



Evaluation of Java with real-time extensions for flight systems

Kirk Reinholtz

Jet Propulsion Laboratory
California Institute of Technology
Mission Data System

4th Quality Mission Software Workshop
Dana Point, CA, May 7-9 2002

Why Java for Flight Systems?

- Accelerates the adoption of current best-software practices.
 - Programmers adopt design patterns methodology since standard class libraries usage is based on design patterns.
- Java appears to enhance productivity and reduce defects.
 - Plentiful evidence of widespread adoption
- Java is easier to use safely than is C++
 - C++ is more complicated and difficult to use effectively
- Java is a complete platform.
 - Standard class library includes collections, threading, networking and all other commonly needed capabilities.

Why Java for Flight Systems? (cont)

- Java has many capabilities that must be added to C++
 - Garbage collection
 - Serialization of code and data
 - Dynamic linking
 - Reflection
 - Dynamic optimization
 - Compact code representation
 - Support for mixed language systems
 - Various IPC models
 - Components
 - ...

Major Concerns

- Concern: Java is not deterministic.
- Concern : Java's performance is inadequate.
- Concern: Java is a greater risk than C++.
- Concern: Java is not ready for current MDS developments.
- Concern: Weak floating-point model.

Approach to retiring the concerns

- Performed a two-phase study
 - First phase: Is Java real? Obviously, YES!
 - Second phase: Identify issues and retire or mitigate each
- This presentation is a summary of the results of phase two
- This study was performed in 1999. Since then Java with RTSJ has grown to address almost all of the issues.

Summary of issues to be retired or mitigated

- Non-deterministic behavior
- Performance and Resources
- Mixed-language Issues
- Integrating Java with existing JPL and vendor capabilities
- Tool Chain Support
- Training/Experience
- Verification
- Hardware Impacts

Non-deterministic Behavior

- Question 1: What do you do about non-deterministic cost of garbage collection?
- Question 2: How do you do priority-based scheduling on vanilla JVM?
- Question 3: How do you handle non-deterministic cost of object creation?
- Question 4: What do you do about heap fragmentation? Closed-loop control of a spacecraft is a real-time process

Non-deterministic Behavior

- Question 1: What do you do about non-deterministic cost of garbage collection?
 - Approach
 - Analysis
 - Result
 - Vanilla Java has non-deterministic garbage collection
 - Forthcoming RTSJ provides deterministic garbage collection
 - Conclusion
 - Must work around in the short term
 - Use RTSJ GC and memory management in the long term
 - Mitigation
 - Vanilla Java does not allow control over execution of the GC
 - Use standard idioms that avoid the creation of garbage
 - Object pools
 - Object recycling
 - Local variables

Non-deterministic Behavior

- Question 2: How do you do priority-based scheduling on vanilla JVM?
 - Approach
 - Analysis
 - Result
 - Vanilla Java specification does not specify behavior of priorities: can even ignore
 - RTSJ specifies sufficiently tightly to allow full use of priority mechanisms
 - Conclusion
 - Requires near-term workaround
 - Mitigation
 - Use vendor-specific information
 - Minimize and track use of priority mechanisms until RTSJ is implemented

Non-deterministic Behavior

- Question 3: How do you handle nondeterministic cost of object creation?
 - Approach
 - Analysis
 - Result
 - Time to execute "new" is non-deterministic in vanilla Java
 - Forthcoming RTSJ provides deterministic object creation
 - Conclusion
 - Requires near-term workaround
 - RTSJ memory management provides good solutions
 - Mitigation
 - Don't use vanilla Java in tight real-time situations
 - Use standard idioms to avoid untimely object creation

Non-deterministic Behavior

- Question 4: What do you do about heap fragmentation?
 - Approach
 - Analysis
 - Results
 - Not a problem with Java, just with particular GC algorithm implementations
 - Not a new problem: C and C++ have the same issue
 - Java enables heap defragmentation
 - Language specification enables transparent solutions
 - RTSJ provides ways to avoid fragmentation in the first place
 - Conclusion
 - Not a problem for non real-time applications
 - For real-time applications, make sure vendor has a defragmenting GC
 - Use RTSJ memory management features to minimize fragmenting of the heap
 - Mitigation
 - Use standard idioms to avoid creating fragmented heap

Multi-process and multi-language issues

- Question 1: What inter-process communication (“IPC”) mechanisms are available in Java?
- Question 2: How does Java call non-Java code?
- Question 3: What are the issues with JNI?
- Question 4: How do we do IPC between Java and C++ threads?

