Санкт-Петербургский политехнический университет Петра Великого
Институт прикладной математики и механики
Высшая школа прикладной математики и вычислительной физики

Грамматика и автоматы

Курсовая работа. Разработка DSL graphEz

Работу выполнили:
Д.А. Козлов
Т. В. Алпатова
Группа:
3630102/70201
Преподаватель:
Ф. А. Новиков

Санкт-Петербург
2021

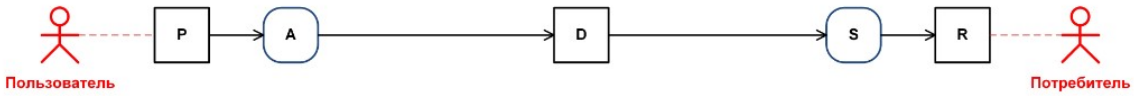
Содержание

1. Назначение и область применения

В рамках курсовой реализовать язык **graphEz** (graph easy) для удобной визуализации взвешенных и невзвешенных графов, орграфов и корневых деревьев.

1.1. Модель использования и архитектура

Язык предметной области уместно называть реализованным, если хотя бы некоторые программы P на языке возможно выполнить и получить результат R. Так получается типовая схема работы основного варианта использования реализованного языка: анализатор A разбирает программу P и строит некоторое внутреннее или интерпретируемое представление D, а затем интерпретатор семантики S по внутреннему представлению вычисляет результат R.



Для данного DSL:

- Пользователь программы P – преподаватель Ф.А. Новиков
- Потребителем результата R – студенты
- Интерпретатор семантики S – пользователь сам задает **layout** (режим отрисовки графа), **extension** (расширение выходного файла).

1.2. Диаграмма использования

2. Описание языка

2.1. Алфавит

Алфавит – описание множества допустимых символов языка. В данном DSL алфавитом являются строчные и заглавные латинские символы, цифры, знаки минус, больше меньше, пробельный символ, открывающиеся и закрывающиеся фигурные и квадратные скобки:

$$A = \{a\text{–}z, A\text{–}Z, 0\text{–}9, \text{-}, >, <, \text{'\_'}, \{, \}, [,]\}$$

2.2. Лексика

Лексика – описание множества элементарных конструкций, называемых лексическими единицами (или лексемами):

$$L = \{\text{->}, \text{<-}, \text{-}, layout, dot, neato, twopi, circo, fdp, sfdp, png, jpg, jpeg, pdf, ps, svg\}$$

2.3. Синтаксис

Синтаксис – описание правил, по которым из лексем строятся предложения (или операторы) языка, а из операторов строятся программы.

Программа на вход принимает файл, в котором должно находиться описание по меньшей мере одного графа, заключенное в фигурные скобки.

Описание состоит из двух частей:

1. описание параметров рендеринга и сохранения (может быть опущено)
2. описание элементов, образующих граф

2.3.1. Параметры рендеринга и сохранения

Описание параметров рендеринга и сохранения должно быть заключено в квадратные скобки и иметь вид пары лексем, идущих друг за другом через пробельный символ. На данный момент поддерживаются следующие параметры:

- layout *<value>*
- extension *<value>*

Параметр **layout** может принимать следующие значения:

- **dot** – фильтр для построения ориентированных графов
- **neato** – фильтр для построения неориентированных графов
- **twopi** – фильтр для радиальной компоновки графов
- **circo** – фильтр для круговой компоновки графов
- **fdp** – фильтр для построения неориентированных графов
- **sfdp** – фильтр для построения больших неориентированных графов

Значения **extension** могут быть следующие: **png**, **dot**, **jpg**, **jpeg**, **pdf**, **ps**, **svg**.

В данной версии программы требуется соблюдать указанный порядок параметров, но это временное ограничение и будет исправлено при дальнейшей разработке программы.

Описанный блок может отсутствовать вовсе или быть пустым, тогда используются значения по умолчанию:

layout dot
extension png

2.3.2. Описание элементов

Описание элементов графа следует за блоком описания параметров рендеринга и сохранения и представляет из себя последовательность команд (каждая на новой строке) одного из следующих типов:

1. **u edge v**

   где u, v – вершины графа, edge – тип ребра между ними:

   - -> ориентированное ребро из u в v
   - <- ориентированное ребро из v в u
   - - ребро между u v (граф не ориентирован)

2. **u edge v weight**

   Возможность создавать взвешенные ребра, weight – вес ребра, который задается целочисленным значением.

3. **v**, где v отдельная вершина

Вершина графа – строковый литерал с использование латинского алфавита, может содержать цифры, но не начинаться с них.

