Project Plan: Build Your Efficient Programming Language Quickly

Lin Han, 3035027851
Xiang Ji, 2012588999
Shixiong Zhao, 3035028659

October 4, 2015

Contents

Abstract

This project plan is for the Computer Science final year project Build your efficient programming language quickly, 2015-2016, HKU. This plan aims to give an overview for the project, which includes two parts. The first part is a case study which intends to demonstrate that, based on an efficient VM and an advanced design pattern called object algebra, it's possible to build efficient programming language quickly, and it's viable to extend this framework to a larger scale. The second part will be the development of tools which support the creation of new programming languages under our language prototyping framework. Details about methodology, project plan and milestones of projects are included.

1 Introduction

This project aims to use Oracles new research Graal Java Virtual Machine and the Truffle framework to build efficient programming language implementations quickly. Truffle provides a way to build efficient interpreters that run in the Graal JVM. Programming language group of HKU is developing a framework on top of Graal and Truffle that makes the development of such interpreters quicker by providing various ready-to-use language components. The idea is that, in order to build a new programming language, a programmer just needs to pick the desired features for the language, assemble them together, add some additional custom features and quickly get a prototype implementation.

The project is divided into two parts. The first is a case study, and the second is tools development for the framework.

1.1 Case Study of ZipPy

Zhao Shixiong and Han Lin will be responsible for the case study of ZipPy. ZipPy is a fast and lightweight Python3 implementation built using the Truffle framework. ZipPy leverages the underlying Java JIT compiler and compiles Python programs to highly optimized machine code at runtime. And object algebra is used as design pattern to rewrite grammar of ZipPy to have a better extensibility in the future. With the project going on, we will rewrite ZipPy using object algebra and test it on the top of Graal and Truffle. At the end, this project will show that its possible to have a large-scale programming language like python running on the top of our framework.

1.2 Tools Development for the Framework

JI Xiang will be responsible for developing various tools for the language prototyping framework. Those tools will include:

- A pretty printer which helps users in the process of prototyping and debugging.
- Emacs & Vim plugins which provide supports such as syntax highlighting for the new languages designed by users
- Potential future tools, such as logger and debugging supports.

2 Objectives

Efficiency and extensibility are two concerns of the developing framework.

2.1 Efficiency

The project aims to provide a framework that is used to build efficient programming language. With the high-efficiency Graal virtual machine and Truffle framework, the interpreter on top of it will perform very much close to compiler, which ensures this framework could be used into practice.

2.2 Extensibility

This project aims to provide various ready-to-use language components in the future within the framework. For ZipPy, using object algebra, both grammar and features could be rewritten, making it reusable for more features or even custom features. At the finish of this project, using the case of ZipPy, we want to show that this framework is compatible to build efficient programming language on top of it.

The tools will offer essential supports for users who wish to utilize this framework to prototype new programming languages. They will make the development process feasible.

3 Methodology

3.1 Case Study on ZipPy

For the purpose of this project, we have made iteration milestones for various parts of the project. In the given methodology, we have attempted to elaborate upon the following iteration: choose one part of ZipPy (Python) grammar rewrite this part in object algebra pattern make test programs featured specific grammar run benchmarks. All tests of ZipPyOA are on top of Graal VM and Truffle framework

3.1.1 Choose one part of ZipPy(Python) grammar

Following the principle that giving the best modularity to the final framework, the choices of grammar made during iterations are mainly modularized grammar pieces. Since Python is such a big language, the iteration will begin with basic arithmetic operation, including variables, operators. And for the rest grammar, we divide them into control flow, functions, modules, object-oriented programming. In addition, required data structures for each iteration will be implemented iteratively.

3.1.2 Rewrite grammar in object algebra pattern

Object Algebras can be used to achieve FOSD, and to implement languages in a modular and type-safe way. Object Algebras are a design pattern, which provide a generalization of abstract factories and contain a set of parameterized factory methods. The factory methods can be used instead of concrete constructors to build ASTs. This indirect way of building ASTs enables extensions in multiple dimensions. This rewrite ensures the extensibility and modularity of ZipPyOA in theory. A tool called Antlr will be used as the parser generator to easily parse the grammar.

