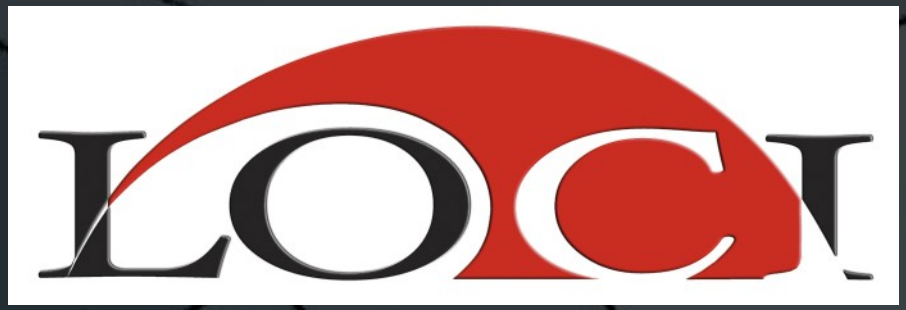
Cross-language Interoperability of Java Software

Curtis Rueden
Laboratory for Optical and
Computational Instrumentation

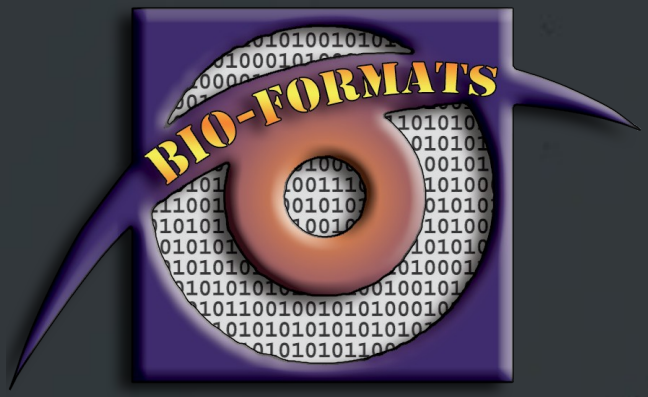
Background

- LOCI is a cross-disciplinary research lab:
  - **Engineers & physicists** design and build microscope acquisition systems
  - **Computer programmers** write software to acquire, visualize, analyze, manage data
  - **Biologists** use these tools to do life sciences research



History

- Based on research exploring integration of Bio-Formats with non-Java code
- Some of our integration targets:
  - OME server (Perl)
  - ITK (cross-platform C++)
  - WiscScan (MSVC++)
  - CellProfiler (Python)



- More information available online:
  - http://www.loci.wisc.edu/bio-formats/interfacing-non-java-code

Motivation

- Goal: Invoke an existing Java library from non-Java code (e.g., C/C++ or Python)
- Backwards from usual... why?
  - Avoid full language translations
  - Java has numerous advantages
    - Fewer cross-platform issues
    - Good time performance
    - Great tools
    - Huge community with many libraries

The Alternative

- Use C or C++ and wrap to Java et. al
- Advantages:
  - Easier to interface with C from other languages
  - More mature tools exist for this direction
- Disadvantages:
  - Longer development time needed
  - As your library grows more complex, portability becomes a nightmare

No Magic Bullet

- Plethora of integration solutions
- Best one for your project depends on:
  - Size and complexity of your Java code
  - Target software for integration
    - Which language(s)?
    - Which platform(s)?
  - Needed granularity of the integration
  - Your tolerance for dependencies
  - Many other factors

Inapplicable Technologies

- Some familiar technologies do not apply:
  - **JNA:** easily call native code from Java
  - **SWIG:** creates higher-level language wrappers around C/C++ code
  - **Java RMI:** limited to Java clients only
- Ideally, we want cross-platform, cross-language solutions
- Open source is also very important

In-process Approaches

- Directly invoke Java code from your app:
  - A) **JNI:** Spawn internal Java Virtual Machine (JVM) and pass data across a bridge
  - B) **Compile-time:** Compile Java source or Java bytecode to native code or bytecode
  - C) **Runtime:** Execute Java bytecode with an alternative interpreter (e.g., .NET)

