Klein Metacircular Virtual Machine Kit 0.1 Release Notes

Adam Spitz, Alex Ausch, and David Ungar

August 14, 2006

1. Introduction

Klein is (or will be, someday, hopefully) a virtual machine for the Self language, written entirely in Self, and a development environment for that VM. (The project is named after the Klein bottle.)

There is a *lot* of work left to be done before Klein will be a usable VM for Self. For example, garbage collection has not been implemented yet. Also, the VM runs very, very slowly - we have put almost no effort into optimizing the generated code. And so far, the VM runs only on the PowerPC architecture (though the development environment can be run on any system that runs Self). (See the “Work Left To Be Done” section for a more complete list.) This release is not intended to be used as a practical VM; our hope is that it will be of interest to people who are interested in developing metacircular virtual machines for dynamic object-oriented languages. (The Klein project has not been active at Sun since late 2005, and it was looking like Sun had no further interest in funding it, so we asked for permission to open-source it.)

Following are some brief explanatory notes. Send questions to adam.spitz@gmail.com, alex.ausch@gmail.com, or ungar@mac.com.

2. Why Build A VM This Way?

For a more thorough description of the principles behind Klein, see our OOPSLA 2005 paper, “Constructing a Metacircular Virtual Machine in an Exploratory Programming Environment”.

In constructing Klein, we have attempted to adhere to various principles, including object-orientation, metacircularity, heavy code reuse, reactiveness, and mirror-based reflection. We expect these principles to yield many benefits, including:

- A simpler, more malleable VM (because it is written almost entirely in a high-level language).
- Less replicated code (because the VM and the applications running on top of it are written in the same language and can share code, as can the VM and its development environment).
- Better performance (because the VM will be able to do optimizations across the entire code base, including the VM code itself - imagine being able to inline the hotspots of the VM right up into an application).
- Faster turnaround when making changes to the VM (because the VM will no longer be tied to C++ and its compilation system)

3. Building and Running Klein

Building the Klein debug server

- Use Apple's Xcode to open up the klein_C_code.xcodeproj file in the klein_C_code/xcode/klein_C_code directory.
- Make sure that the active target is "OSX debug server", and that the active build configuration is "Release" (we've had problems using it in Debug mode - it tends to put the Klein image at too high an address in memory, which Klein isn't smart enough to be able to handle yet).
- Click "Build".

Filing the Klein source code into Self

The Klein development and build environment runs inside Self, so you will need to:

- Download Self from http://research.sun.com/self and install it.
- Execute the Self code `'applications/klein/vmKits.self' runScript`.

Building Klein inside Self

- Start the Self desktop if it is not already started (run the code `desktop open`).
- Right-click the background and choose "Klein host" from the menu. A yellow box should come to your hand, with the status message, "down: connect failed: EINVAL". This yellow box represents the Klein debug server.
- Open a Terminal window, navigate to the self/bin/mac_osx directory, and run the kleinDebugServer executable that you built earlier. (Leave the Terminal window open.) The yellow thing's status message should change to "up on a PPC".
- Right-click the yellow box and choose "Get program." Then choose the image you wish to build. (The full Klein VM is named "selfVM".) A green box should appear; this represents the Klein image.
- Wait for the build process to complete. (This may take a while for a large image - on a 1.33 GHz G4 Powerbook, it takes about ten minutes to build the full selfVM image.) The build process is complete when the status line on the green box says "ready".

Running Klein

- Pick up the green thing and drop it on the yellow thing. A green debugger should appear.
- Click "Continue" to make the Klein process start running.

4. Work Already Done

Here's a quick list of what works in Klein today:

- The compiler can produce correct (but slow) machine code for most Self code (not including code that uses dynamic inheritance).
- The export system can produce a Klein bootstrap image containing a given list of modules. (We have used this to create several Klein images; the two most interesting ones are the full

selfVM image, which contains the Klein compiler and object system and enough of the core Self functionality to support them, and the Yoda smallImage, which is a stripped-down image that might someday be suitable for putting on some sort of small device.)
- Reflection (inside the Klein image) seems to work.
- The Klein compiler, running inside Klein, can compile simple methods. (It could probably compile more complex ones, but it runs out of memory, since there's no garbage collector yet.)
- The remote debugging environment allows source- and machine-level viewing and debugging of live Klein objects and processes.
- The incremental update system allows changes to be made to Klein objects (in either a not-yet-running bootstrap image or a live, running image) without needing to rebuild the entire Klein image.
- There is a half-written Self interpreter written in C++, called Yoda. (Some functionality seems to work, including a *very* rudimentary and unfinished garbage collector.) Yoda is not really part of the main Klein system; it was an experimental offshoot of it that we never got a chance to finish.

5. Work Left To Be Done

There is a *lot* of stuff that would need to be done before Klein could be a useful VM for the Self language:

- Garbage collection. We expect this to be trickier in Klein than in a normal VM, because of Klein's metacircularity. (For example, what if the GC runs out of memory during a collection? What if the GC moves its own code out from under itself?)
- A lot of primitives (graphics, etc.).
- Performance. We've put almost no effort at all into making Klein run efficiently; at present, it's hideously inefficient (in both time and space). The compiler produces very slow code, and we haven't even begun to implement any Self-style dynamic optimization tricks.
- We haven't implemented dependencies yet (which is fine for now because we haven't implemented recompilation either).
- Source-level single-stepping in the debugger. (There is a sometimes-useful approximation implemented now, but not proper source-level single-stepping.)
- Porting the assembler and compiler (so far there's only a PowerPC and SPARC assembler, and only a PowerPC compiler).
- Multiple processes inside Klein.
- Dynamic inheritance (though we hardly ever use it, and it may not be worth the cost).
- Floating-point numbers.
- Many other things - search for senders of `todo` to find some of them.

6. Conclusion

As you have read, this release embodies a lot of work with a lot of rough edges. We hope it works for you.