2.4. Контекстные условия

Контекстные условия – описание синтаксических правил, которые невозможно или неудобно выразить средствами контекстно-свободной грамматики.

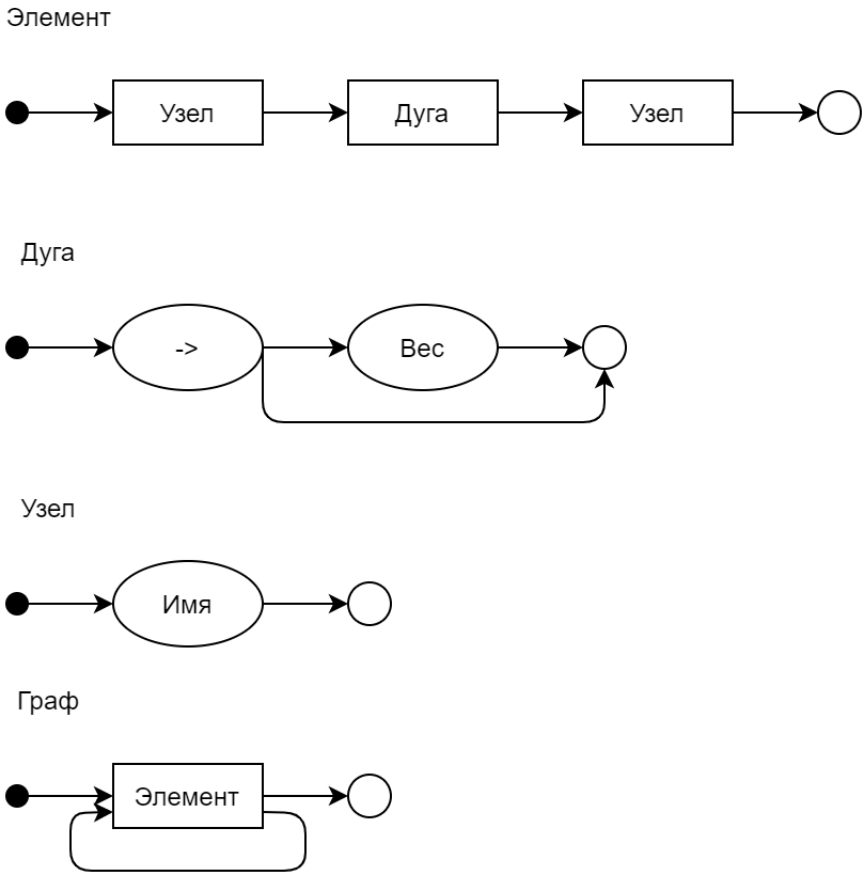
В данном DSL вводятся следующие контекстные условия:

1. все ребра графа либо ориентированы, либо неориентированны
2. на всех ребрах графа либо указаны веса, либо не указаны

