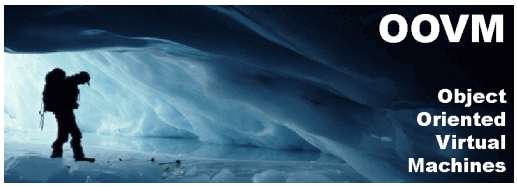
Building Robust Embedded Software

by Lars Bak, OOVM A/S



Demands of the Embedded Industy

- Increased reliability
- Low cost -> resource constraints
- Dynamic software updates in the field
- Real-time capabilities
- Rapid development cycles

Is embedded Java the solution?

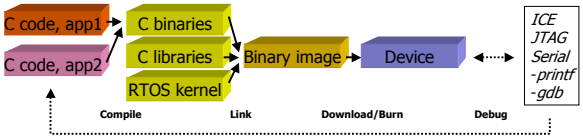
Purpose of the Presentation

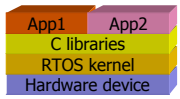
To argue for and present a new compact and fast embedded Smalltalk system that can run both hosted and on the bare metal

Why not Embedded Java?

- Does not support incremental execution
- Virtual machine specification is very complicated
- Bytecodes not designed for speed and compactness
- Configurations for embedded systems too big
  - CLDC and CDC

Embedded Software Today



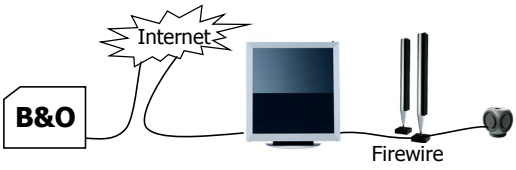


- Slow development
- Low productivity
- Unsafe programming language
- No servicability
- Very static model

What can be Better?

- Use of safe dynamic programming language
- Increase productivity
  - Connect programming environment to running system
  - Provide incremental execution
- Provide serviceablility
  - Debugging supported in production
  - On-the-fly software update

Scenario



- B&O can update software and debug/fix potential problems at customer site

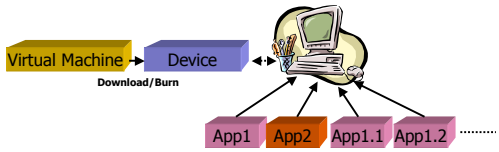
The OOVM System

- OOVM embedded platform
  - Tiny Smalltalk based virtual machine
  - Executes platform independent bytecodes
  - Fully dynamic system
  - Unified real-time resource management
  - No RTOS, no user C code!
- OOVM programming environment
  - Connects to running program
  - Supports true incremental execution
  - Full serviceability and visibility

Who am I?

- Spent the last 18 years designing and implementing object-oriented virtual machines:
  - Beta
  - Self
  - Strongtalk
  - HotSpot for Java
  - CLDC HotSpot for Java
  - OOVM

Embedded Software Using OOVM



Deployed Product

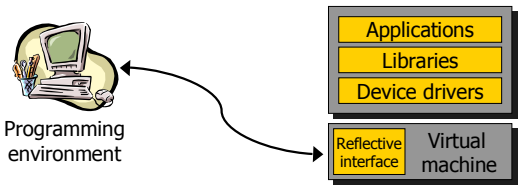
- Runs directly on hardware
- Fully dynamic system
  - Full application isolation
  - Change anything on-the-fly
- Unified resource management
- Very small memory footprint

Presentation Outline

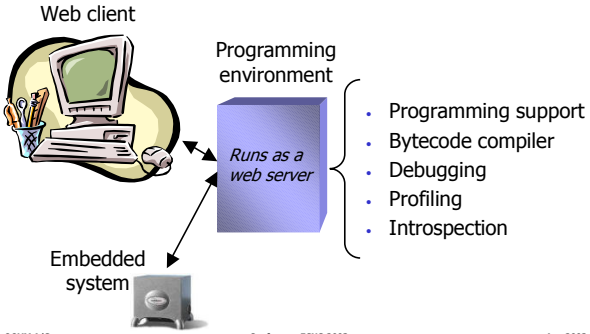
- OOVM Smalltalk in embedded systems
- A different Smalltalk system
  - Reflection vs. execution
  - Atomic test and store statement
  - Namespaces
  - Typed LIFO blocks
- Benchmarks
- OOVM demo
- Product licensing

