



The Da Vinci Machine Project: Many Languages, One Machine

September 24, 2008

John R. Rose, Sr. Staff Engineer

john.rose@sun.com

<http://blogs.sun.com/jrose>



Opportunities...

- VM-based systems have become normal
- CPU cycles are cheap enough for JIT, GC, RTT, ...
- many Java programmers, tools, systems
- much of the ecosystem is now open-source

Great (J)VM features

- flexible online code loading (with nice safe bytecodes)
- GC & object schema
- reflective access to classes & objects
- lots of ancillary tools (JMM, JVMTI, dtrace)
- good libraries & a nice language to write more
- optimizing JIT, object- and library-aware
- clever performance techniques:
 - > type inference, customization, profiling, deoptimization, fast/slow paths, etc., etc.

Opportunities...

Bottom line...

VMs and tools are both mature and ubiquitous

- So what shall we build now...?
 - > partial answer: more languages!
- There seem to be about 200 JVM language implems:
<http://robert-tolksdorf.de/vmlanguages.html>

Opportunities...

High level languages often require:

- very late binding (runtime linking, typing, code gen.)
- automatic storage management (GC)
- environmental queries (reflection, stack walking)
- exotic primitives (tailcall, bignums, call/cc)
- code management integrated with execution
- robust handling of incorrect inputs
- helpful runtime support libraries (REs, math, ...)
- a compiler (JIT and/or AOT) that understands it all

Problems

- VMs can do much more than C/C++,
 - > but not quite enough for emerging languages
 - > historically, the JVM was for Java only...
 - > (historically the x86 was for C and Pascal...)
- Language implementors are trying to reuse Vms
 - > Near-misses are experienced as “pain points”

So what's missing?

- Dynamic invocation
- Lightweight method objects
- Lightweight bytecode loading
- Continuations and stack introspection
- Tail calls and tail recursion
- Tuples and value-oriented types
- Immediate wrapper types
- Symbolic freedom (non-Java names)
- And always, higher performance



the Da Vinci Machine

*a multi-language renaissance
for the Java™ Virtual Machine
architecture*

[http://openjdk.java.net/
/projects/mlvm/](http://openjdk.java.net/projects/mlvm/)



A Solution from Sun

- Evolutionary adaptation of the present JVM
- Open-ended experimentation on Sun's Hotspot
 - > wild ideas are considered, but must prove useful
 - > while incubating, features are disabled by default
- Eventual convergence on standards
- Extension of the standard JVM architecture
 - > deliberate, measured, careful extension

Da Vinci Machine Mission Statement

- Prototype JVM extensions to run non-Java languages efficiently
- First-class architectural support (not hacks or side-cars)
- Complete the existing architecture with general purpose extensions
- New languages to co-exist gracefully with Java in the JVM

Dynamic invocation: A great idea

- non-Java call site in the bytecodes
- language-specific handler
 - > determines call linkage at runtime
 - > works in a reflective style
 - > installs direct (non-reflective) methods
- type-sensitive target method selection
- stateful: updated or revoked over time

Method handles

- Method Handle = lightweight reference to a method
- caller invokes without knowing method's name, etc.
- call runs at nearly the speed of Java call
- required to glue together dynamic call sites
- requires VM and/or library support for common adaptation patterns (curry, receiver check, varargs)

Anonymous classes

- Faster and more reliable loading and unloading
- Little interaction with system dict. or class loaders
 - > (“class names considered harmful”)
- Library-directed code customization
 - > via constant pool patching

Performance work

- No-brainer: Support less-static bytecode shapes
 - > Ongoing for years; see website for fixed bugs
 - > Examples: `Class.isInstance`, `Arrays.copyOfOf`
- Faster reflection
- More subtle: Faster closure-type objects
- Escape analysis (etc.) to remove auto-boxing
- Etc., etc.

Other great VM ideas

(which might need community champions)

- Interface injection (traits, mega-inheritance)
- Continuations (cf. Scheme call/cc)
- Value object (cf. Lisp fixnums)
- Tuple types, tail-calls (stack frame economy)

These are old ideas — the news is that they can fit into the JVM with modest adjustment.

Are we re-inventing the world?