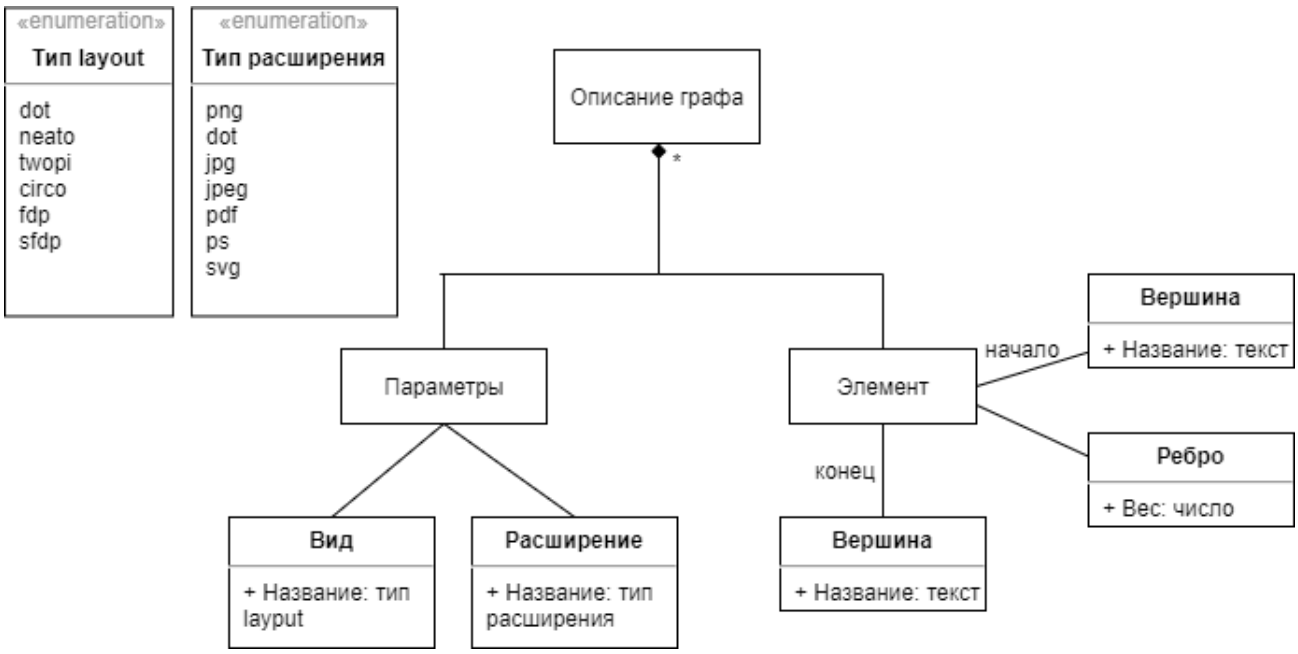
2.5. Семантика

Семантика – описание правил приписывания смысла синтаксически правильным конструкциям языка

Изобразим семантику **graphEz** в виде синтаксической диаграммы:



3. Моделирование предметной области

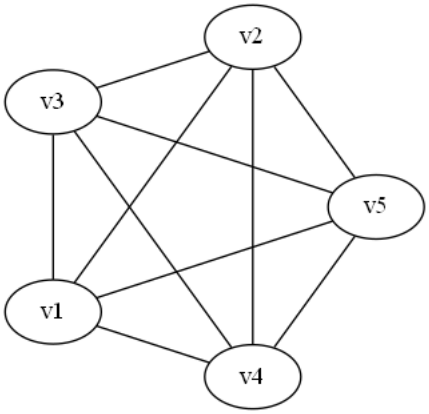


4. Программа и методика испытаний

4.1. Анализ осуществимости

Программа реализовывается на языке Python с использованием библиотеки graphviz. С помощью программы построим граф K-5 и двоичное дерево.

4.1.1. Пример 1



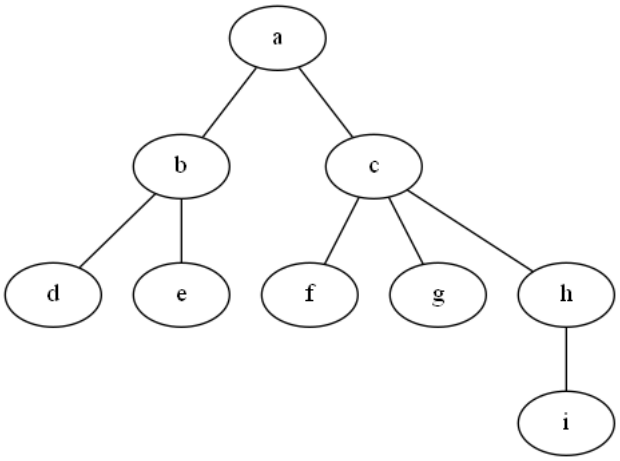
Код на языке DSL:

```
{
    [
        layout circo
        extension png
    ]

    v1 - v2
    v1 - v3
    v1 - v4
    v1 - v5
    v2 - v3
    v2 - v4
    v2 - v5
    v3 - v4
    v3 - v5
    v4 - v5
}
```

Соответсвующий код на языке Python:

```
G = pgv.AGraph()
G.add_edge('v1', 'v2')
G.add_edge('v1', 'v3')
G.add_edge('v1', 'v4')
G.add_edge('v1', 'v5')
G.add_edge('v2', 'v3')
G.add_edge('v2', 'v4')
G.add_edge('v2', 'v5')
G.add_edge('v3', 'v4')
G.add_edge('v3', 'v5')
G.add_edge('v4', 'v5')
G.draw('k5.png', prog='circo')
```

4.1.2. Пример 2



Код на языке DSL

```
{
    [
        layout dot
        extension jpg
    ]
    a - b
    a - c
    b - d
    b - e
    c - f
    c - g
    c - h
    h - i
}
```

Соответствующий код на языке Python:

```
G = pgv.AGraph()
G.add_edge('a', 'b')
G.add_edge('a', 'c')
G.add_edge('b', 'd')
G.add_edge('b', 'e')
G.add_edge('c', 'f')
G.add_edge('c', 'g')
G.add_edge('c', 'h')
G.add_edge('h', 'i')
G.draw('tree.jpg', prog='dot')
```

Из продемонстрированных примеров был сделан вывод, что написание предложенной DSL осуществимо.

