Pie 0.3 Language Specification/RLO

Design Document

Comp 601

Jason Bell 3078931

April 2, 2017

Professor: Dr. Huntrods

I must admit that not a lot of structured design went into this RLO. This is the case with many of my written works: I spend a most of my time researching and collecting citations, and thinking about ideas. For me, the "writing process" pretty much consists of regurgitating the contents of my brain onto a document, then spending a few days cleaning it up. I don't start with an outline; instead, I write everything I can think of, then reorganize until I'm happy with it. It's probably not the kind of writing technique one would teach in school, but it's easy to do with modern word processors, and my grades show that it works fine for me.

I opted to write a preliminary language specification for the next version of Pie, as this is my graduate project, and because doing so is the logical conclusion from the first two assignments. The first assignment surveyed the history of successful programing languages, to determine what made them popular. A very important question to consider when developing Pie: how does one prevent a new language from joining countless others in the language graveyard? The second assignment surveyed current popular programming languages, and future programming needs.

With the learnings from the previous assignments, it was logical to apply them to the larger project as a preliminary language specification. Nothing in this specification is final, and everything is potentially subject to change. While the writing process was an organic one, I carefully considered everything learned in the previous assignment, and looked for ways to adopt those ideas into the language specification.

As a language specification, this RLO is geared towards people who are already familiar with programming concepts. This means I do not explain fundamentals, such as pointers, late binding, and memory management, in great detail: I assume that the reader is already familiar with these concepts. I also assume at last basic familiarity with C++. If I was to create a RLO geared towards teaching these fundamentals, I'd do so with an established language such as Python, C#, or Java, which would enable the learner to also make use of the vast online resources available.

Instead of an outline (since as I said, I don't write outlines ahead of time), here is a summary of the RLO:

1. Introduction
   Gives a broad overview of the features of the language, followed by language fundamentals such as data types and control structures. There's nothing too revolutionary in this section: it's mostly intended as a means of familiarizing the user with the language syntax before proceeding to later sections.
2. Object-Oriented Paradigm
   This section discusses how Pie implements the object-oriented paradigm, particularly in comparison to other languages such as C++ and Java.

3. Functional Programming Paradigm
   This section discusses how Pie implements the functional paradigm, such as pure and high order functions. This section is admittedly light, as functional programming is a very complicated and advanced topic, of which I still have much to learn.
4. Parallelization
   This section discusses abstract processors, threading, concurrency control, and distributed data. This is particularly in reference to ideas inspired by Fortran, which is still widely used for scientific and numerical computing in clouds and supercomputers.
5. Unit Testing
   This section discusses Pie's support for unit testing and test coverage analysis as language features. This is a feature inherited from previous versions of Pie, and implemented for those as a means of learning about and assessing the efficacy of test driven development.
6. Embedded C++
   This section discusses the language feature that I am currently most excited about: C++ code embedded seamlessly in Pie code. This will allow easy offloading of performance critical code to C++, use of native libraries, and Assembly. It also means that any language that can be invoked from C++ can potentially be invoked from Pie.
7. Language Interoperability
   As said in 6, Pie will potentially be able to interoperate with any language invokable from C++ through language plugins.
8. Project Management
   I very briefly discuss potential project manager settings: a project manager could enforce things like minimum test coverage and restrict to certain paradigms.