Universitatea Tehnică a Moldovei

Crearea limbajului de programare de uz general folosind infrastructura pentru compilatoare LLVM

Creation of general purpose programming language using LLVM infrastructure

Student:
Alexandr Vdovicenco

Conducător:
lector univ. Mariana Catruc

Chişinău 2020

Rezumat

Acest memoriu explicativ a fost elaborat pentru planificarea, documentarea s,i implementarea limba-jului de uz general folosind infrastructura pentru compilatoare LLVM, prezentat de Alexandr Vdovicenco ca proiect de master la Universitatea Tehnicaˇ din Moldova. Acest document este scris ˆın englezaˇ s,i cont,ine

- pagini - figuri - tabele s,i - de referint,e. Proiectul ˆıs,i propune saˇ construiascaˇ o un limbaj de programare de uz general, destinat rezolvarii spectrului larg de probleme. Scopul proiectului este demonstrarea crearii si modelarea procesului de realizare a unui compilator modern folosind instrumentele oferite de catre infras-tructura LLVM.

Sistemul este compus din mai multe elemente informat,ionale fiecare responsabil de o parte dedicata, care ˆımpreuna permit tranformarea codului sursa in forma de text in comenzi binare executate de catre procesor. Partˇ,ile componente ale sistemului sunt reprezentate prin mai multe servicii dedicate care ˆımpreuna formeazaˇ partea de ”frontend” al sistemului, din asa servicii fac parte serviciul de divizarea a codului sursa in lexeme, serviciul de parsare si serviciul de creare a arborelui abstract de sintaxa. La fel partˇ,ile componente ale sistemului sunt s,i aplicat,iile dedicate din complexul de programe ce constituie infrastructura LLVM si sunt responsabile de transformarea reprezentarii intermediare in codul binar executat de catre procesor.

Teza este compusaˇ din 3 capitole care au fost scrise ˆın timpul dezvoltariiˇ proiectului s,i care sunt:

Domain analysis: ˆın acest capitol a fost analizat proiectul din punct de vedere al afacerii. Sau evident,iat aspectele generale ale sistemului pe care se vor baza ˆın continuare toate cerint,ele s,i documentat,ia s,i sa efectuat, de asemenea, o cercetare de piat,aˇ care a detectat caracteristicile cheie ale sistemului care il diferent,iazaˇ de celelalte;

System design: acest capitol ˆıs,i propune saˇ descrie sistemul din punct de vedere arhitectural. Capi-tolul este plin de diagrame UML care ajutaˇ la construirea s,i modelarea sistemului, precum s,i la gasireaˇ s,i reducerea riscurilor ˆınainte de a ˆıncepe implementarea realaˇ. De asemenea, ˆın acest capitol este de-scris procesul de lucru aplicat s,i conceptele-cheie despre BBD care a fost utilizat pe scaraˇ largaˇ ˆın timpul dezvoltariiˇ proiectului;

System implementation: acest capitol descrie ˆın detaliu procesul de implementare a sistemului. Acesta evident,iazaˇ cele mai grave probleme care au aparutˇ ˆın timpul procesului de dezvoltare s,i explicaˇ solut,iile pentru aceste probleme. Acesta exemplificaˇ diferite aspecte ale dezvoltariiˇ unui compilator modern. De asemenea, det,ine o documentat,ie despre modul de utilizare a compilatorului, care descrie foarte bine toate caracteristicile acestuia.

Abstract

This explanatory memorandum was developed for the planning, documentation and implementation of the general language using the infrastructure for LLVM compilers, presented by Alexandr Vdovicenco as a master project at the Technical University of Moldova. This document is written in English and contains - pages - figures - tables and - references. The project aims to build a general-purpose programming language for solving a wide range of problems. The aim of the project is to demonstrate the creation and modeling of the process of making a modern compiler using the tools provided by the LLVM infrastructure.

The system is composed of several informational elements each responsible for a dedicated part, which together allow the transformation of the source code in the form of text into binary commands executed by the processor. The component parts of the system are represented by several dedicated services that together form the ”frontend” part of the system, such services include the service of dividing the source code into lexemes, the parsing service and the service of creating the abstract syntax tree. Likewise, the component parts of the system are the dedicated applications from the complex of programs that constitute the LLVM infrastructure and are responsible for transforming the intermediate representation into the binary code executed by the processor.

The thesis consists of 3 chapters that were written during the evolvement of the project and which are:

Domain analysis: in this chapter was analyzed the project from the business point of view. Was highlighted general aspects of the system on which will be based all requirements and documentation further and also was done a market research that detected key features of the system that makes it different from others;

System design: this chapter aims to describe the system from the architectural point of view. The chapter is full of UML diagrams which help to build and model the system and also to find and reduce risks before starting the actual implementation. Also in this chapter is described the applied work process and key concepts about BBD that was widely used during the project evolvement.