4.2. Методика испытаний

Испытание программы проводилось на двух тестовых наборах:

1. Первый набор состоит из тестовых кейсов, которые заведомо нарушают установленные синтаксические правила и контекстные условия. На нем была осмотрена корректная обработка ошибок при использовании **graphEz**.

2. Второй набор представляет из себя корректно описанные графы и деревья из курса "Дискретная математика для программистов".

4.2.1. Примеры некорректный программ

- Код:

```
{
    [
        layout dot
        layout circo
    ]
    c - h
    h - i
}
```

  Результат работы:

```
line 4:8:mismatched input 'layout'
expecting {']', 'extension'}
```

  Пояснение: парметр layout должен быть задан единожды

- Код:

```
{
    layout dot
    extension png
    a - h
    h - i
}
```

  Результат работы:

```
line 2:4 mismatched input 'layout' expecting {'[', TEXT}
```

  Пояснение: параметры рендеринга должны либо отсутствовать, либо быть заключенными в квадратные скобки

- Код:

```
{
    [
        layout dot
    ]
    a - b
    b - c
    c <- a
}
```

Результат работы:

```
line 7:4 all edges must be oriented or not oriented
```

Пояснение: невыполнение контекстного условия – в одном графе присутствуют как ориентированные, так и неориентированные ребра

- Код:

```
{
    a - b 10
    b - c <7
}
```

Результат работы:

```
line 3:10 token recognition error at: '<7'
```

Пояснение: Вес ребра должен быть задан целочисленным числом.

- Код:

```
{
    [
        layout mylayout
    ]
    a - b
    b - c
}
```

Результат работы:

```
line 3:15 mismatched input 'mylayout' expecting
{'dot', 'neato', 'twopi', 'circo', 'graphs', 'fdp', 'sfdp'}
```

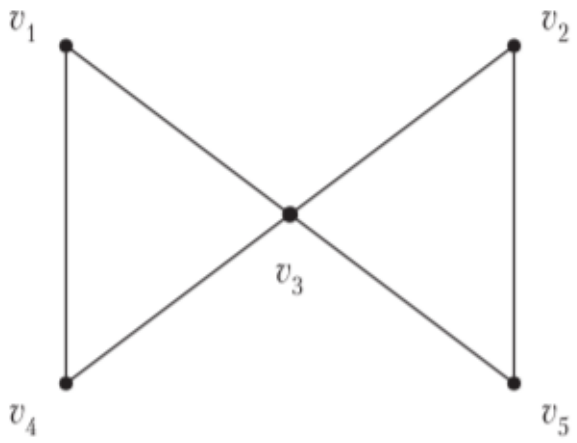
Пояснение: указано неизвестное значение параметра layout.

4.2.2. Примеры корректно построенных графов

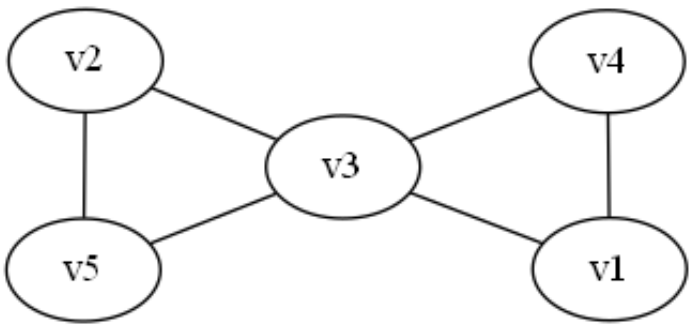
- Код:

```
{
    [
        layout sfdp
    ]
    v3 - v1
    v3 - v2
    v3 - v4
    v3 - v5
    v1 - v4
    v2 - v5
}
```

Исходное изображение (7.2.3. Маршруты, цепи, циклы (3/3)):



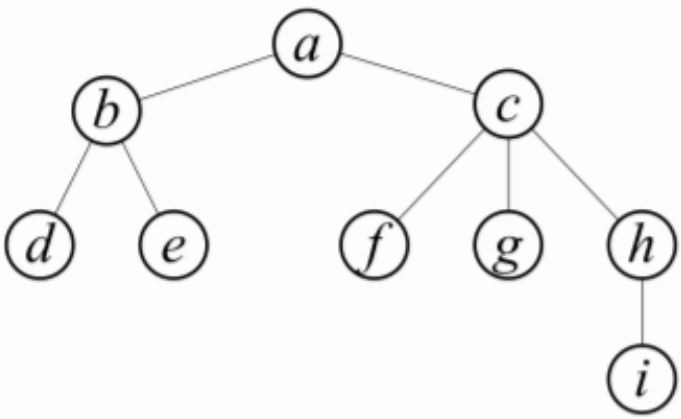
Полученное изображение:



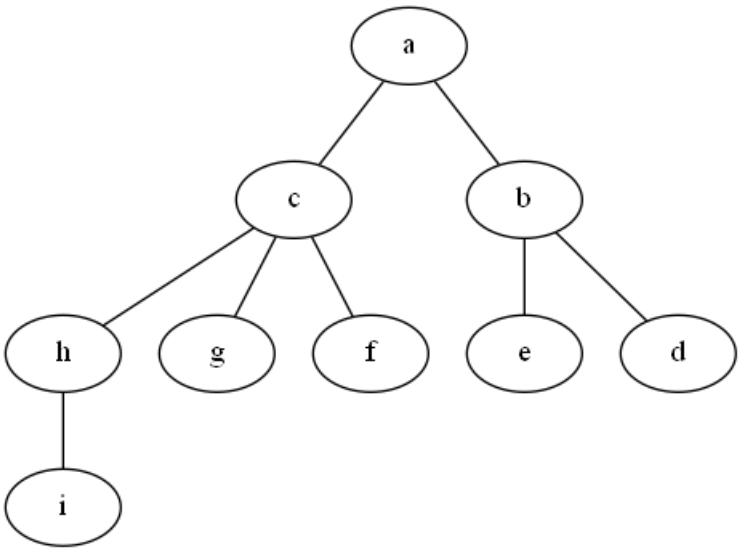
- Код:

```
{
    [
        layout dot
    ]
    a - b
    a - c
    b - d
    b - e
    c - f
    c - g
    c - h
    h - i
}
```

Исходное изображение (9.2.3. Упорядоченные деревья (5/5)):



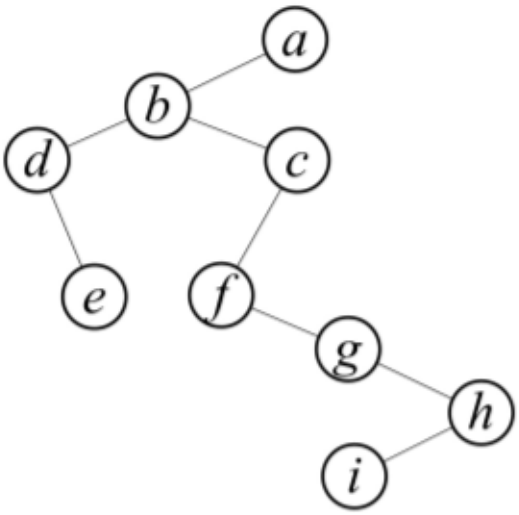
Полученное изображение:



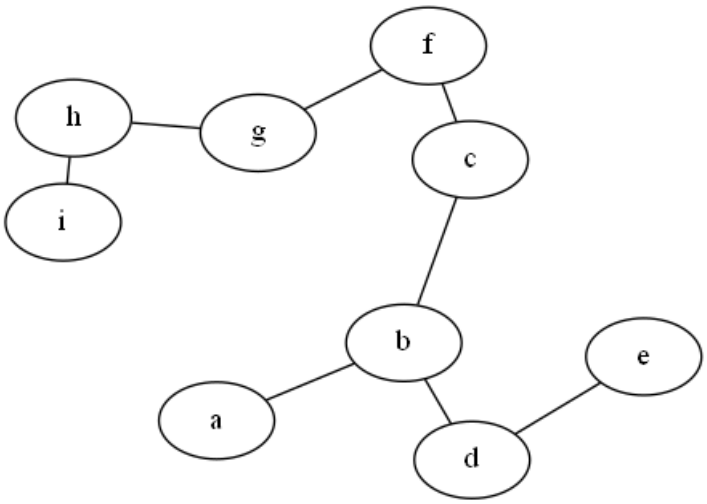
- Код:

```
{
    [
        layout fdp
    ]
    b - a
    b - c
    b - d
    d - e
    c - f
    f - g
    g - h
    h - i
}
```

Исходное изображение (9.3.5. Представление бинарных деревьев (5/5)):



Полученное изображение:

