Dynamic Code Evolution
for the
Java HotSpot™ Virtual Machine

Thomas Wuerthinger (wuerthinger@ssw.jku.at)
Institute for System Software
Johannes Kepler University Linz, Austria
09/16/09
My Background

**Array Bounds Check Elimination**

http://wikis.sun.com/display/HotSpotInternals/Publications+JKU

- for the HotSpot client compiler

**Ideal Graph Visualizer**

http://www.kenai.com/projects/igv

- focused on HotSpot server compiler graph
- visualization of evolving graph data structures

**Maxine**

http://www.kenai.com/projects/maxine

- meta-circular virtual machine written in Java
- recent focus: Porting the HotSpot client compiler to Java
Class Evolution

Program Execution

Redefine B, C

Suspend VM

Resume
Debugging

- Atomic changes of a set of classes
- No restrictions on the type of changes
- Update at specific point (e.g. at a break point)
- No additional indirections (e.g. that produce strange stack traces)

Long-Living Server Applications

- No performance penalty on normal execution
- Find a “good” update point
- Stability and security issues

Scripting Languages

- Fast, small, incremental changes
- Focus on additions (new methods, new fields, ...)
Levels

- Swap Method Bodies
- Change Methods
  - vtable
- Change Fields
  - instances
- Change Supertypes
  - type-safety
Redefine B, D

Find Affected Classes

Build Side Universe
Swap \((B, B')\)
Changing Active Method

```
void foo() {
    // <= REDEFINE
    bar();
    ...
}
```

Deoptimization

```
void foo() {
    // <= RESUME
    bar'();
    ...
}
```

Update Constant Pool Cache
Removing a Method

```
int foo() {
    // <= REDEFINE
    return bar();
}
```

```
int foo'() {
    return 5;
}
```

possible alternative solutions:

- Continue executing „deleted“ methods
- Replace method invocation by a constant value
- Delay code evolution until invalid old code is no longer active

throws NoSuchMethodException in foo()
Instance Updates

Future Plans: Connect with the NetBeans refactoring facilities
GC with increased object size

Side Buffer:

a

a

a

b

a

b
Removing a Field

Possible alternative solutions:
- Keep the deleted field in „old“ objects
- Replace the field access by a constant value (e.g. 0)
A a = new B();

Possible solutions:
  - Prohibit type narrowing through hotswapping if instances exist
  - Replace type system violating object pointers with null
Summary

Find Affected Classes ➔ Topological Sorting ➔ Load New Classes ➔ GC run

- Swap Pointers
- Update Instances

Update CP Caches ➔ Deoptimize Methods ➔ Continue Execution

Arbitrary Changes Possible

No Additional Runtime Overhead
Future Work

Performance
Remove need for GC run in case of „small“ updates (adding methods or fields)

Improved IDE Support
Connect with the NetBeans refactoring facilities

Security
API for selecting „safe“ program execution points
Patch + detailed technical documentation available at

http://wikis.sun.com/display/mlvm/HotSwap

Feel free to post any questions to

mlvm-dev@openjdk.java.net or wuerthinger@ssw.jku.at

Thanks
for your attention!

Q & A