DOMAIN SPECIFIC LANGUAGE FOR INTERACTIVE STORYTELLING MANAGEMENT

Ștefan BERESTEAN, Anna CHIRICIUC, Cătălin TINCU, Elena GRAUR*

Software Engineering and Automatics Department, Group FAF-201, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, mun. Chișinău, Republic of Moldova

*Correspondent author: Elena Graur, elena.graur@isa.utm.md

Abstract. *This paper discusses the implementation process of a domain specific language (DSL) for creating a visualization in acyclic directed graph form of an interactive story telling. Therefore, the DSL for creating the graph of the flow allows an easier modification and visualization of the story.*

Keywords: *DSL, interactive storytelling, graph, grammar, storytelling-management*

Introduction

The interactive storytelling industry has become a mainstream phenomenon. Games, for example, are now as ubiquitous as movies, books, and other forms of popular culture. The industry's growth might even be outpacing that of the more classical art forms. Nevertheless, the development of games is currently very labor intensive. However, currently many tasks still require intense participation from the development staff.

For a better understanding of the problem, it is necessary to look into the organizational structure of medium to large-sized game development studios. At a high level, there are various common departments that are present in a development studio for an interactive story scenario: character description, actions, timeline, events [1]. These broad departments might be partitioned into smaller sections depending on the size of the studio. In other words, the various artistic designers must communicate their ideas and intent to the programmers. Not only can take much time and effort, it can also easily introduce misunderstandings.

This inefficient and failure-prone communication channel can really hamper team productivity. It would be much simpler, and indeed more efficient, if the scenarists could "code" their ideas in a way that is both natural for them, but also usable for the production of the game software. For some of the domains involved, a domain specific language (DSL), could be a significant part of the solution, and especially a language that will translate a code into a schema where will be visible all the actions, characters, locations and interaction in that way that would be easier to understand main principle of story [2].

To offer a better understanding of the solution, it is introduced the definition of a domain specific language as a specialized language used for a specific purpose, to solve specific problems [3].

1 Domain analysis

Storytelling is the vivid description of ideas, beliefs, personal experiences and life-lessons through stories or narratives that evoke powerful emotions and insights. Most features that are considered as gameplay process in an interactive story are about how the player moves their character, how they interact with the environment, and how they interact with other characters. Interactive storytelling has a vast range of target audience [4]. It can be applied anywhere, for example, in studying process (outside of gaming); it includes different themes and genres.

Unlike the classical linear structure of the flow of the narrative story, interactive storytelling has a non-linear structure, either it is a branching story structure, or a parallel path structure, or threaded story structure, or even dynamic hierarchical story structure. The proposed DSL is a tool aimed to manage an interactive storytelling by creating a graph representation of the story flow. Given a plain text input into a specific form, the DSL outputs the graph that corresponds to the described instances. Therefore, this means that the user describes only the logic of the computation

The described DSL will contribute to solving such problems as internal consistency (coherency and plot holes that go against the flow of logic established by the story's plot) so that none of the future events contradict the previous ones. Furthermore, the elimination of factual errors, impossible events, unbelievable character choices, illogical plot developments, unresolved storylines and continuity errors can be easily avoided by creating a DSL that will analyze the storyline. Comfortable use of this tool will ensure more accessibility to people who are not familiar with any general-purpose programming language.

2 Grammar

The domain specific language design includes more stages, one of them being the grammar of the language development. For a better understanding of how the proposed DSL is supposed to work, below it is presented its structures.

First of all, it is necessary to define the terms the grammar operates [5]. As follows, the grammar of the proposed language is described as follows: $L(G) = (S, P, V_N, V_T)$, where:

- S – start symbol;
- P – finite set of production of rules;
- V_N – finite set of non-terminal symbols;
- V_T – finite set of terminal symbols.

The notations presented in the Tab. 1, represents the meta notations used for the grammar description, inclusively to mark some key points of it.

Table 1

Meta notations

Notation	Meaning
<foo>	means foo is a nonterminal
foo	means foo is a terminal
x*	means zero or more occurrences of x
|	means separates alternatives
→	means deriving
{}	means optional occurrences

That being given, below it is represented the grammar itself, including the mention of the start symbol, the non-terminal and terminal symbols, as well as production rules, aimed to obtain each entity in the described DSL.