Multi-process and multi-language issues

- Question 1: What Inter-process communication (“IPC”) mechanisms are available in Java?
 - Approach
 - Analysis
 - Result
 - Java Remote Method Invocation (“RMI”)
 - CORBA
 - Via JNI, any capabilities in underlying operating system e.g. message queues
 - Conclusion
 - Not an issue
 - Mitigation
 - None required

Multi-process and multi-language issues

- Question 2: How does Java call non-Java code?
 - Approach
 - Analysis
 - Result
 - Java provides Java Native Invocation (“JNI”)
 - Currently specified for C and C++
 - Conclusion
 - Java can call C and C++ code
 - Mitigation
 - None required

Multi-process and multi-language issues

- Question 3: What are the issues with JNI?
 - Approach
 - Analysis
 - Results
 - Performance
 - May be a little worse than a normal method call, but not necessarily so.
 - Data accesses may be expensive, due to format conversions
 - Check your vendor documentation
 - Generality, ease of use
 - Each side has access to all the capabilities of the other side
 - Interface is a little clumsy (in the name of portability)
 - Robustness
 - Inflicts C/C++ risks on Java

Multi-process and multi-language issues

- Question 3: What are the issues with JNI? (continued)
 - Results (continued)
 - Exceptions
 - C/C++ code can raise Java exception
 - Java exception can be passed to C/C++ code
 - GC
 - Zero-copy access to Java objects
 - » JNI provides for requests for zero-copy access to Java objects
 - » Vendor not obliged to provide zero-copy access
 - If we had a full-blown GC capability in C++, how would it interact with Java GC
 - » Java and C++ heaps are separate, so there will be no interactions
 - Debugging
 - Debugging covered in later slides
 - Conclusion
 - JNI is ready for use today
 - Mitigation
 - None required

Multi-process and multi-language issues

- Question 4: How do we do IPC between Java and C++ threads
 - Approach
 - Analysis
 - Results
 - Two IPC forms “built in” to Java
 - RMI - Java-to-Java
 - CORBA - Java-to-anything (anything-to-anything, in fact)
 - Others likely to follow due to Java being an “Internet Language”
 - Can use JNI to implement other forms of IPC as necessary
 - OS message queues already adapted
 - Conclusion
 - Java doesn't introduce any new IPC problems
 - Mitigation
 - None required

Integrating Java with existing JPL and Vendor capabilities

- Question 1 : How do we accommodate existing C or C++ implementations that we do not have time to redo in Java or cannot do in Java ?
 - Approach
 - Segments of the DS-1 code were re-implemented in Java to expose Java to C interface issues and to expose JVM to vxWorks issues (Phase 1)
 - Results
 - Java to C (via JNI) interfaces worked well
 - The vxWorks Java environment allowed Java threads and vxWorks tasks to work together with mixed Java and C threads at different priorities
 - Exposed limitations of the multi-language debugging tool set
 - Conclusion
 - No additional risk
 - Mitigation
 - none

Integrating Java with existing JPL and Vendor capabilities

- Question 2 : Are there any language or architectural issues that complicate or preclude using legacy code ?
 - Approach
 - Phase 1 tested interactions with RTOS and DS-1 flight code.
 - Examined the following multi-language capabilities: cross-language exceptions, cross-language IPC, CORBA compatibility and cross-language memory management.
 - Results
 - Java code functioned flawless inside the DS-1 flight code. Deeply nested C calls made debugging difficult.
 - Cross-language exceptions and cross-language IPC work. Java is compatible with CORBA. Java and C++ manage memory in separate heap spaces.
 - Conclusion
 - Avoid deeply nested calls across the JNI.
 - No additional risks
 - Mitigation
 - none