3.1.3 Make test programs featured specific grammar

Test programs are Python programs. During each iteration, with new grammar added to ZipPyOA, we would use some test program to test whether the newly added grammar works well on top of Graal VM and Truffle.

3.1.4 Benchmarks

To test that the interpreter fulfill the efficient requirement, we will mark two time stamp variable start and end using time.time() function. At the end of each iteration, we will run benchmarks respectively using Python and ZipPyOA on top of Graal VM and Truffle. If the time is close, we may believe that the efficiency requirement is met.

Using this iteration methodology, this case study will build up an efficient interpreters for mostly python features on top of Graal VM and Truffle framework.

3.2 Tools Development for the Framework

The development of framework tools encompasses a variety of software. The pretty printer and related tools will be developed in Java in cohesion with the framework's existing code; it'll also build upon existing Java libraries. The Emacs/Vim plugins will be written in Emacs Lisp/Vimscript respectively and will be relatively more independent.

3.2.1 Pretty Printer

The mechanism for pretty printing will be based on *Java Annotation*: According to the algebraic interface annotations specified for each new language that's developed, an *annotation processor* will automatically generate a unique pretty printer for this target language, which knows how to print various language components respectively in an appropriate way.

Currently, there's already a first version of the *annotation processor* present in the framework. However, it currently has two drawbacks upon which we'll improve:

1. It's still largely primitive and only works for the simplest type of language. When the target language gets a bit more complicated, it produces some error and can't generate the printer. We'll fix the bugs and ensure it can work with all types of languages

2. The printing format is still crude, which means it's not really a "*pretty*" printer: The outputted strings are mostly an amalgamation of text and are not really pleasant on the eyes. We'll seek to either utilize currently existing `Java` pretty printing libraries or translate a pretty printing framework from `Haskell` to use for this purpose.

The expected result would be an *annotation processor* that has both functionality and aesthetic quality.

3.2.2 Emacs/Vim Plugin

In order to write codes efficiently in the newly created language, it's better for the developers to have something like an IDE(Integrated Development Environment). However, developing an IDE from scratch is no doubt a huge task. An alternative would be to write plugins in Emacs/Vim (two widely used extensible text editors) which will generate appropriate syntax highlighting and corresponding editing facilities respectively for the target languages being developed by the users, according to their algebraic interface annotations.

The plugin development for Emacs will be done in `Emacs Lisp`, while the plugin for Vim will be written in `Vimscript`.

4 Schedule and Milestones

(The following schedule is tentative and subject to potential change)

4.1 Case Study on ZipPy

ID	Task	Duration	Start	End	Comments
0	**Scoping**	2 weeks	01.09.15	15.09.15	
1	Determine scope	2 days	01.09.15	03.09.15	
2	Define preliminary	5 days	03.09.15	08.09.15	
3	Secure core resources	1 week	08.09.15	15.09.15	
4	Scoping Complete	0 day	15.09.15	15.09.15	Milestone 1
5	**Configurations**	2 weeks	15.09.15	30.09.15	
6	Go through Object Algebra	3 days	15.09.15	18.09.15	
7	Using Truffle to build programming language (Mumbler)	1 week	18.09.15	25.09.15	
8	Import ZipPyOA project and run existing benchmark	4 days	25.09.15	30.09.15	
9	Finish basic configurations	0 day	30.09.15	30.09.15	Milestone 2
10	Detailed project plan and website	4 days	30.09.15	04.10.15	Milestone 3
11	Go through python grammar	7 days	04.10.15	11.10.15	Milestone 4
12	First iteration for ZipPyOA (Arithmetic part)	2 weeks	11.10.15	25.10.15	
13	Bug fixed for first iteration	3 days	25.10.15	28.10.15	Milestone 5
14	Second iteration for ZipPyOA (Control flow part)	2 weeks	28.10.15	11.11.15	
15	Bug fixed for second iteration	1 week	11.11.15	18.11.15	Milestone 6
16	Third iteration for ZipPyOA(Functions and modules)	2 week	18.11.15	02.12.15	
17	Bug fixed for third iteration	5 days	02.12.15	07.12.15	Milestone 7
18	Forth iteration for ZipPyOA(Object-oriented programming)	2 weeks	07.12.15	21.12.15	
19	Bug fixed for forth iteration	1 week	21.12.15	28.12.15	Milestone 8
20	Collect and sort data produced during the previous project	1 week	28.12.15	04.01.16	
21	Prepare for presentation	1 week	04.01.16	11.01.16	
22	First presentation	5 days	11.01.16	15.01.16	Milestone 9
23	Write and run test cases for preliminary implementation	5 days	15.01.16	20.01.16	
24	Prepare implemented project and interim project report	4 days	20.01.16	24.01.16	Milestone 10
25	Go through project and better test cases into a wider range	1 month	24.01.16	24.02.16	Milestone 11
26	Test and debug the interpreter on top of Graal VM and Truffle using general Python applications and extend previous implementation	1 month	24.02.16	24.03.16	Milestone 12