In-process Approaches

- Examples:

  A) JNI:
  Java Invocation API, Jace, CppWrap

  B) Compile-time:
  GNU Compiler for Java (GCJ)

  C) Runtime:
  IKVM.NET

In-process Approaches

- Strengths:
  - Tight (API-level) integration
  - Minimal performance overhead
  - Fewer security considerations
- Weaknesses:
  - No shared state between processes
  - Limited portability

Inter-process Communication

- Various techniques for exchanging data between multiple running programs:
  - A) **Local communication:** Share information via a common resource
  - B) **Messaging:** Send and receive messages using the network stack
  - C) **Object request broker (ORB):** Middleware for transferring objects between processes
- Client-server model, with Java as server

Inter-process Communication

- Examples:

A) **Local communication:**
files, pipes

B) **Messaging:**
sockets, XML-RPC, JMS

C) **Object request broker (ORB):**
Ice, CORBA, EJB3

Inter-process Communication

- Strengths:
  - Share state between multiple processes on multiple machines
  - Broad portability and language support
- Weaknesses:
  - Object marshalling incurs significant overhead
  - Potentially vulnerable to security exploits
    - Example: ImageJ socket listener

Files & Pipes

- Simple to implement
- Deceptively powerful
- Surprisingly robust and performant
  - Pipes performance can exceed JNI
  - Caveat: Pipes have issues on Windows
  - Caveat: Using files on disk is slow
- Use case: OME Perl Server

Java Invocation API

- JNI mechanism for calling Java from C++
- Link to Java shared lib (libjvm.so/jvm.dll)
- Native application instantiates and interrogates its own personal JVM
- Several problems:
  - Lots of boilerplate and complexity
  - JVM behaves badly in some ways
    - E.g.: memory use, AWT on Mac, exiting
  - How to build cross-platform?

Jace, Jar2Lib & CppWrap

- Higher-level tools to ease this use of JNI:
  - **Jace:** a library for wrapping a Java class in a corresponding C++ proxy class
  - **Jar2Lib:** a library that uses Jace to wrap a Java JAR file as a C++ shared library
  - **CppWrap:** a Maven plugin that uses Jar2Lib to create a C++ project wrapping a given Maven project

Jace, Jar2Lib & CppWrap

- Wraps the entire Java API
- Eliminates some of JNI's difficulties:
  - No more boilerplate code
  - Uses CMake to compile cross-platform
- Still some problems:
  - Java behavior issues (memory, etc.)
  - Dependencies: Java, Jace, Boost
- Source: showinfJNI.cpp vs. showinf.cpp

Use Case: ITK

- Goal: provide a Bio-Formats plugin for ITK
- Implemented three different solutions:
  1. BF-ITK-jace: use BF-CPP library
  2. BF-ITK-jni: use raw JNI
  3. BF-ITK-pipe: use a system call with pipes

ZeroC Ice

- Advantages:
  - Performant binary inter-process remoting
  - Language agnostic “slice” definitions
  - Small dependency footprint (<2MB)
- Disadvantages:
  - Versioning of remote API is very rigid
  - Inferior performance to JNI
  - Longer development effort (fewer tools)
- Use case: OMERO server

Compilers & Runtimes

- GCJ
  - Compile Java into native code using GCC
  - In development since 1996
- IKVM.NET
  - Execute bytecode within .NET runtime
  - Works cross-platform with Mono
- Some parts of Java not fully implemented
- Java is a moving target for these solutions

Use Case: WiscScan

- Goal: use Bio-Formats to write OME-TIFF files from WiscScan acquisition software
- Explored four different approaches:
  1. IKVM.NET (circa 2007)
  2. JVMLink (custom sockets solution)
  3. ZeroC ICE (“BF-ICE” library)
  4. JNI via BF-CPP

Conclusions

- Cross-language integration is hard!
- Think carefully when choosing a solution
- Exploring multiple options often pays off
- I hate C++...

Discussion

- Questions?
- What other technologies have you used?