Integrating Java with existing JPL and Vendor capabilities

- Question 3: Is Java appropriate for numerical applications like NAV?
 - Approach
 - Evaluate current implementation and future plans of support for the IEEE floating point standard
 - Result
 - Java has limited floating point capabilities. Only single precision (32 bit) and double precision (64 bit) per IEEE-754
 - Some navigation areas might require ≥ 80 bit
 - JVMs must guarantee machine independent floating point results
 - Default is no floating point hardware use. JVM floating point is done in software.
 - Performance is much less than with hardware support
 - Java Grande Group is currently formulating solutions, but not in the near-term
 - Conclusion
 - If Java is used for numerical applications, use it cautiously.
 - Mitigation
 - Do prototyping to understand numerical behavior of the language.
 - Choose the right tool for the job.
 - FIX: There's a JSR on this, note it

Performance and Resources

- Question 3: What's the impact of using Java in a resource constrained system?
 - Approach
 - Analyze optimization techniques used by Java compilers
 - Result
 - Byte codes are a compact representation of a program. Java programs have a small footprint compared to C++ programs.
 - The Java kernel (aka JVM) is larger. Depending on the size of the trusted classes, the breakeven point is somewhere between .5 and 5 megabytes.
 - Dynamic compilers convert byte codes to machine codes as needed. Footprint size can be traded against performance.
 - Compiled byte codes can be cached to improve performance. Performance can be traded against footprint size.
 - Current dynamic compilers do not allow users to change optimization strategies dynamically.
 - Conclusion
 - Risk can be eventually retired. Short term mitigation needed now.
 - Mitigation
 - Optimize for best overall balance of footprint and performance.
 - Explore static compiler path

Performance and Resources

- Question 5: What's the performance impact of runtime checking?
 - Approach
 - Evaluate approach to run-time checking on several JVMs
 - Result
 - The language requires arrays be checked for validity. JVMs must support runtime checking to be compliant.
 - Runtime checks decrease performance.
 - Runtime checks can only be removed by converting the byte codes to machine code with a static compiler.
 - Runtime checking is a *good* thing. Removes the risk of out of bounds arrays and dereferenced pointers. These errors can be handled gracefully with the Java exception mechanism
 - Conclusion
 - No additional risk
 - Mitigation
 - High performance code segments can be written in C++ if Java performance is inadequate.

Training and Experience

- Question: How long does it take to become an effective Java programmer?
 - Approach
 - Query vendors at Java One. Report on MDS activity.
 - Result
 - Many offerings of 5 day classes.
 - Easy to Learn
 - A many MDS developers are already familiar with Java. They have been prototyping in Java without training. (Compare to C++: training but no prototyping.)
 - Java is being taught at universities. New hires are likely to know Java.
 - Conclusion
 - Risk that can be retired eventually but need short term mitigation
 - Mitigation
 - Provide training for MDS developers

Verification

- Question: Can Java be reliably verified?
 - Approach
 - Report produced by the Automated Software Engineering group at NASA Ames
 - Result
 - With respect to verifiability, the study group saw no apparent disadvantages of Java vs. C++ for non-real time. In general Java is a superior to C++ with respect to verifiability.
 - Java WORA bytecodes are easier to verify than platform dependent C++ object code.
 - Java has strong typing and runtime checks
 - Java has no pointer calculus
 - Java has a built-in thread model. A specific verification solution can be built for the Java thread model.
 - Ames is actively developing a verification environment for Java
 - However, Java is less mature than C++ and doesn't have the same tool support
 - Conclusion
 - A benefit
 - Mitigation
 - none

Summary

- Java is technically ready
 - Java is gaining widespread industry support
- Java offers considerable benefits today and the promise of greater benefits in the near term
 - Enhanced productivity, greater reliability
- Put the development infrastructure in place for Java development
- Plan for and select applications for development
 - Careful selection should be possible so that near term commitments can still be met
- Progress toward having multi-language implementations in the long term