The OOVM System

- Programming environment provides all reflective behavior
- Virtual machine provides simple reflective interface
- They use a simple reflective protocol for communication



The Programming Environment



The Programming Language

- Smalltalk with a few twists
  - Introduced
    - Syntax for full classes
    - Atomic test and store statement
    - Namespaces
    - Typed LIFO blocks
  - ... and removed
    - Pool variables
    - Class instance variables

The Virtual Machine

- Basic philosophy: keep it simple!
- 32bit virtual machine
- Scalable object heap
- Compact object model
  - 1-word object headers
- New bytecode set for Smalltalk
  - 20 bytecodes with uniform format
- Portable design

Test and Store Example

Class name

Super class

Instance variable

Method

```
Mutex = Object (
  | owner |
  "acquire the lock prior to evaluating the
   block and then release the lock"
  do: [block] = (
    [
      owner ? nil := Scheduler current
    ] whileFalse: [ Scheduler yield ].
    block value.
    owner := nil
  )
)
Example: m do: [ self update ]
```


The Virtual Machine

- Safe and fast control of
  - Memory mapped devices
  - Interrupts
- Unified automatic resource management
  - Real-time garbage collection
  - Policy based, user definable
  - Guaranteed allocation/scheduling behavior per thread/application
- Serviceability
  - True incremental program execution
  - Dynamic updating of user and system software with running apps
  - Full introspection even when running in production

Semaphore Implementation

```
Semaphore = Object (
  | count |

  acquire = (
    [ | c |
      c := count - 1.
      c < 0 ifTrue: [ ^Scheduler acquire: self ].
      count ? c + 1 := c
    ] whileFalse
  )

  release = (
    [ | c |
      c := count + 1.
      c < 1 ifTrue: [ ^Scheduler release: self ].
      count ? c - 1 := c
    ] whileFalse
  )
)
```

The Libraries

- Minimal set of classes to provide basic execution behavior
- No reflective behavior
  - Only the programming environment can create classes
  - `perform:` is not supported
- Scheduler and device drivers
- Networking libraries
  - TCP/IP (SLIP, NIC, Firewire)

Namespaces

- Desirable for modularizing code and dynamic application loading
- The namespace consists of nested classes
- Any class can be a namespace
- Examples:
  - `Services::DebuggerAgent install`
  - `::Network::Services::DebuggerAgent install`

Integer Class Hierarchy

```
- Object
  - Integer
      - SmallInteger (30 bits)
      - LargeInteger (32 bits)
```

- Writing device drivers on a 32 bit computer requires 32 bit arithmetic

Achilles Heel of Smalltalk Performance

- Allocation of block contexts
  - Inlining of basic control structures
  - Flattening of code (ex. Collection hierarchy)
- Interpretation overhead
- Slow method invocation
  - Results in breaking down code abstractions

... or apply advanced inlining compiler

Collection Class Hierarchy

```
- Object
  - Collection
      - OrderedCollection
            - IndexableCollection
                  - Interval
                  - String
                        - CompactString
                        - UnicodeString
                  - UpdatableIndexableCollection
                        - Array
                        - ByteArray
                        - ObjectArray
            - UpdatableOrderedCollection
                  - List
                  - Tree
      - UnorderedCollection
            - Dictionary
```


Typed LIFO Blocks

- Stack allocated contexts require no-escape-guarantee
- Blocks cannot be returned nor stored into heap objects
- Example from Collection

Block Block

```
collect: [collect] do: [block] = (
  self do: [ :e | block value: (collect value: e) ].
)
```

Making Bytecodes Compact

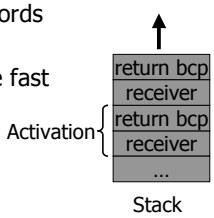
- Bytecode set designed for compactness
  - 20 simple bytecodes
- Methods with identical bytecodes are shared
  - Saves 10% of space used by methods
- Super bytecodes are computed based on static bytecode-pair-histograms
  - Reduces the bytecodes with 45%

Serviceability

- Programming environment connect to running embedded system
- Enables debugging and updating
- A set of changes can be "atomically" applied to preserve integrity of system

Compact Execution Stacks

- Stacks contain activations but are also objects
- Initial size is 512 bytes but grows as needed
- Minimal activation size is 2 words
  - No frame pointers
- Send bytecodes then become fast



Is Interpretation Fast Enough?

