Java on the Bare Metal of Wireless Sensor Devices The Squawk Java Virtual Machine

Cristina Cifuentes, Dave Cleal, John Daniels, Doug Simon, Derek White

presented by Florian Landolt
University of Salzburg
Department of Computer Sciences

June 24, 2009

Outline

Basic Problems Of Modern Operating Systems

- Security Vulnerabilities
- Instability
- Error Propagation in the Kernel
- Kernel development is time consuming and maintenance-intensive

Weaknesses of C

- Type-unsafe language
- Manual memory management only (memory leaks, dangling pointers)
- No formal exception handling
- No information about an array's length
- No explicit string data type (array of characters)

therefore the resulting software is often

- Error-prone
- Vulnerable to security attacks (overflows, ...)

The Java Programming Language

Java provides some key features that ease software development

- Object-oriented Programming Language (modular programs, reusable code)
- Garbage Collection
- Pointer Safety
- Exception Handling
- A mature thread library

Advantages of Java OSs

- Stability and Security
  (type- and memory-saftey of the language)
- Code reuse and Modularization
  (inheritance and polymorphism)
- Memory Management / Memory Protection is provided by the language's runtime environment
  (No need for virtual memory)
- Reduced error propagation and improved error handling
- Simplification of kernel development and maintenance

Sun SPOT

- Sun SPOT stands for Sun Small Programmable Object Technology
- basically a wireless sensor network node
- capable of performing some processing
- gathering sensory information
- sharing gathered information with other nodes over a WPAN network

Figure: Sun SPOT device

Specifications

Main Board:

- Processor: ARM 920T (180 MHz, 32 bit)
- 512 KB of RAM
- 4 MB of flash memory
- Chipcon 2420 IEEE 802.15.4 radio chip (WPAN)
- USB interface

Additional sensor boards can be attached. The demo sensor board includes:

- 3-axis accelerometer
- light sensor
- temperature sensor
- 8 tri-color LEDs

General Information

Squawk

- meta-circular design approach
- runs on the bare metal on the ARM Architecture
- is designed for resource constrained devices
- has minimal external dependencies

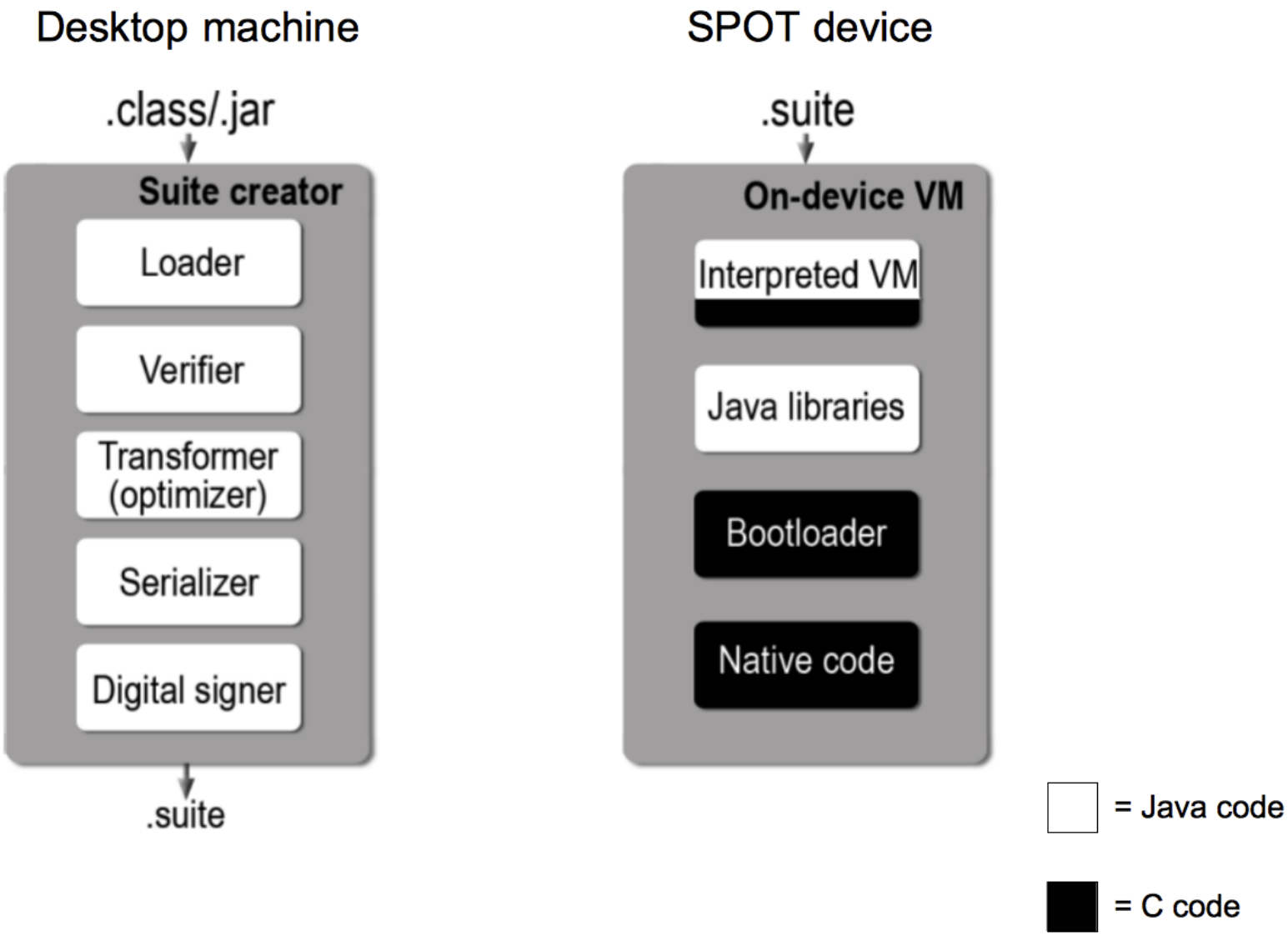
Squawk OS functionality:

- handling of interrupts
- resource management
- networking stack
- application isolation
- Java device drivers

Split VM Architecture

- The Problem:
  Resource constrained devices usually do not have enough memory to perform class file loading on the device itself.
- The Solution:
  Split the VM in two parts. Resource-intensive parts of a VM (loader, verifier, etc.) are carried out on a desktop machine.

Split VM Architecture (cont'd)



Suite Creator

- The suite creator is basically a class file pre-processor
- The output is written to a so-called suite file
- Java bytecodes are converted into a more compact form
- Creates the bootstrap suite for the on-device VM

Suite Files contain

- Squawk-internal data structures
- Squawk bytecodes (position-independent)
- internal object memory of an application

On average, suite files are 38 % the size of class files.

Application Isolation

- Each application is represented by a Java object (an instance of Class `Isolate`)
- Isolates are equivalent to the concept of processes in operating systems
- The VM can execute numerous isolates simultaneously
- Each isolate can have multiple threads

Application Isolation (cont'd)

- An isolate's resources are shared amongst its threads
- The immutable state of an isolate is shared (bytecode, string constants, parts of classes, ...),
- whereas static fields, class initialization state, class monitors are not shared

Thread Scheduling

- Isolates
  - Isolates are not considered by the scheduler.
  - The thread scheduler treats all the threads across all isolates as equal.
- Squawk VM system code
  - non-preemptible
    (for simplification of the VM design)
- Rescheduling
  - every 1000 backward branches

Thread Scheduling (cont'd)