- No, we are adapting classic ideas to the JVM.
 - > In some cases, exposing mature JVM internals to language implementors, for the first time.
 - > In other cases, adjusting JVM architecture to be less Java-centric.
- Language implementors know what they want
 - > (and know how to simulate it with 100x slowdown)
- VM implementors know what VMs can do
 - > (and know how to make their favorite language sing)
- Let's bring them together.

❖ Yay: This is a great time to be doing languages

❖ Some personal observations...

(Confessions of a congenital nerd: I used to prowl the college library stacks, hoping I'd find a book on Pascal or Fortran.

➤ And the day I found the Lisp 1.5 book...)

Not all parts of our field are equally wonderful.

- ❖ A word to the conscientious: Use this Power only for Good.
 - Create formalisms and assemblies that are assertion-rich.
 - Don't settle for indeterminate behavior.
 - When in doubt, build on rock, not on sand.
 - Exception: Prototype really good sand castles on sand.
 - Don't ship ga***ge.
 - Let's not be the generation of programmers that got Congress interested in policing how we do business.

- ❖ Nowadays, the computers are strong, the tools are mature, and the runtime—just runs.
- There is more freedom than ever before to re-discover how to program computers.
- ...to engineer apt and graceful formalisms to solve a horde of challenging and practical problems

- ❖ *Aside: Everybody owns many computers these days.*
 - The ideal was once the 3M machine: Now it's 300M, 30M, and 30K.
 - Tiny device has to work very hard to keep a tiny screen interesting => javafx, etc. Hard problem!
 - But, you can put a lot of runtime into 300M/30M.
 - Real programmers don't want assembler any more
 - (except for the occasional chance to show our chops)

- ❖ We've spent over a decade building a huge, multi-foundational Java economy.
 - Strong standards (thanks James, Guy, Bill, Tim, Gilad...)
 - Excellent implementations (thanks IBM, JRockit, Sun)
 - with standardized quality metrics, customer choice, and => competition
 - ten+ calendar years (about 1000 hacker-years) of hard work
 - robust, high-performing, plug-and-play, richly diversified JVM runtimes

❖ Extra benefits from managed runtimes:

- True portability, WO[RD]A
- (remember when C was the portable language?)
- Security, type safety, reliable access control
- Online, profile-driven (re-)optimization
- Scalability, only from smart, self-tuning systems
- A great substrate for all sort of computations!
- ...including computations in new languages (being created by people in this room)

- ❖ OK, are we ready for the next set of challenges!?
- one of the best: bringing the language dreamers (yo!) onto the same team with the managed runtime propeller heads (yo!)
- what's the missing bit?
- (or has it already happened?)

❖ missing bit is simple: JVM, a “VM” designed for “J” only,
needs a little retuning to become the mlvm

- The pleasant news is it really is “a little”
- ... at least to get started

- ❖ First real challenge is method handles: Methods unbound.
 - Java method calls are absolutely static in (N, T)
 - Escape hatches (reflection) are used at your peril
 - A method handle is a method which has forgotten its name
 - (Its name has been “erased” to a constant name “invoke”.)
 - But it is still just as fast: `invokevirtual MH.invoke(RS)T`
 - and it can easily be persuaded to waver on its type also
 - by means of low-level adapters (e.g., `checkcast-&-go`)
 - JVMs internally have method handles; let’s surface them
 - We’re picking up a very old tool from grandfather’s toolbox: Anonymous functions.

- ❖ Next challenge is inextricably linked: invokedynamic
 - Call sites unbound.
 - Complete app. control of call site semantics.
 - (Uses a semi-reflective bootstrap method, which installs a method handle at a call site. Nearly same as JVM linkage.)
- ❖ It's a fairly conservative change.
 - MH and invokedynamic work with NO verifier changes.
 - (No bytecode format changes. It's all in the linkage rules.)

❖ Status:

- Spec. has been hammered since last 6/2007 by the 292 EG.
- It is much simpler now, and much less Hotspot-specific.
- RI is working (barely) since code-sprint of this summer.
- Lots of finish work to do... It's open source; check it out.

Questions?

Let's talk...

John Rose

Charles Nutter

`{john.rose, charles.nutter}@sun.com`