- OOVM interpreter will be 2x the speed of the fastest interpreted JVM
- Profiled based compilation is possible for performance critical code
- However, too much compiled code will compromise memory footprint

Bytecode Example

```
Benchmark::BenchPress::Dispatch

benchmark = (1 to: 20000 do: [ :i | self method: i ])
```

benchmark

```
1   load_constant 17
3   load_block     0
5   load_constant 21
7   load_positive_smi 1
9   send to:do:[]
11  load_local 5
13  return 4
17  block_method
21  Literal 20000
25  Literal to:do:[]
```

block_method

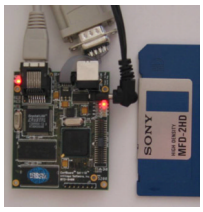
```
1   load_local 2
3   load_local 2
5   load_outer 2
7   send method: [13]
9   return 1
13  Literal method:
```


OOVM System Characteristics

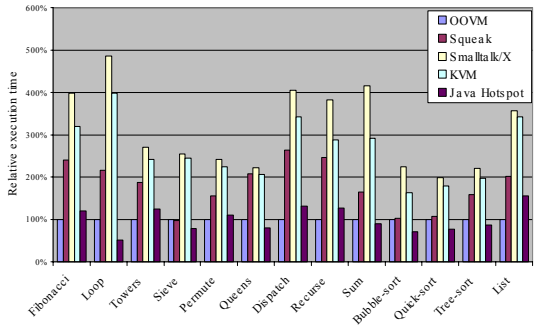
- The system runs all the time!
- Compact memory footprint
  - Minimal system executes in 128KB
  - Smaller than all OS+Java systems
- High performance
  - 2x fastest interpreted JVM
- Minimal power consumption
  - Performance is important for battery life
  - Battery size often determine the product size

Current Supported Platforms

- Embedded platforms
  - Intrinsyc Cerfcube eval board
    - 200MHz StrongARM
    - 32MB RAM, 16MB Flash
  - ICE Lynx from TI
    - 50MHz StrongARM
    - 256KB RAM, no flash
    - Firewire
    - Audio and video streaming
- Hosted platforms
  - IA32/Linux
  - StrongARM/Embedded Linux



Microbenchmark Results

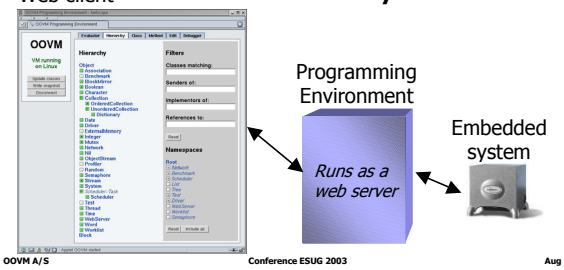


Benchmarking OOVM

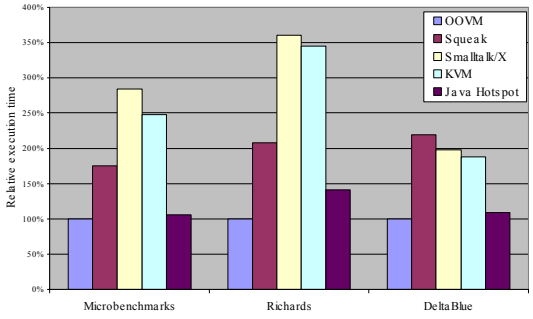
- Benchmarks
  - Microbenchmark: Stanford integer benchmarks suite
  - DeltaBlue
  - Richards
- OOVM compared to
  - Smalltalk-X version 4.1.7 (JIT disabled)
  - Squeak version 3.2-4
  - Java KVM version 1.0.4
  - Java HotSpot version 1.4.0 (JIT disabled)
- Benchmarking platform
  - Red Hat Linux 7.3
  - Intel IA-32 PIII 1133MHz

Demo of OOVM

Changing network services on-the-fly



Benchmark Results



OOVM Summary

- Next-generation platform for embedded systems
  - Order of magnitude smaller
  - Simple, fast and reliable
- Fully dynamic system
  - Dramatic improvement in development times and productivity
  - Malleable - change anything anywhere
  - Better than most desktop systems today!
- Customer benefits
  - Decrease software R&D expenses
  - Fewer system resources required
  - More reliable and serviceable products

When is it Available?

- OOVM version 1.0 scheduled for end of this year
- Dual licensing model
  - Free non-commercial use
  - Commercial use requires a license