Design and Implementation of Object Oriented Virtual Machines

Lars Bak

Sun Microsystems, Inc.

Ulrik Pagh Schultz

Aarhus University

Agenda for Today

- Dynamically typed systems
- Crash course in SmallTalk™
- Simple Object Machine (SOM)
  - Simple virtual machine for SmallTalk
  - Implemented in Java
- Sunil Kothari: Deutsch, L. P., Schiffman, A. M.: *Efficient Implementation of the Smalltalk-80 System*, POPL Proceedings, 1984.

Object Oriented Languages

- Dynamically-typed (pure OO)
  - Smalltalk
  - Self
- Statically-typed (unpure OO)
  - Java
  - Beta
  - C#

Smalltalk & Java Example

SmallTalk

```
bingeEating = (
  10 timesRepeat: [ self eatLunch ]
)
```

Java

```
bingeEating() {
  for (int index; index < 10; index++) {
     eatLunch();
  }
}
```

Statically-type Languages

- Separation of objects and basic types
- Behavior of basic types is fixed and usually implemented in special byte codes
- Generic data types are likely to exclude basic types
- Believed to have fast predictable behavior

Dynamically-typed Languages

- Everything is an object
- Behavior of all objects can be changed
- Generic data types trivially includes basic types
- Easy to provide well-defined semantics for numbers (integer, double, fractions)
- Believed to have slow unpredictable performance

Discussion

- Does this make a difference at all?
- Are the claims about performance valid?
- What are some of the performance and footprint implications?
- How can integers be be represented efficiently?

Efficient Object Representation

- Trivial implementation would cause all integer operations to allocate objects
- Integers must be represented efficiently

Tagging for Dynamically-typed Languages

- All values must be same size
- Need way to distinguish integer from pointer
- 32 bit implementation
  - Use first bit for tagging
- Simplify scanning in garbage collection

Tagging for Typed Languages

- Exception for programming languages for with interface types instead of implementation types
- Programming language ex. StrongTalk

Tagging Consequences

- Prevents support for efficient 32-bit operations
- Each dispatch (message send) must do tag checking

- Is it enough to have integers represented efficiently?

The SmallTalk Language

BYTE
AUGUST 1981 Vol. 6, No. 8
$2.50 in USA/$2.95 in Canada
A McGraw-Hill Publication
the small system
SMALLTALK

The Simple Object Machine

- Virtual machine and runtime system written in Java™
- Simple object oriented virtual machine
  - Interpreted system with only 16 bytecodes
- Pure object oriented system
  - Everything is an object - even integers, booleans, and classes

Running Applications on SOM

- SOM reads classes from class path and from the system class path
- System class path defaults to the SmallTalk directory distributed with SOM

Printing

- SOM objects can be printed
  - The generic print method is implemented in the Object class and uses asString to convert the object to a string
  - The generic asString method returns strings of the form ’instance of ...’ where ... is replaced by the name of the class of the object

Bootstrapping

- The virtual machine starts by executing the initialize method on an instance of the System class
- The initialize method uses dynamic class loading to load the specified class
  - When the specified class has been loaded it is instantiated and the run method is called on the new instance

Dynamic Types

- SOM has no static type annotations as found in Java™ and C++
  - Any variable can hold a reference to any kind of object
  - Programming is much faster
- Runtime lookup errors (message not understood) can occur

Control Structures

- SOM has no built-in control structures
  - if (i < 5) { i := 16 } else { i := 2 }
  - while (i < j) { i := i + 1 }
- SOM uses blocks and virtual dispatch to implement control structures
  - (i < 5) ifTrue: [i := 16] ifFalse: [i := 2]
  - [i < j] whileTrue: [i := i + 1]

Micro-Benchmarks

- All benchmarks inherit from the abstract benchmark class
  - The abstract benchmark class shows which benchmark is running and reports timings
- SOM comes with only two benchmarks
  - Dispatch (benchmarks virtual dispatch)
  - Fibonacci

SOM FAQ

- How do I check out a SOM workspace

```
cvs -d<username>@amigo.daimi.au.dk:/users/verdich/cvsroot
   co SOMcore
```

- How do I update the workspace

```
cvs update –Pd
```

- How do I build the first time

```
SOMcore\build SOMcore    (after that just type build)
```

- How do I run a simple program

```
bin\som -cp SOMcore/Examples/Hello Hello
```

SOM Example

- Left shift in integer
- Let’s program.....

Programming Assignment

- Implement a read-eval-loop
- Enhance String behavior
  - Implement equals, length etc.
- Implement HashTable
  - Needs hashcode primitive in Object
- Implement Doubles
- Big integer implementation
  - Fixes SmallInterger overflow
- Implement proper Time class

Programming Assignments

- Make the code readable
- Code from the assignments will be integrated into the CVS root
- Make test code that verifies the correct behavior

Next Week

- Object Model
- Garbage collection
- Read (available on home page):
  - Incremental Mature Garbage Collection by Steffen Grarup & Jacob Seligmann
    - Chapter 2 & 3
  - Tenuring Policy for Generation-Based Storage Reclamation by David Ungar & Frank Jackson (who wants to present?)