemented in node.js. Finally, the Netflix conductor for orchestrating microservices, implemented by the mentioned company [4 - 6].

Implementation

Basic computation that the DSL performs:

The DSL that will be created, will generate an automaton in the form of a matrix. This matrix will be contained in a class and other FSMs can be created by this new created class.

Basic data structures in the DSL:

The language will have the following data structures:

1. State - it represents the usual state in an automaton or Final State Machine. It is defined by the following syntax:

state <name of the state>

If the user wants to make a state the starting one it will write the following, syntax:

state <name of the State>(start = True)

or if it needs to be defined as a killer state, use the following syntax:

state <name of the State>(killer = True)

2. State transition matrix (STM) - an actual python dictionary with 2 keys that will implement the state transition matrix, inside the class factory.

3. Gateways - the gateways to user defined input, that will influence the FSM states. They are defined by the following syntax:

gateway <name of the gateway> <dtype of the input>

When defining a gateway, the user must define its data type, that can be:

- numerical;
- string (str);
- Boolean (bool).

Basic control structures in the DSL

The main control structure of the language is the STMs (Phobos and Deimos) - Phobos is the matrix built by the DSL after reading the code file. Usually, the Phobos is defined by the following syntax:

<state name 1> → <state name 2> : <change condition>

Usually changing conditions are defined using the input from the gateway.

The Phobos must look like the table shown in *Table 1*.

After the Phobos is built, on its base is built the Deimos - a python dictionary with a tuple as an input and a new state as the value related to it. The tuple will contain 2 values, the state and the action, but the value will be represented by the new state.

Input/Output

The input of the program will be a .txt file with the definition of the state machine. See the example below in code snippet 1:

Table 1

State-transition table (S: state, I: input, O: Output)

Input	Current State	Next State	Output
I_1	S_1	S_i	O_x
I_2	S_1	S_j	O_y
…	…	…	…
I_n	S_1	S_k	O_z
I_1	S_2	$S_{i'}$	$O_{x'}$
I_2	S_2	$S_{j'}$	$O_{y'}$
…	…	…	…
I_n	S_2	$S_{k'}$	$O_{z'}$
…	…	…	…
I_1	S_m	$S_{i'}$	$O_{x'}$
I_2	S_m	$S_{j'}$	$O_{y'}$
…	…	…	…
I_n	S_m	$S_{k'}$	$O_{z'}$

Code snippet 1. The example of the code in LOONA,

```
state Start(start=True)
state FL
state FR
state FI
state BL
state BR
state BI
state SL
state SR
state SF
state trap(trap=True)

Start → FI : *
FI → SF : threshold(dist1, 4, min)
SF → FI : threshold(calc_dist, car_lenght, max)
FI → BS : timepassed(4)
BI → BR : timepassed(6)
BR → BL : timepassed(6)
BL → FI : timepassed(6)
FI → SI : timepassed(3)
```

```
gateway gate1 numerical
gateway gate2 numerical
gateway dist1 numerical
gateway gate4 numerical
gateway gate5 numerical
gateway gate6 numerical
gateway calc_dist numerical
gateway car_lenght numerical
```

The output of the model will be a python class that will be used to create other state machines.

Error Handling

The language will have the following errors, which it will show:

1. Error1 : Start state isn't defined: *row_number* = <*n*>:

This error raises when the starting state is not defined.

2. Error2 : Parameter <*name of the parameter*> is not defined : *row_number* = <*n*>:

This error arises when a parameter is used by a function, but is not defined in the gateway.

3. Error3 : State <*name of the state*> isn't defined : *row_number* = <*n*>:

This error arises when a state is used in the STM definition but isn't defined.

4. Error4 : No such input type : *row_number* = <*n*>:

This error is raised when a gateway is defined with a non-existent dtype.

5. Error5 : No such transition function as <*fun_name*> *: row_number* = <*n*>:

This error is raised when a transaction function that doesn't exist is used when the STM is defined.

Conclusion

LOONA language aims to use simple writing and a very restrictive syntax in order to streamline the process of implementation of Finite State Machines in various environments such as: automotive, FinTech, entertainment and the legal aspect.

References:

1. EDMUND L., *Enhanced NPC Behaviour using Goal Oriented Action Planning* , MSc dissertation, Dundee, Scotland, University of Abertay Dundee, 2007
2. JOHN E.H., RAJEEV M., JEFREY D.U., *Introduction to automata theory*, Addison-Wesley, 2001.
3. SAINI S., CHUNG P.W.H., DAWSON C.W., Mimicking Human Strategies in Fighting Games using a Data Driven Finite State Machine, *Proceedings of the 6th IEEE Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, Chongqing, China, 20 - 22 August 2011, pp. 389 - 393
4. Tinder State Machine [online], [accessed 04.02.2021 20:30PM] Available:https://github.com/Tinder/StateMachine
5. FSM for digital assistants [online], [accessed 04.02.2021 20:45PM] Available:https://github.com/VoxaAI/voxa
6. Netflix Conductor: A microservices orchestrator [online], [accessed 04.02.2021 21:20PM] Available:https://netflixtechblog.com/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40