VMKit: a Substrate for Managed Runtime Environments

Nicolas Geoffray, Gael Tomas, Julia Lawall, Gilles Muller, Beril Folliot

Dresden, 2010-05-25
Managed Runtimes

- Java, .NET

- Intermediate code representation (Java byte code / MSIL)
  - Portability
  - Language-independence
  - Safety / Security
MRE: Implementation is hard
Providing an MRE construction kit

Diagram showing the components of a memory management system, including:
- Compilation related functions: Call back
- High Level MRE: Execute Code
- GC related functions: Find global Roots, Modify Header, Trace Object, Clone Object, Find thread Roots
- Method Stub
- JIT Compiler
- Memory Manager
- Thread Manager
- Native Code
- Stack Maps
- VMKit
- System
- Input/Output
Providing an MRE construction kit
The JIT: LLVM

- LLVM as abstract language
  - No requirements on object model or call semantics
  - Efficient code generation
  - Ability to generate stack maps

- Instruction set
  - Stack-based (Java) vs. register based (LLVM)

- Supports lazy compilation
LLVM: Intrinsic functions

- **Intrinsic**: function / instruction sequence the compiler recognizes and replaces with a more efficient version
  - `printf()` → `puts()`
  - compiler-builtin functions (memcpy, strcpy, ...)
  - OpenMP

- Supported by LLVM

- Language-dependent
  - MRE must provide compilation callbacks
Providing an MRE construction kit
Memory Management Toolkit (MMTk)

• State-of-the art collection of GC algorithms

• Written in Java
  – 2-phase compilation:
    • MMTk/Java → MMTk/LLVM
    • MMTk/LLVM → MMTk/native
    • Keep LLVM representation for inlining

• MRE callbacks
  – 5 functions to support GC
    • Find root objects
    • Trace object (find sub-objects)
    • Modify object header
    • Clone object
Providing an MRE construction kit

Diagram showing the components and interactions of a system, including High Level MRE, Memory Manager, Thread Manager, and VMKit.
Threading

- Thread support layer
  - POSIX Threads

- Per-thread memory for multi-threaded GC
  - Located on thread's stack
  - Easily accessible

- Handshake lock to be polled by all threads
Java and .NET runtimes

- **J3 runtime**
  - Interfaces with GNU Classpath libraries
  - Optimized in many cases
  - Developed in parallel to VMKit

- **N3 runtime**
  - Interfaces with Mono
  - Developed on top of VMKit (~ 2 months)
  - Less optimized
Evaluation

- Implemented a Java MRE in 20,000 LoC (Sun: 6,500,000)
- VMKit: 450,000 LoC
- Memory footprint not yet optimized
- Competitive in Java / .NET benchmarks
  - Still potential for optimization though
Discussion

• Is Flash™ a managed runtime?
• L4Re (NUL) integration?