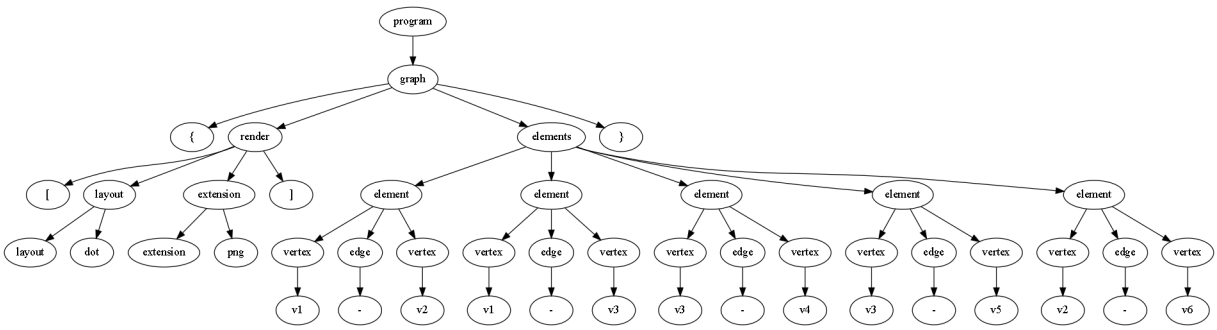


5. Код

5.1. Описание грамматики для ANTLR4

```
program: graph+ ;
graph:
        '{'
           render?
           elements
        '}';

render: '['
            layout?
            extension?
        ']';

layout: 'layout' ('dot' | 'neato' | 'twopi' | 'circo' | 'fdp' | 'sfdp') ;

extension: 'extension' ('png' | 'dot' | 'jpg' | 'jpeg' |
            'pdf' | 'ps' | 'svg') ;

elements: element+ ;

element:   (    (vertex) |
                (vertex edge vertex weight?)
           ) ;

vertex: TEXT ;
TEXT: [a-zA-Z][.a-zA-Z_0-9]* ;

weight: INTEGER ;

INTEGER: [0-9]+ ;

edge: ('->' | '<-' | '-') ;

SPACE: [ \t\r\n] -> skip;
```

Корректность грамматики может быть проверена путем автоматизированного составления дерева разбора:

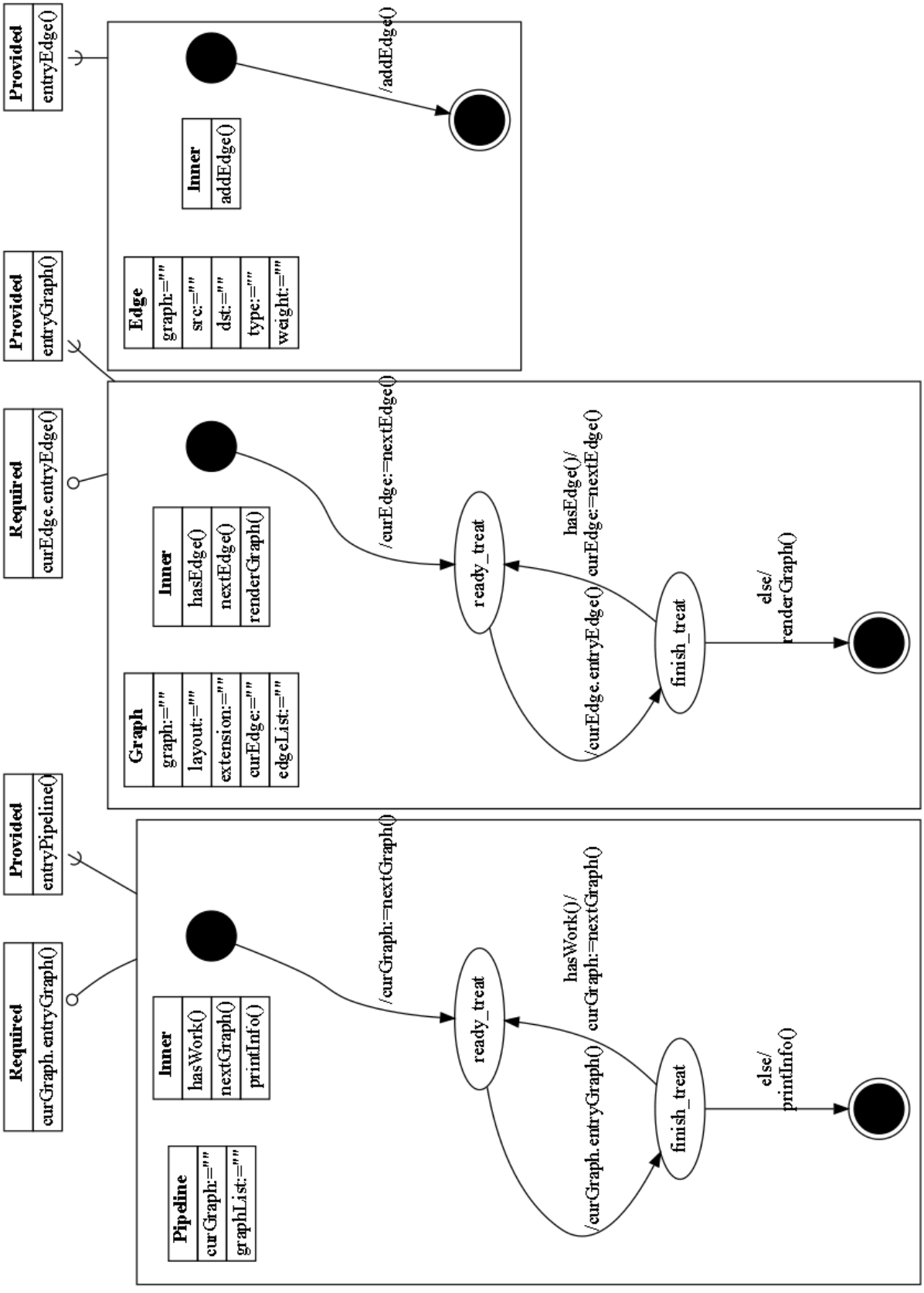


5.2. Описание автоматных объектов при помощи CIAO

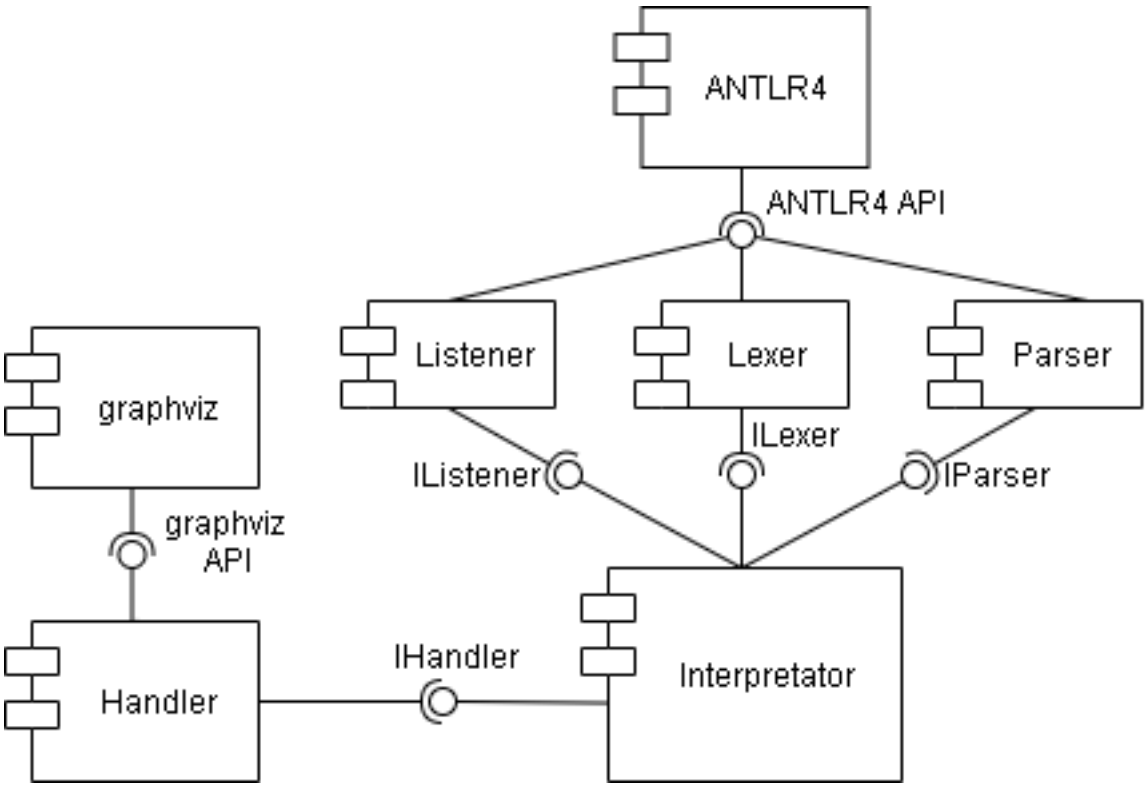
```
Pipeline
VAR
  curGraph := ""
  graphList := ""
REQUIRED
    curGraph.entryGraph()
PROVIDED
    entryPipeline()
INNER
    hasWork()
    nextGraph()
    printInfo()
STATE
    entry -> / curGraph := nextGraph() -> ready_treat
    ready_treat -> / curGraph.entryGraph() -> finish_treat
    finish_treat -> hasWork() / curGraph := nextGraph() -> ready_treat
    finish_treat -> else / printInfo() -> exit

Graph
VAR
    graph := ""
    layout := ""
    extension := ""
    curEdge := ""
    edgeList := ""
REQUIRED
    curEdge.entryEdge()
PROVIDED
    entryGraph()
INNER
    hasEdge()
    nextEdge()
    renderGraph()
STATE
    entry -> / curEdge := nextEdge() -> ready_treat
    ready_treat -> / curEdge.entryEdge() -> finish_treat
    finish_treat -> hasEdge() / curEdge := nextEdge() -> ready_treat
    finish_treat -> else / renderGraph() -> exit

Edge
VAR
    graph := ""
    src := ""
    dst := ""
    type := ""
    weight := ""
PROVIDED
    entryEdge()
```

```
INNER
    addEdge()
STATE
    entry -> / addEdge() -> exit
```

