Review #1

VMKit: a Substrate for Managed Runtime Environments

N. Geoffray, G. Thomas, J. Lawall, G. Muller, B. Folliot

ACM/USENIX International Conference On Virtual Execution Environments, 2010

Jean-Pierre Lozi

April 30, 2010

1 Problem

Experimenting (implementation of new languages or features) with Managed Runtime Environments (MREs) for research purposes is difficult because of their complexity and size (100k+ lines of code): reimplementing an MRE from scratch is usually out of the question, and modifying one can be a daunting task.

2 Solution

VMKit was developed as an attempt to factor most of the code usually found in MREs. Small and easily-modifiable high-level MREs can be developed on top of it, allowing researchers to test new languages or features without having to worry about the implementation of MRE modules that are irrelevant to their work. Since VMKit does not impose any design decisions (memory allocation, call semantics...), there should not be any limitation regarding the high-level MREs.

VMKit aims to achieve performance comparable to commonly used VMs so that performance measurements and comparisons of VMKit-based projects will be meaningful.

VMKit's three main modules are a Just In Time (JIT) Compiler, a Memory Manager and a Thread Manager. Implementing these three modules by hand would have been infeasible given the resources allocated to the project. Therefore, VMKit is based on the LLVM project (JIT Compiler), the MMTk project (Memory Manager/Garbage Collector) and Posix Threads (Thread Manager). Most of VMKit's development time was dedicated to writing the glue between these pre-existing components.

VMKit uses LLVM code as its general-purpose bytecode. A high-level MRE must provide a translator from its bytecode to LLVM code as well as general functions that are needed to keep VMKit as generic as possible. Two high-level MREs have been implemented: J3, a Java MRE providing advanced features, and N3, a .NET MRE that was quickly developed but lacks optimizations.

3 Experiments

Experiments show that, on a performance level, VMKit is comparable to, albeit slower than, most other widely-used solutions (commercial or not). This is enough since VMKit only aims to be used as an experimentation tool. Moreover, authors explain that performance could be greatly improved with further optimizations.

4 Benefits

VMKit facilitates quickly implementing new languages or features on an MRE, which can be very useful for research purposes. The authors show that it is possible to develop a simple high-level MRE very quickly (N3) or an MRE with more advanced features (J3) on top of VMKit. They also show that modifying high-level MREs is relatively easy: in particular, it took them only one month to implement a Java virtual machine based on J3 that enforces the isolation of components in OSGi (I-JVM).

Since VMKit is based on two active open-source projects, LLVM and MMTk, any features and optimizations that are implemented in future versions of these projects should be directly usable in VMKit.

Given the resources allocated to the project, the performance shown is rather good.

5 Shortcomings

VMKit is slow compared to most other solutions, which prevents using it for other purposes than research/experimentation. A lot of further optimizations would be required to reach a reasonable performance level.

Moreover, VMKit's development is hindered by the fact that its two main building blocks (LLVM and MMTk) cannot be altered for compatibility reasons. This prevents some important optimizations (such as an adaptative compiler, for instance) from being implemented.

Finally, VMKit does not ultimately solve the problem of easily adding new features to a complex MRE: the code of the high-level MREs can also be complex to understand. Some features might be easier to add on a well-developed and documented standard MRE. Also, VMKit does not help with experiments requiring modifications of the lower layer (JIT Compiler, Garbage Collector...).