|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт информационных технологий | |
| Кафедра корпоративных информационных систем  **КУРСОВАЯ РАБОТА**  по дисциплине  Структура и алгоритмы обработки данных  **Тема курсовой работы**: «Создание декларативного языка для описания GUI с автоматической генерацией кода для GUI-библиотек» | |
| Студент группы ИКБО-07-18 | Басыров Сергей Амирович |
|  |  |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  |  |
| Руководитель курсовой работы | Советов Пётр Николаевич |
|  |  |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись руководителя) |
|  |  |
| Работа представлена к защите | «20» декабря 2019 г |
|  |  |
| Допущен к защите | «23» декабря 2019 г. |

# содержание

# ВВедение

Предметной областью данного курсового проекта является графические интерфейса пользователя с использованием GUI-библиотеки в рамках какого-либо высокоуровневого языка программирования.

Основные задачи, возникающие при создании GUI как правило, являются: изучение GUI-библиотеки и реализация графического интерфейса. Первая задача как правило реализуема в относительно короткие сроки, однако, вторая задача может потребовать значительных временных затрат, ввиду специфики языка программирования. Если такое сравнение уместно, то, например, реализация сложного графического интерфейса на базе Qt в С++ будет быстрее и не трудоёмкой задачей, нежели на базе Tkinter в Python.

Исходя из темы курсовой работы поставим вопрос, возможно ли создать декларативный язык для описания графического интерфейса, который одновременной решал бы и первую задачу и вторую, за короткие сроки? Чтобы время на его изучение было минимальным и при этом реализация графического интерфейса не уступала бы библиотечным аналогам.

Одним из вариантов решения является создание языка на базе существующей GUI-библиотеки. Под этим подразумевается создание предметно-ориентированного языка декларативного стиля решающий задачу source-to-source генерации. Иначе говоря, мы описываем графический интерфейс на понятном для разработчика языке, а получаем сгенерированный код под GUI-библиотеку. Для решения такой задачи необходимо:

1. проектирование языка;
2. спроектировать и разработать транслятор;
3. спроектировать и разработать генератор кода;
4. спроектировать и разработать консольное приложение, которое позволит получать сгенерированный код.

Для реализации транслятора, генератора и приложения, в рамках курсового проекта был выбран язык Python, по требованию технического задания.

В качестве GUI-библиотеки, для которой будет генерироваться код, был выбран Tkinter, так как данная библиотека является «визитной карточкой» Python для создания графических приложений и входит в стандартную библиотеку языка.

Создаваемый декларативный язык для описания графического интерфейса будет назван Lui. Название в свою очередь является сокращением предложения – «Language of user interface».

В итоге целью курсового проекта является автоматизация создания графического интерфейса с помощью декларативного языка. А непосредственно актуальность заключается в том, что разработчик, не знающий GUI-библиотеку Tkinter сможет для своего приложения без графического интерфейса реализовать графическую часть, в короткие сроки.

# основная часть

## 1 Описание декларативного подхода и существующих решений

Декларативный язык программирования – это язык, с помощью которого решение задачи происходит посредством описания решаемой задачи, а не указания конкретной последовательности действий, приводящей к выполнению задачи.

Одним из примеров декларативного языка, с помощью которого описывается графический интерфейс является QML (Qt Modeling Language). Данный язык используется в рамках Qt Quick для создания динамичных приложений. Он предоставляет разработчику следующие возможности:

* описание GUI с помощью привычных компонентов;
* создание адаптивного GUI и создание анимации;
* обработка событий;
* вставка JavaScript-кода;
* интеграция с С++ логикой.

Достоинствами языка являются:

* простота удобочитаемость кода;
* интегрируемость с JavaScript;
* интегрируемость с Qt/C++.

Недостатками языка являются:

* (don’t know)

Пример кода на QML приведён в таблице 1

Таблица – Примеры кода на QML

|  |
| --- |
| **Пример кода на QML** |
| import QtQuick 2.12  import QtQuick.Window 2.12  Window {  visible: true  width: 640  height: 480  title: qsTr("Window app")    Text {  anchors.centerIn: parent  text: "Hello World"  }  } |

При запуске вышеприведённого кода мы увидим (Рисунок 1) созданное приложение.



Рисунок – Пример приложения на QML

Приведённый пример показывает как можно быстро описать графический интерфейс и получить приложение.