Графическое представление описанных автоматных объектов:



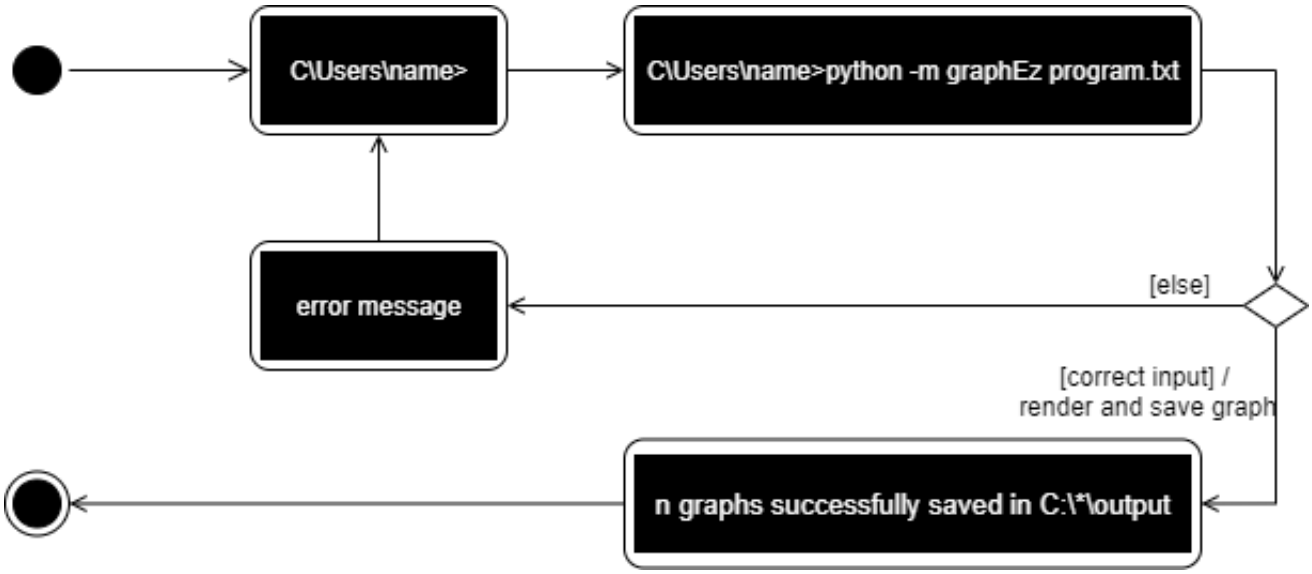
5.3. Диаграмма компонентов



5.4. Исходный код программы

https://github.com/KoDim97/DSL-graphEz

6. Моделирование поведения программы graphEz



7. Распределение обязанностей в команде

1. Татьяна Алпатова
    - Составление описания языка
    - Выбор библиотеки для отрисовки графов
    - Проведение анализа осуществимости
    - Описание грамматики для ANTLR4
    - Тестирование программы
    - Составление UML диаграмм
2. Козлов Дмитрий
    - Установка и настройка библиотеки graphviz
    - Разработка методики испытаний
    - Описания автоматных объектов при помощи CIAO
    - Дописание кода интерпретатора и методов автоматных объектов
    - Составление UML диаграмм