S = {<source code>}

V_N = {<source code>, <set of affirmations>, <act description>, <name>, <flow description>, <stagename>, <section name>, <digits>, <nenule digits>, <operation>, <type>, <string>, <letter>, <conditions>, <condition>, <identifier>, <numeric type>, <variable declaration>, <variables declaration>, <variable>, <variable attribution>, <attribution>, <link construction>, <condition>, <expression>, <operand>, <composed operand>, <function>, <comments>}

V_T = {0.9, a.z, A.Z, STORY, STAGE, SECTION, int, string, FROM, TO, ACTION, if, else, while,for, in, not, "_", "&&", "||", ":", " ", " ", "-", ",", ".", ";", ">", "<", "=", "!", "?", ":", "+", "-", "*", "/", "%","(", ")", nr_nodes, nr_sections, nr_stages, width, nr_of_interactions, color, style}

P = {

<source code> → <variables declaration><set of affirmations>{<comments>}

<source code> → <set of affirmations>

<variables declaration> → <variable declaration> | <variable declaration><variables declaration>

<variable declaration> → <type><variable>

<variable> → <identifier>

<type> → **int** | **string**

<identifier**>** → <letter> | _ (<letter> | <digits> | _)*
<letter> → **a** | **b** | **c** | … | **A** | **B** **C** | … | **Z**
<digits> → **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
<nenule digits> → **1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
<set of affirmations> → <set of affirmation> | <set of affirmation> <set of affirmations>
<set of affirmation> → <act description><flow description> | <variable attribution>{<set of affirmations>} |**if** <conditions>**: <**set of affirmations**> | if** <conditions>**: <**set of affirmations**> else:** <set of affirmations> | **while** <conditions>**: <**set of affirmations**> | for** <conditions>: <set of affirmations> | <function>
<variable attribution> → <operand> {<operation>} = <attribution>
<act description> → **STORY** <identifier>**:** | **STAGE <**identifier**>:** | **SECTION** <identifier>**:**
<flow description> → **FROM** <string> **TO** <string> {<flow description>}| **FROM** <string> **TO** <string> **ACTION:** <string> {<flow description>}
<conditions> <operand> <condition> <operand>}{**&&** <conditions> | **||** <conditions>}
<operand> → <variable> | **"**<string>**"** | <nenule digits>(<digits>)* |<function> | <composed operand>
<composed operand> → <operand> <operation> <operand>{<operation><composed operand> | <operation><operand>}
<operation> → **+ | - | * | / | % | // | ****
<function>→ **nr_nodes(**<variable>**)** | **nr_sections(**<variable>**)** | **nr_stages(**<variable>**)** | **nr_of_interactions(**<variable>**)** | **color**(<variable>) | **width**(<variable>) | **style**(<variable>)
<condition> → **> | < | >= | <= | == |** != | **in | not**
<string> → <characters>*
<characters> →0 | 1 | 2 | … | 9 | a | b | c | … | z | A | B | C | … | Z | - | _ | , | ; | " | " | : | . | ' | ? | !
<comments> → // <string>
}

3 Semantic and lexicon

Program is working based on frames, frame in frame, so fulfilling "STORY", "STAGE" and "SECTION" field is mandatory in case they are used. It is important to mention that nodes can be connected to the nodes from another section, not only with the nodes in the samesection, also provides different connection within nodes.

The presented DSL is case sensitive meaning that the keyword "STAGE" is different from the instance "stage". The keywords that are related to the flow of the story and mark either a part of it, or the interaction between nodes, are uppercase other being lowercase. Each keyword should be separated by other words with space token that is nor keyword, not identifier, as "(", ")", "[", "]", etc. Any instance that is not separated be a white space or is not between quotation marks, are considered as tokens.

There are two basic data types in the developed DSL: integer (denoted by "int") and string (denoted by "string"). Integer range is considered between -214783648 and 214783647. A <string> is considered any instance consisting of <characters> that could be any printable ASCII character described in the grammar.

