The Amanda Programming Language

J. Marrero

...To Amy, my inspiring muse in all kinds of matters.

Abstract

<insert abstract>

Contents

Introduction

In this article it is being presented the Amanda Programming Language (from now on referred just as Amanda for brevity), an integral part (as well as reference implementation of the technologies provided by) the Amanda Computing Platform. Amanda is an imperative multi-paradigm language with support for the structured, object-oriented and functional paradigms; derived partially from practical lessons gathered through the use of the most common industrial-strength languages in the wilderness (C/C++ & Java). Amanda is not designed from a theoretical point of view, and it is intended to be user in practical applications as a strong and competitive general purpose language. For being able to achieve this purpose, Amanda is implemented using a variety of optimizations provided by the Amanda Computing Platform that aim to make feasible the development of programming languages with ease.

Development

The Amanda Computing Platform

Amanda is the result of the need to provide a reference implementation for the features provided by the Amanda Computing Platform. The Amanda Computing Platform is a free software project that aims to gather a plethora of interfaces and services making possible the development of programming languages with emphasis in flexibility, scalability, portability and runtime correctness. In order to provide all these services, the Amanda Computing Platform separates the process of developing programming languages into discrete components that may be used at will by several programs, as it is implemented as a collection of C/C++ libraries. The Amanda Computing Platform provides a language-and-hardware-agnostic intermediate representation based in Static Single Assignment <reference> form that allows front-end developers to forget the details of code generation and optimization at lower levels. It also allows to generalize several optimizations that otherwise would have to be implemented by the front-end compiler developers. On the other hand, Amanda also provides a low level virtual machine with its own assembly language. This assembly language is hardware independent and the virtual machine guarantees to provide all the runtime services needed. This allows a clear separation of responsibilities and enhances the mod-

ular nature of the compiler design. This is somewhat aligned with the goals of another well known open source project, the LLVM project. Yet, while they are somewhat similar in purpose, they are different at implementation on the lower level; being LLVM a little more general. However, due to the modular nature of the Amanda project, it is possible to rewrite and reuse specific portions of the code and re-target the system to produce other kinds of output (such as native machine code or even LLVM IR). This characteristic of modularity makes the Amanda project different from the mainstream free software compiler suite: the GNU Compiler Collection <elaborate here?>.

The Amanda Programming Language

As previously stated, Amanda is the practical consequence of the developing of the Amanda Computing Platform. Amanda is designed as a general purpose multi-paradigm imperative programming language with emphasis in clean object orientation, structured programming and functional programming. Amanda's syntax is somewhat derived from that of C++ and Java, for ease of transition for programmers of both languages. However, Amanda incorporates a series of features that makes it distinct from both Java and C++. Amanda is designed to be a kind intermediate to these two languages. The following sections elaborate on this topic.

Multiple inheritance

For simplicity of implementation, the Java designers adopted the decision to left out the multiple inheritance; since it can heavily difficult object oriented design (not a problem of the multiple inheritance itself, but of mediocre programmers <reference!>). However, a number of mainstream languages actually do support multiple inheritance, namely C++, Eiffel, Perl and Python <references!>. Multiple inheritance allows a class to take functionality from other classes, but may introduce ambiguities in the treatment of inherited types. In C++, these ambiguities are resolved either by explicitly indicating from which class the functionality is deduced or declaring the inheritance as virtual. As far as heavy use (or mistreatment) of multiple inheritance is discouraged; it is permitted in Amanda. The design consideration is that, differently from C++, multiple inherited class always behave as if they are "virtual inherited". However, the concept of "interface" still exists in Amanda, and it is referred to abstract base classes that do not contain data, only constants and definitions.

Another OO concept existing in Amanda is the Mixin. A Mixin is a form of object composition referred to a class that offers functionality to be inherited from another class, but it is not designed to be autonomous. A Mixin is not an specialization or refinement mode, but rather a way to acquire functionality. A subclass may decide to inherit parts of the functionality simply by inheriting from several Mixins with multiple inheritance.

Mixins delay the process of method binding to execution time, although the attributes and parameters are defined at compile time. Mixins, not being a particular form of inheritance, are designed for code reuse. Mixins declared as such are abstract classes and cannot be instantiated.

Generics

Amanda implements generic programming via "reified" <footnote> templates. This allows for a more powerful reflective programming scheme, nearer to the C++ templates than to the Java generics. However, this characteristic is actually inherited from another mainstream programming language: C#.

Template specialization

Functional programming

Operator overloading

Constructor delegation

Destructors and memory management

Name spaces

Conclusions