Рассмотрим как работает QML. Предварительно разработчик создаёт файл и пишет в нём QML-код, после чего он его сохраняет. После этого разработчик запускает компиляцию приложения и в работу включается QQmlEngine. Который отвечает за создание всех описанных компонентов уже в виде С++ объектов, для последующей компиляции и запуска самого приложения. Схема работы изображена на рисунке 1.



Рисунок – Схема работы QML

Файл QML проходит следующие этапы:

1. разбор;
2. компиляция;
3. создание скомпилированного файла.

В результате разбора строится синтаксическое дерево, которое отражает компонент, связанные свойства, связанные события.

В результате компиляции из полученного дерева, получается байт-код, который в последствии будет выполняться виртуальной машиной, для создания C++ объектов, в соответствии таблицей типов.

В результате создания С++ объектов, они записываются в скомпилированный файл.

## 2 Проектировании языка

Проектирование языка включается в себя следующие задачи:

1. описание синтаксиса;
2. описание структуры кода;
3. описание ключевых слов;
4. описание РБНФ.

## 2.1 Описание синтаксиса

Синтаксис представляет собой последовательное описание вложенных друг в друга компонентов, которые впоследствии будут отражены в графическом интерфейсе.

Описание компонентов начинается с названия компонента, с заглавной буквы. После чего описываются свойства и вложенные компоненты.

Для наглядности приведём пример кода, который выглядит следующим образом (Таблица 2).

Таблица – Пример кода на языке Lui

|  |
| --- |
| **Пример кода языка Lui** |
| Window {  title: "Window app"  width: 200  height: 200    Label {  caption: "It's a label caption"  }  } |

В приведённом коде идёт описание компонента Window, который является основным окном приложения, и указание свойств:

1. title – заголовок окна;
2. width – ширина окна;
3. height – высота окна.

После чего, внутри компонента Window, описывается компонент Label, который является надписью, и для которого указывается свойство caption, что является самим текстом надписи.

Таблица – Структура кода языка Lui

|  |
| --- |
| **Структура кода языка Lui** |
| Window {  title: "Window app"  [property: value]  Label {  caption: "It's a label caption"  }    [Component: { [property: value] }]  } |

Такая структура удобна для восприятия и не должна вызывать трудности в её понимании. Так как это понимается чуть ли не интуитивно. Вложенная структура друг за другом следующих элементов и свойства, характеризующие тот или иной компонент.

## 2.2 Описание ключевых слов

В качестве ключевых слов будут выступать имена компонентов и их свойства. Все компоненты можно разделить на четыре категории:

1. компоненты форм;
2. компоненты слоёв;
3. компоненты ввода;
4. компоненты вывода.

## 2.3 Описание РБНФ

РБНФ (Расширенные формы Бэкуса-Наура) – это форма записи грамматики контекстно-свободных языков. Для разрабатываемого языка они будут иметь вид, приведённый в таблице 4.

Таблица – РБНФ языка Lui

|  |  |
| --- | --- |
| **№** | **Синтаксическое уравнение** |
| 1 | syntax = mainComponent "{" {property} {component} "}" |
| 2 | component = componentName "{" {property} {component} "}" |
| 3 | property = propertyName":" string | number |
| 4 | componentName = string | {string} |
| 5 | propertyName = string | {string} |
| 6 | mainComponent = "Window" |
| 7 | string = char | {char} |
| 8 | number = digit | {digit} |
| 9 | char = "A" | ... | "Z" |
| 10 | digit = "0" | ... | "9" |

Учитывая, что синтаксические уравнения описывают рекурсивную структуру, то для реализации синтаксического анализатора, распознающего вышеописанные РБНФ, будет использоваться метод рекурсивного спуска.

Для реализации алгоритма методом рекурсивного спуска необходимо написать для каждого из правил свою процедуру. Приведём реализацию с помощью псевдокода (Таблица 5).

Таблица – Реализация МРС с помощью псевдокода

|  |
| --- |
| **Реализация правила syntax** |
| procedure syntax() {  token = tokens.next()  if (token != mainComponent)  error()    token = tokens.next()  if (token != OBRACE)  error()    property() // Add property to root of AST    component() // Add components to root AST    token = tokens.next()  if (token != CBRACE)  error()  return ast  } |
| **Реализация правила component** |
|  |
|  |

## 3 Изучение абстрактного синтаксического дерева

Абстрактное синтаксическое дерево (АСД) – это структура данных, используемая для представления структуры синтаксических уравнений в виде дерева по полученному языку.

АСД строится на этапе синтаксического анализа и для его построения реализуется алгоритм рекурсивного спуская для каждого из синтаксического уравнения.