With statement "STORY" the story is marked. "STAGE" is a smaller part than "STORY", "SECTION" is the smallest mean of dividing a telling. Nodes has a simple construction, the description introduced by "ACTION" being optional. Also, it is important to mention the functions can be called only to variables that they describes in other words nr_sections(<division>), called for larger divisions, as well as nr_stages(<division>) only for story division.

Commands are executed from top to bottom, one after another, similar to the scripting languages.

4 Parse tree

Parse tree is a hierarchical structure, which represents the derivation of the grammar to yield input strings. In fact, it is an order-rooted tree that represents the syntactic structure of a string according to some context-free grammar [6]. In Fig. 1, it is presented the parse tree, in accordance to the described grammar, of the code written below.

```
SECTION section1:
    FROM node1 TO node2
nr_nodes(section1)
```

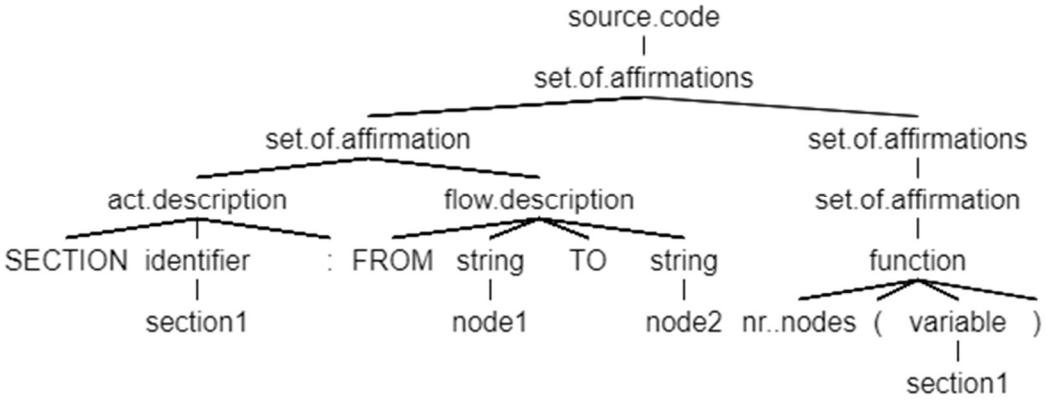

Figure 1. Parse tree of an example of sequence of code written in the described DSL

Conclusion

Despite showing a large applicability of the DSL, the main reason of this article is to present a method to ease the process of writing interactive storytelling by managing the flow using a domain specific language. The DSL is designed to make writing of the interactive stories simpler by offering a graphical representation of the story flow, making the analyzing and following of the plot more convenient for the writer. Unlike other tools used in present for this purposes, it is directed especially for this purpose and have an easy and intuitive grammar.

Nonetheless, it should be mentioned the advantage of the proposed domain specific language. The tool is new into domain as it represents a comfortable DSL for interactive story creators (who are not supposed to be programmers) that increases the productivity of story plotting. The software is aimed to become an abstract language of object-oriented designs of state transformers, a nice way to overview the plot developing depending on each of player's choices and, a straightforward design of an external DSL that is supposed fulfill the story flow design that the creator should keep in mind.

References

1. HOGUET, Benjamin, *What is interactive storytelling?* [online]. 2014, 12 [visited 2.01.2022]. Available: https://bit.ly/3tslk58
2. SIMPSON, Char. Digital storytelling I – History and Theory of interactivity.In: *Columbia DSL – 2021*, 11 August
3. LIANG, Xiaoyao. Programming methods. In: *Ascend AI Processor Architecture and Programming* - 2020
4. The Designer's Notebook: Three Problems for Interactive Storytellers, Game Developer [online]. [visited 28.01.2022]. Available: https://www.gamedeveloper.com/design/the-designer-s-notebook-three-problems-for-interactive-storytellers
5. Introduction to Grammars, TuturialsPoint, [online]. [visited 3.02.2022]. Available: https://www.tutorialspoint.com/automata_theory/introduction_to_grammars.htm
6. Parse Tree, pythonds, [online]. [visited 12.02.2022]. Available: https://runestone.academy/ns/books/published/pythonds/Trees/ParseTree.html