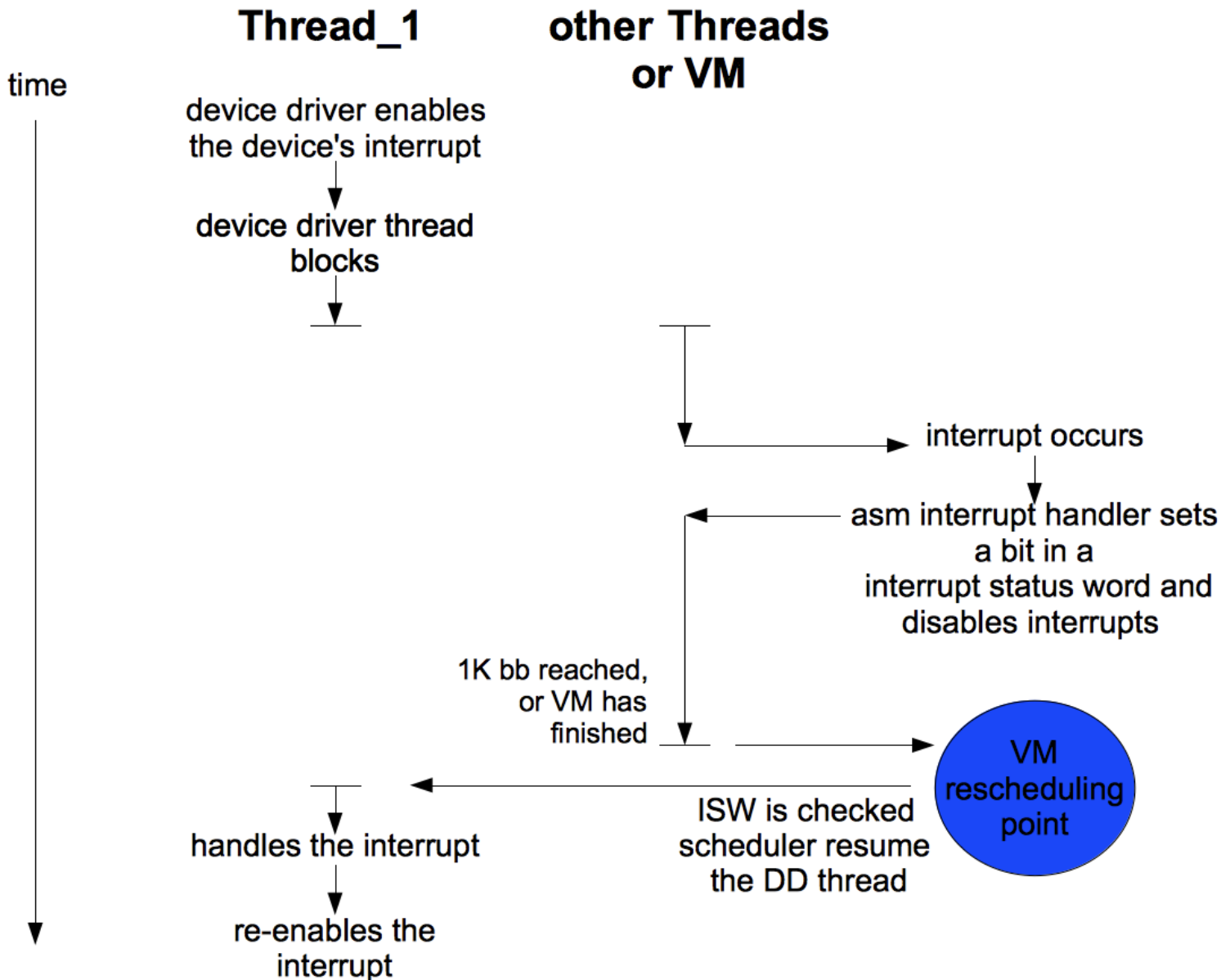
The Squawk VM implements so called green threads:

- they are managed and scheduled by a Virtual Machine
- emulate multithreaded environments and therefore do not rely on native OS functionality

Context switch:

- when control is explicitly given up by a thread (`Thread.yield()`)
- when a thread performs a blocking operation (`read()`, `sleep()`, `...`)

Interrupt Handling



Interrupt Latency

The interrupt latency is affected by the time until the next VM reschedule takes place.

Benchmarks

Best Case:	VM is idle	avg. latency of of 0.1ms
Average Case:	VM is executing bytecodes	avg. latency of 0.15 ms
Worst Case:	VM is executing a GC	latencies up to 15 ms

There are no real-time claims being made!

Rescheduling Implications of Backward Branch Counting

Definition

Backward Branches (in the context of the Squawk VM): "... a kind of bytecode instruction. Which means that the reschedule will happen more or less frequently according to the nature of the Java code being executed (tight loops will reschedule more frequently)." informal definition by Dave Cleal

- Squawk does not have a mechanism to pre-empt Java execution at the time an interrupt occurs
- The rescheduling frequency depends on the number of backward branches the currently running thread causes
- Because of this dependency the scheduling is co-operative at heart
- Although, from the thread's / programmer's point of view the system is preemptible

Test Environment

Squawk 1.1

- Sun SPOT device
- ARM920T, 180 MHz, 32-bit
- 512 KB RAM

KVM 1.1

- Sharp Zaurus
- ARMv41, 200 MHz, 32-bit
- Linux box

Results

Benchmark	Squawk (ARM920T 180 MHz) ms	KVM (ARMv4l 200 MHz) ms
Richards (gibbons)	1,296	980
Richards (gibbons_final)	1,287	948
Richards (gibbons_no_switch)	1,412	1,262
Richards (deutsch_no_acc)	1,895	2,118
Richards (deutsch_acc_virtual)	3,314	6,002
Richards (deutsch_acc_final)	3,303	3,119
Richards (deutsch_acc_interface)	3,664	4,555
DeltaBlue	792	470
Game of Life	6,699	5,848
Math int	6,764	4,077
Math long	27,282	12,813

Conclusion

- Squawk is a small JVM, mostly written in Java
- Easy to port because of minimal external dependencies
- The main focus is more on simplicity than on performance
- Performs reasonably well when compared to other interpreted JVMs
- Small, despite implementing OS-level functionality

Thank you for your attention!