По описанным синтаксическим уравнениям в разделе 2.1 построим синтаксическое дерево. Корнем такого дерева будет является один из оконных компонентов. Листьями корня будут:

* список свойств в формате словаря;
* список вложенных компонентов.

При этом каждый элемент списка компонентов будет иметь такую же структуру. То есть вершиной будет компонент, а листьями свойства и список компонентов. Пример синтаксического дерева изображён на рисунке 1.



Рисунок – Общее синтаксическое дерево

В качестве пример приведём пример кода, в котором будут вложенные компоненты (таблица 5).

Таблица – Пример кода

|  |
| --- |
| **Пример кода** |
| Window {  title: "Window app"  width: 150  height: 200    Button {  caption: "Click!"  }    Grid {  Row {  Label {  caption: "It's mine Label"  }    List {  data: lst  }  }  }  } |

По приведённому коду построим синтаксическое дерева, получаемое при синтаксическом анализе. На рисунке 2 изображено построенное дерево.

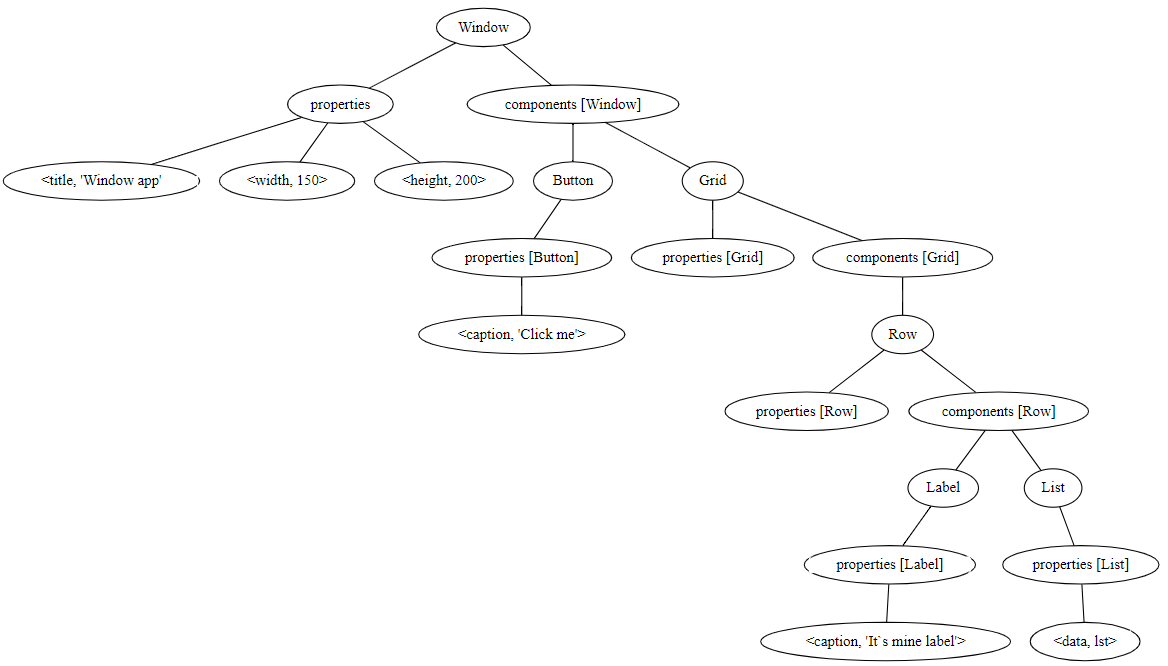


Рисунок – Построенное по коду АСД

На рисунке 3 изображён результат работы синтаксического анализатора, реализованного на Python. В окне вывода результата выведено построенное синтаксическое дерево.

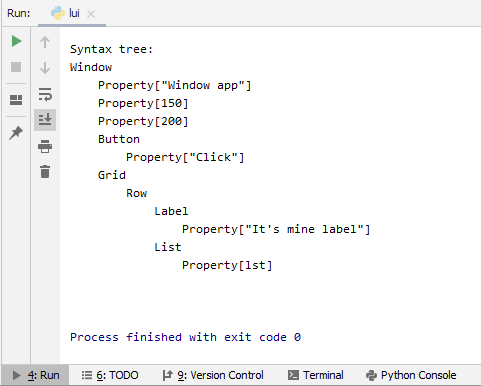


Рисунок – Результат работы синтаксического анализа

## 4 Проектирование и реализация транслятора

## 5 Изучение GUI-библиотеки Tkinter

## 6 Проектирование и реализация генератора

## 7 Проектирование и реализация приложения

В качестве приложения будет выступать консольное приложение, которое принимает на вход команды, а выдаёт Python файл со сгенерированным кодом. По существу, данное приложение является транслятором языка. Список принимаемых команд приведен в таблице.

