Ada-Java Middleware for Legacy Software Modernization

Kelvin Nilsen
Aegis Software Modernization Uses Soft Real-Time Java Based on PERC Ultra

What’s a Java VM and why did it replace me?

It doesn’t eat, it doesn’t sleep, and it doesn’t stink up the Command Intelligence Center.

Picture courtesy Lockheed Martin.
“Voice over” courtesy of Atego.

USS Bunker Hill (CG 52) was first of 22 Ticonderoga-class guided-missile cruisers to undergo extensive capability upgrade as part of Cruiser Modernization Program
Context and Motivation for “Ada Java Method Invocation”

- Billions of dollars of software IP implemented in Ada: energy, transportation, aerospace, defense
- Many companies are shifting attention to Java for new development
- The transition to Java is easier if the value of existing Ada IP can be preserved
  - AJMI enables Ada and Java to be efficiently and robustly combined in mixed-language applications
  - Majority of existing Ada software based on Ada 83 and Ada 95 standards
  - Minimize certification disruption by building AJMI on Ada 95 run time environment
Alternative Approaches (Related Work)

- Roll your own interface with Ada ⇔ C ⇔ JNI ⇔ Java
  - Cumbersome, error prone, expensive to maintain
  - Allows C (Ada) to manipulate Java data but does not directly allow Java to access C (Ada) data.

- GNAT Ada Java Interface Suite (AJIS)
  - Automatically generates Java wrappers for Ada specifications
  - Java programmers can extend the auto-generated Java wrapper and an Ada proxy can represent this extended Java object
  - But GNAT AJIS does not generate Ada wrappers for Java classes
  - GNAT AJIS programs are vulnerable to memory leaks and inter-language dangling pointers
  - Relies on heap memory management and Ada 2005 features
AJMI Capabilities

- Auto-generates Ada wrappers for Java and Java wrappers for Ada
- Enables mixed-language object orientation
  - Ada tagged types may override Java
  - Java may override Ada tagged types
- Compatible with Ada 95 and Ada 2005 run times, Ada 83/95/05 source
- Compatible with standard edition and safety-critical Java
- Different middleware implementations enable Java and Ada to reside in shared memory of same process, in isolated partitions of ARINC 653 or MILS OS or Linux, on different networked processors
- Restrictive AJMI subset enablea reliable integration of JSR-302-style Java with Ravenscar-style Ada in stack memory
AJMI interface generation tools: ava

Ada package specification → ava → Java class definition(s)
(API allows Java application to invoke Ada services, and maps invocations to ajmi communication infrastructure)
AJMI interface generation tools: jada

- **Java class**
- **jada**
  - **Ada package specification** *(allows Ada application to invoke Java services)*
  - **Ada package body** *(maps invocations to ajmi communication infrastructure)*
AJMI Execution model

Java proxies for Ada objects (courtesy of ava)

Ada proxies for Java objects (courtesy of jada)
Sample mixed-language application (Java perspective)

public class JavaMain {
    public static void main(String[] args) {
        UARTDriver uart = new UARTDriver(iocallback); // create an Ada object: UARTDriver is Java Proxy
        JavaGUIListener listener = new JavaGUIListener();
        JavaMonitorGUI gui = new JavaMonitorGUI(listener);
        DeviceMonitor monitor = new DeviceMonitor(gui); // create an Ada object: DeviceMonitor is JavaProxy
        JavaApplication app = new JavaApplication(uart);

        // JavaIOCallback is a Java extension of an ava-generated Java proxy
        JavaIOCallback iocallback1 = new JavaIOCallback(app, JavaIOCallback.INPUT AVAILABLE);
        JavaIOCallback iocallback2 = new JavaIOCallback(app, JavaIOCallback.OUTPUT READY);
        app.doWork(); // spawns a Java background thread

        // ask Ada to call back to my Java code under certain circumstances
        uart.notifyWhenInputAvailable(iocallback1); // invoke Ada service
        uart.notifyWhenOutqueueEmpty(iocallback2); // invoke Ada service
        monitor.monitorUART(uart, 1000, gui); // invoke Ada and do not return
    }
}
Memory organization overview

- Java and Ada have very different “temporary memory” models
  - Ada allocates on the stack. Strong typing assures absence of dangling pointers.
  - Java allocates on the heap. Garbage collection assures absence of dangling pointers.

- The mixed-language programming model allows each language to allocate in its own style.

- By default, shared objects have a stack-oriented life time:
  - Java objects may live longer than an “interaction”, but proxies are “disabled” at the moment when stack memory would normally be reclaimed

- Optional (lower integrity) protocols are available to deal with objects that live longer than a particular “interaction”.
  - These protocols would be discouraged in safety-critical integrations
Thread organization overview

- In Java, thread identity is important because a Java thread that “locks” a Java resource is allowed to “relock” the same resource without restriction.
- If Java calls Ada and Ada calls back to Java, the call-back into Java needs to be the same Java thread.
- Model: an Ada task melds with a Java thread to become a conceptual AJMI thread.
  - Implementations may optimize certain scenarios by allowing the Ada task and Java thread to run as a single operating system thread.
  - Memory allocations sometimes need to be taken from the stack frame of the companion language’s run-time environment at the point of the most recent AJMI invocation. Examples follow.
Thread stack usage: Ada invokes Java method

Ada Thread Stack

- Ada Application Stack
- ajmi frame for infrastructure

Java Thread Stack

- ajmi frame for infrastructure
- stack frame for invoked service

ajmi invocation

ajmi communication
Thread stack usage: Java calls back to Ada
Thread stack usage: invoked Ada stack allocates Java object

Ada Thread Stack

Ada Application Stack
ajmi frame for infrastructure
ajmi frame for infrastructure
stack frame for invoked service
ajmi frame for infrastructure

allocate Ada proxy

Java Thread Stack

ajmi frame for infrastructure
stack frame for invoked service
ajmi frame for infrastructure
proxyed object (ajmi frame)

ajmi communication

text on slide:

extension of ajmi level 2
Status

- Detailed design has been completed but the technology is not yet fully implemented
  - This design is much more ambitious than existing technologies
  - Represents Java objects in a style that is natural to the Ada 95 environment
  - Represents Ada 83/95/05 objects in a style familiar to Java programmers
  - Supports reliable and efficient inter-language sharing of stack-allocated objects

- An initial implementation integrates Object Ada 95 with PERC Ultra in shared memory as a single Linux x86 process
  - Support for other Ada, Java, processors, operating systems and middleware configurations will be prioritized according to customer demand

- Performance measurements and user experiences with the initial implementation may result in changes to the API design, the AJMI run time, and the AJMI tool chains