COPenv: Using COP to optimize code on various environments

Bastian Kruck

Hasso-Plattner-Institut Potsdam
bastian.kruck@student.hpi.uni-potsdam.de

Abstract. In this seminar, I will develop a system that enables developers to optimize programs on the outer environment. With outer environment, I mean the virtual machine(s) a program is running on. Currently, a program is exposed to the deviations in behavior of different outer environments. COPenv enables developers to specify layers that are activated by the outer environment depending on the environments product name and version. Using those layers, developers write code to avoid or facilitate specific outer-environment behavior in a maintainable way without needing to tradeoff between platforms.

Keywords: context-oriented programming, virtual machines, environment-specific, cop, copenv, runtime environment

1 Background

A virtual machine provides hardware abstraction. Although nowadays VMs work pretty transparent, their abstraction leaks when looking at performance and bugs.

Building a performance-optimized program requires knowledge about the outer environment of the language system and its own performance characteristics. After doing application-specific performance-optimizations (such as removing unnecessary event bindings), developers start optimizing with respect to outer-language performance characteristics, such as Google Chrome JIT-ing only methods smaller than a maximum block size.

Those environment-specific optimizations may be useless in different environments or even conflict between various products leading to tradeoffs between environment systems.

A similar type of outer-environment optimizations is code written to avoid bugs in the environmental system.

2 Solution

I use COP to enable programmers speeding up their applications with the awareness of how the code gets translated into processor instructions. The language runtime will enable or disable layers depending on the hardware and other virtualization layers it is running on. The developer will avoid a bug by writing the workaround-code into a layer which is specific to the product and it's version. On systems with a newer version where the bug is fixed, the specified version doesn't match and the layer stays turned off. Once all environments are fixed, the workaround can easily be removed as it is isolated to that layer.

The layers should be characterized by the name and version of the products being involved. It needs to be determined how to describe those layers. A bug might be present in multiple versions or even in all versions of a product (see Internet Explorer box model bug [1]). The version description syntax of ruby gems could be an inspiration to that[2]. Presumably, a central position of mapping outer-environment names and versions to a fix layer will facilitate the decentral implementation of layers without duplicating the product version and characteristics.

In this seminar, I will (a) investigate a mechanism to describe the activation criteria of environment-optimizing layers and (b) implement an environment-optimization that illustrates the use of such layers.

3 Impact

With COPenv, developers can write environment-optimized programs in a maintainable way without needing to tradeoff between platforms.

[1] Source: http://en.wikipedia.org/wiki/Internet_Explorer_box_model_bug, retrieved at 28.10.2014

[2] Source: http://guides.rubygems.org/patterns/#declaring-dependencies, retrieved at 28.10.2014