The Squawk Java™ Virtual Machine: Java on the Bare Metal

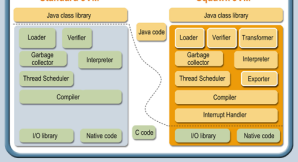
Doug Simon and Cristina Cifuentes, Sun Labs
(doug.simon@sun.com, cristina.cifuentes@sun.com)

The Squawk JVM

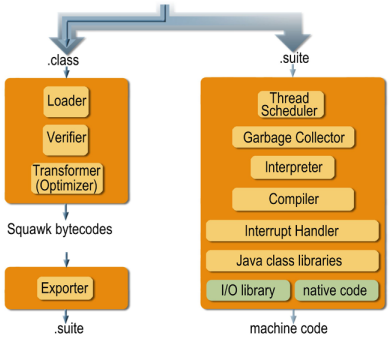
J2ME + OS Functionality

- Small JVM written in Java ⇒ Portable
- Runs on bare ARM hardware (without an OS)
- Runs on Solaris, Linux, MacOS and Windows (within standard J2SE VM)
- Designed for small devices
- Runs multiple applications (isolates) in one VM
- Allows migration of running applications (isolates) between VMs

Standard JVM vs. Squawk JVM



Squawk's Split VM Architecture



Allows Migration of Running Applications

Isolate (Application) Migration

- Move running application with its state to another machine with the same resources
- Limitations with respect to external state
- Uses:
  - > Load balancing
  - > Field replacement of hardware
  - > Simple client-server applications
  - > Local debugging of remote application

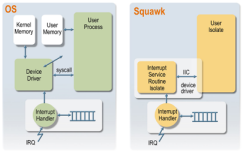
Load Balancing Example

A Platform for Cooperating Devices

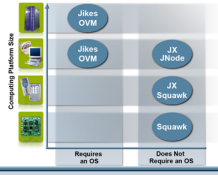
Runs Without an OS on Bare ARM

Squawk's Interrupt Handler

- Interrupt handlers in Java
  - > Support hardware interrupts
  - > Support device drivers written in Java
- Version 1 – one heap, interrupt-driven
- Version 2 – Isolate-based, two heaps, interrupt-driven
- Effects of GC on interrupt handling latency

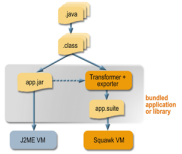


Comparison to Other Work



Optimized for Small Devices

Suites: An Execute-in-Place (Position Independent Code) File Format



Uncompressed JAR vs. Suite File Size Comparison

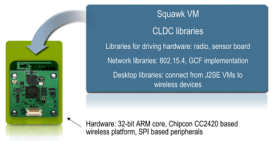
Application	JAR	Suite	Suite/JAR
CLDC	458,291	149,542	0.33
cubes	38,904	16,687	0.42
hanoi	1,805	835	0.46
delta blue	30,623	8,144	0.27
mpeg	100,917	54,888	0.54
manyballs	12,017	6,100	0.51
pong	17,993	7,567	0.42
spaceinvaders	50,854	25,953	0.51
tilepuzzle	18,516	7,438	0.40
wormgame	23,985	9,131	0.38
Total	753,905	286,285	0.38

Squawk Bytecodes vs. Java Bytecodes

Squawk Bytecode Property	Benefit
Commonly used bytecodes are 2 bytes instead of 3 bytes	More compact
References to fields and methods resolve into physical offsets	More efficient for interpretation
Local variables are typed	More efficient for compilation
One OOP map per method, nothing on the operand stack at GC points	Simplifies garbage collection

Case Study: Squawk on the Sun SPOT Platform

The Sun SPOT System



Sample Application Code

Squawk on Sun SPOT Facts

- Interpreter based (at present)
- Memory sizes:
  - > 80K RAM for VM
  - > Libraries 270K flash
- Suites
  - > 38% the size of jar'd class files
- Performance
  - > Comparable to KVM (J2ME C-based, interpreted JVM)
- Device drivers written in Java (no C)
- 802.15.4 MAC layer in Java (no C)

Experimental Results

Benchmark	.class	.suite	Sampling	(samples/sec)
Richards (Gibbons)	11,770	4,584	ARM PIO lines	11,760
Richards (Deutsch)	19,655	6,788	Sensor board input lines	300-800
Delta Blue	27,520	9,724		
Game of Life	7,390	3,396	Radio range:	90 mts

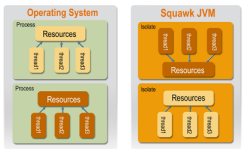
Benchmark	LOC	ms on ARM7 EB40 board
Richards (Gibbons)	410	5,277
Richards (Deutsch)	456	8,382
Delta Blue	984	4,766
Game of Life	354	4,032

Runs Multiple Applications in One VM

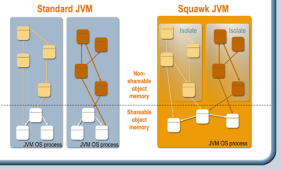
Isolate Mechanism

- An implementation of JSR 121
- Similar to operating system processes
  - > Provides isolation of applications within one VM
  - > Each isolate can have multiple threads
  - > System-level resources are shared without contention
- Allows reification of applications
  - > Can pause, resume, ask things about the application

JVM Isolates and OS Processes Analogy



A Research Implementation of JSR 121



Acknowledgments

VM
Nik Shaylor
Alex Garthwaite
John Daniels, Syntropy, UK
Dave Cleal, Syntropy, UK
Interns
Andrew Crouch
David Liu
Gary Yee
Edward Carter
Olaf Manczak
Randy Smith
Nancy Snyder
Poster Presentation
Graphic Design

Bibliography

IBM Jikes
http://jikesrvm.sourceforge.net
JX
http://www.jxos.org
OVM
http://www.ovm.org
Squawk
http://research.sun.com/projects/squawk