|  |  |
| --- | --- |
| **Команда** | **Описание** |
| --help | Выводит справку по командам |
| --file=[filename] | Путь до файла \*.lui файла |
| --debug | Включает режим отладки, записывая все шаги в файл |
| --version | Выводит текущую версию приложения |

Схема работы с приложением представляется следующим образом (Рисунок …):

1. разработчик создаёт файл и пишет код на Lui;
2. передаёт файл на вход консольного приложения;
3. если во время трансляции ошибок нет, то получает Python файл;
4. запускает Python файл;
5. иначе ему выводится сообщение об ошибке.

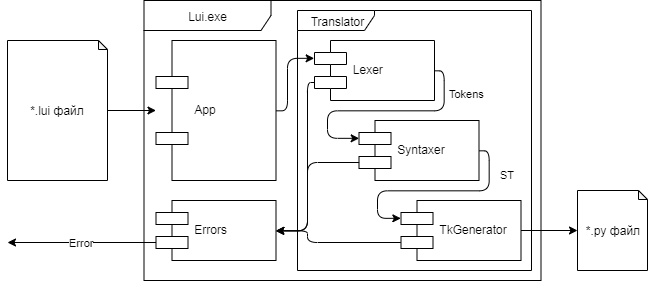


Рисунок – Схема работы консольного приложения Lui

# заключение

По итогу выполнения курсового проекта был спроектирован и разработан языка декларативного описания GUI с автоматической генерацией кода под GUI-библиотеку Tkinter.

В процесс выполнения были выполненные следующие этапы:

1. анализ декларативного подхода и существующих решений
2. проектирование декларативного языка;
3. проектирование и реализация транслятора;
4. изучение GUI-библиотеки Tkinter;
5. проектирование и реализация генератора кода;
6. проектирование и реализация консольного приложения
7. тестирование.

Как и у каждого разработанного программного продукта, имеются свои достоинства и недостатки. Выделим их.

Достоинствами являются:

* простота декларативного языка;
* наличие документации;
* открытый исходный код языка;
* кроссплатформенность.

Недостатками являются:

* поддерживает только библиотеку Tkinter;
* не является полноценной альтернативной GUI-библиотеки.

# список используемых источников

Нормативные документы:

1. ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: Стандартинформ, 2017. – 32 с.;

2. Методические указания по курсового проекта. – М.: РТУ МИРЭА, 2019. – 45 с.;

3. О введении в действие Инструкции по организации и проведению курсового проектирования. – М.: РТУ МИРЭА, Приказ №1325 от 05.10.2018. – 17 с..

Книги:

1. Построение компиляторов [Текст]: / Н. Вирт – М.: ДМК Пресс, 2016. – 272 с.;

2. Алгоритмы и структуры данных [Текст]: / Н. Вирт – М.: ДМК Пресс, 2014. – 192 с..

Электронные ресурсы:

1. Graphical widget [Электронный ресурс]. – URL: https://en.wikipedia.org/wiki/Widget\_(GUI);

2. Writing a C Compiler, Part 1 [Электронный ресурс]. – URL: https://norasandler.com/2017/11/29/Write-a-Compiler.html;

3. An Introduction to Tkinter (Work in progress) [Электронный ресурс]. – URL: http://effbot.org/tkinterbook/;

4. tkinter – Python interface to Tcl/Tk [Электронный ресурс]. – URL: https://docs.python.org/3.7/library/tkinter.html;

5. unittest – Unit testing framework [Электронный ресурс]. – URL: https://docs.python.org/3.7/library/unittest.html;

6. pydoc – Documentation generator and online help system [Электронный ресурс]. – URL: https://docs.python.org/3.7/library/pydoc.html;

7. Built-in Functions [Электронный ресурс]. – URL: https://docs.python.org/3/library/functions.html#exec;

8. QML Syntax Basics [Электронный ресурс]. – URL: https://doc.qt.io/qt-5/qtqml-syntax-basics.html;

9. QML Engine Internals, Part I [Электронный ресурс]. – URL: https://www.kdab.com/qml-engine-internals-part-1-qml-file-loading/.