27	Conclude the implementation and data, prepare for final report and presentation	3 weeks	24.03.16	17.04.16	Milestone 13
28	Final presentation	5 days	18.04.16	22.04.16	Milestone 14
29	Project exhibition	1 day	03.05.16	03.05.16	Milestone 15

4.2 Tools Development for the Framework

ID	Task	Duration	Start	End	Comments
0	**Scoping**	2 weeks	01.09.15	15.09.15	
1	Determine scope	2 days	01.09.15	03.09.15	
2	Define preliminary	5 days	03.09.15	08.09.15	
3	Secure core resources	1 week	08.09.15	15.09.15	
4	Scoping Complete	0 day	15.09.15	15.09.15	Milestone 1
5	**Configurations**	2 weeks	15.09.15	30.09.15	
6	Test existing pretty printer tool	1 week	15.09.15	09.22.15	
7	Find suitable Java pretty printing library	1 week	09.23.15	30.09.15	
8	Finish basic configurations	0 day	30.09.15	30.09.15	Milestone 2
9	Detailed project plan and website	4 days	30.09.15	04.10.15	Milestone 3
10	First iteration for pretty printer (Mumbler language)	1 week	04.10.15	10.10.15	Milestone 4
11	Bug fixes for first iteration	4 days	11.10.15	14.10.15	Milestone 5
12	Second iteration for pretty printer (general cases)	2 weeks	15.10.15	28.10.15	
13	Bug fixes for second iteration	1 week	29.10.15	04.11.15	Milestone 6
14	Third iteration (Integrate the printer generator with pretty printing libraries)	2 weeks	05.11.15	18.11.15	
15	Bug fixes for third iteration	1 week	19.11.15	25.11.15	Milestone 7
16	Plugin development for Emacs	1 month	26.11.15	25.12.15	
17	Bug fixes for Emacs plugin development	3 days	26.12.15	28.12.16	Milestone 8
18	Collect and sort data produced during the previous work, fix bugs	1 week	29.12.15	05.01.16	
19	Prepare for presentation	1 week	06.01.16	11.01.16	
20	First presentation	5 days	11.01.16	15.01.16	Milestone 9
21	Prepare interim project report	1 week	16.01.16	23.01.16	Milestone 10

22	Plugin development for Vim	1 month	24.01.16	23.02.16	
23	Bug fixes for Vim plugin development	5 days	24.02.16	28.02.16	Milestone 11
24	Fix bugs, improve other tools, or help ZipPy implementation if necessary	1 month	01.03.16	30.03.16	
25	Review the project, prepare for final presentation	2 weeks	01.04.16	17.04.16	
26	Final presentation	5 days	18.04.16	22.04.16	Milestone 12
27	Project exhibition	1 day	03.05.16	03.05.16	Milestone 13