VM interfaces: GNU/Classpath vs. OpenJDK vs. PhoneME

Christian Thalinger
Vienna University of Technology
twisti@complang.tuwien.ac.at

What is the VM interface

- some Java classes have to "talk" to the VM
  - this is done via native Java methods
  - these native Java methods are the "VM interface"
- various Java classes need native VM support, e.g.
  - threading
  - GC
  - management
  - JVMTI
  - ...
- CACAO uses an extra abstraction layer to support all different interfaces
  - can be seen afterwards in the examples

GNU Classpath

- first VM interface implemented
- very good abstraction layer (`VM*.java` classes)
  - is designed to support different VMs
  - reference implementation in `vm/reference/`
    - CACAO replaces 13 of them
- uses JNI naming scheme (e.g. `Java_java_lang_*`)
- one `.c` file per interface class
- GNU Classpath 0.96.1 has 251 native interface methods
  - CACAO implements 193 of them (including `sun.misc.Unsafe`)
  - all other are implemented by GNU Classpath itself

Example: GNU Classpath

- cacao/src/native/vm/gnu/java_lang_VMClass.c:

```
/*
 * Class:     java/lang/VMClass
 * Method:    isPrimitive
 * Signature: (Ljava/lang/Class;)Z
 */
JNIEXPORT int32_t JNICALL Java_java_lang_VMClass_isPrimitive(
    JNIEnv *env, jclass clazz, java_lang_Class *klass)
{
    classinfo *c;

    c = LLNI_classinfo_unwrap(klass);

    return class_is_primitive(c);
}
```

PhoneME

- second VM interface implemented
- very similar to GNU Classpath interface
  - but no additional abstraction layer
  - was very easy to integrate
- uses JNI naming scheme (e.g. `Java_java_lang_*`)
- one `.c` file per interface class
- all native interface methods must be implemented by the VM
- CLDC-1.1 has 59 native `java.*` interface methods
  - CACAO implements 46 of them
- ...plus 61 native `com.sun.*` interface methods
  - CACAO implements 14 of them

Example: PhoneME

- cacao/src/native/vm/cldc1.1/java_lang_Class.c:

```
/*
 * Class:     java/lang/Class
 * Method:    isInterface
 * Signature: ()Z
 */
JNIEXPORT int32_t JNICALL Java_java_lang_Class_isInterface(
    JNIEnv *env, java_lang_Class *this)
{
    classinfo *c;

    c = LLNI_classinfo_unwrap(this);

    return class_is_interface(c);
}
```

OpenJDK

- last interface added ...and it's different (and was a lot of work)
- uses different naming scheme
  - called `JVM_*`
- all VM interface methods in one file
  - `openjdk/hotspot/src/share/vm/prims/jvm.cpp`
- about 220 interface methods (some of them are OS specific)
  - CACAO implements a lot of them, but not all (don't know the exact number)
    * CACAO also supports the `-XX:+TraceJVMCalls` debug option as HotSpot does

Example: OpenJDK

- cacao/src/native/vm/sun/jvm.c:

```
/* JVM_IsInterface */

jboolean JVM_IsInterface(JNIEnv *env, jclass cls)
{
     classinfo *c;

     TRACEJVMCALLS("JVM_IsInterface(env=%p, cls=%p)", env, cls);

     c = LLNI_classinfo_unwrap(cls);

     return class_is_interface(c);
}
```

OpenJDK (2)

- native VM interface methods are registered in the VM
  - this is done by the static class initializer

- `openjdk/jdk/src/share/classes/java/lang/Class.java`:

```
public final class Class<T> ...
    ...
    private static native void registerNatives();
    static {
        registerNatives();
    }
    ...
}
```

OpenJDK (3)

- opendjk/jdk/src/share/native/java/lang/Class.c:

```
static JNINativeMethod methods[] = {
    ...
    { "isInterface", "()Z", (void *) &JVM_IsInterface },
    ...
};

JNIEXPORT void JNICALL
Java_java_lang_Class_registerNatives(JNIEnv *env, jclass cls)
{
    methods[1].fnPtr = (void *)(*env)->GetSuperclass;
    (*env)->RegisterNatives(env, cls, methods,
                            sizeof(methods)/sizeof(JNINativeMethod));
}
```

OpenJDK (4)

- sometimes also stuff like this happens:

```
JNIEXPORT jint JNICALL
Java_java_lang_System_identityHashCode(JNIEnv *env, jobject this, jobject x)
{
    return JVM_IHashCode(env, x);
}
```

- I guess these methods are inlined by the JIT compiler anyways

Conclusion

- GNU Classpath and PhoneME VM interfaces are very similar
- OpenJDK's is a bit different, but nice too
- it's a good idea to have an additional abstraction layer in your VM