System implementation: this chapter describes in details the process of implementation of the sys-tem. It highlights the hardest problems that appeared during the development process and explains the solutions for that problems. It exemplifies different aspects of the development of an Android ap-plication and also it presents continuous integration setups that helped a lot with testing and sharing the system between beta testers before the actual release of the app in Google Play. Also, it holds a user documentation of how to use the application which describes very well all application features.

Table of contents

INTRODUCTION

Before the development of computer or programming, people did their jobs manually. It used to take a lot of time but they had no choice. Then the computer era came, and now the jobs to be done were fed on the system. It considerably reduced the amount of time taken for the completion of the same task.

Moving further,according to the law of nature, need for evolution was felt, existing systems were then improved for time and purpose.

Simultaneously, many different categories of programming languages came into existence based on the needs of the programmer or the purpose of program development.

Some of them were found efficient for a wide range of purpose, some for specific. Hence the program-ming languages based on purpose were categorized as: General purpose and domain specific programming languages.

Some programming languages are designed specially to suit or a meet a particular need, they are called as domain specific programming languages, as they are made to meet the needs of a particular sphere.

”The programming languages which can meet the needs of individual domain are called as domain specific programming languages.”

E.g. FORTAN and APL are suitable for programming related to mathematical purpose. ML, OCAML, Haskell are appropriate for research work. Lisp is favorable for AI related work. C is believed to be suitable only for system programs. With the domain specific programming languages, we have another category of programming language, which are designed to suit a wide range of domains and are called general purpose programming languages.

”The programming languages which can fulfill the needs of a wide variety of domains are called as general purpose programming languages.”

These languages can fulfill more than one purpose, for example they can be apt for mathematical calcu-lations, research work and application development at the same time.

References

1 T. Ball, “Writing a compiler in go.” https://interpreterbook.com/, 2018. [Online; accessed 24-October-2020].

2 Wikipedia, “Compiler — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/wiki/Compiler, 2017. [Online; accessed 02-November-2020].

3 LLVM, “Getting started with llvm.” https://llvm.org/docs/GettingStarted.html# getting-started-with-llvm, 2012. [Online; accessed 27-September-2020].

4 A. NGUYEN, “Compiler design and implementation in ocaml with llvm framework.” https://polly. llvm.org/publications/grosser-diploma-thesis.pdf, 2019. [Online; accessed 12-October-2020].

5 D. A. Großlinger,¨ “Enebling optimization in llvm.” https://polly.llvm.org/publications/ grosser-diploma-thesis.pdf, 2011. [Online; accessed 15-October-2020].

6 LLVM, “First language froented.” https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/, 2013. [Online; accessed 27-September-2020].

7 R. Eklind, “Llvm ir and go.” https://blog.gopheracademy.com/advent-2018/llvm-ir-and-go, 2018. [Online; accessed 02-December-2020].

8 Wikipedia, “Static single assignment form — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/wiki/Static_single_assignment_form, 2017. [Online; accessed 025-November-2020].

9 Packt, “Introducing llvm intermediate representation.” https://hub.packtpub.com/ introducing-llvm-intermediate-representation/, 2014. [Online; accessed 27-October-2020].

10 C. Lattner, “Llvm.” https://www.aosabook.org/en/llvm.html, 2010. [Online; accessed 29-October-2020].

11 R. Nystrom, “Crafting interpreters.” http://craftinginterpreters.com/contents.html, 2015. [Online; accessed 12-October-2020].

12 T. Ball, “Writing an interpreter in go.” https://interpreterbook.com/, 2017. [Online; accessed 24-October-2020].

13 Wikipedia, “Parsing — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/wiki/Parsing#Parser, 2019. [Online; accessed 03-November-2020].

14 D. Crockford, “Top down operator precedence.” http://javascript.crockford.com/tdop/tdop.html, 2018. [Online; accessed 25-November-2020].

15 B. Nystrom, “Pratt parsers: Expression parsing made easy.” http://journal.stuffwithstuff.com/2011/03/19/pratt-parsers-expression-parsing-made-easy/, 2018. [Online; accessed 25-November-2020].

16 Community, “Library for interacting with llvm ir in pure go..” https://llir.github.io/document/, 2018. [Online; accessed 02-December-2020].

17 C. M. S. of Computer Science, “Llvm, in greater detail.” https://www.cs.cmu.edu/afs/cs/academic/class/15745-s13/public/lectures/L6-LLVM-Detail-1up.pdf, 2017. [Online; ac-cessed 16-November-2020].

18 L. Team, “Llvm static compiler.” https:/llvm.org/docs/CommandGuide/llc.html, 2012. [Online; accessed 06-December-2020].

19 M. Spencer, “Object files in llvm.” https://llvm.org/devmtg/2010-11/Spencer-ObjectFiles. pdf, 2010. [Online; accessed 06-December-2020].