A Practical Comparison Between Java and Ada in Implementing a Real-Time Embedded System

Eric Potratz
University of Northern Iowa

Choosing a Virtual Machine

- ***Real-Time Specification for Java***
  - Contains features critical for real-time systems
  - Only one reference implementation exists
  - Too large for our embedded system

Choosing a Virtual Machine

- **SimpleRTJ**

  (Developed by RTJ Computing Ltd.)

  + Easy to port to the real-time operating system used in the Ada version of the project (MaRTE OS)
  + Small size
  – Lacks real-time features like those in the *Real-Time Specification for Java*

Convenient Java Features

- Native Methods
  - **Java**
    - Particular methods can be declared as *native*
    - Execute machine code, not Java bytecodes
  - **GNAT** & **Ada**
    - Can import & call C/C++ functions
    - Can execute specific sequences of assembly language instructions
  - Provides low-level access to specific hardware that Java and Ada do not

Convenient Java Features

- Concurrency Support
  - *Thread* objects
    - Equivalent to Ada’s *tasks*
    - Allow concurrent control in an application
  - *synchronized* methods
    - Used to enforce mutual exclusion on an object’s operations
    - Used to implement basic equivalents to Ada’s *protected types*
  - Concurrency support better integrated into the Java language than into Ada

Java’s Drawbacks

- Means to implement *barriers* on “protected type” operations

Barriers

- Associated with an operation in a protected type
- Assigned a particular condition
  - When the condition is true:
    - The barrier is “open”
    - Tasks/Threads can execute the operation
  - When the condition is false:
    - The barrier is “closed”
    - Calling tasks/Threads are suspended until the condition becomes true

Barriers In Ada

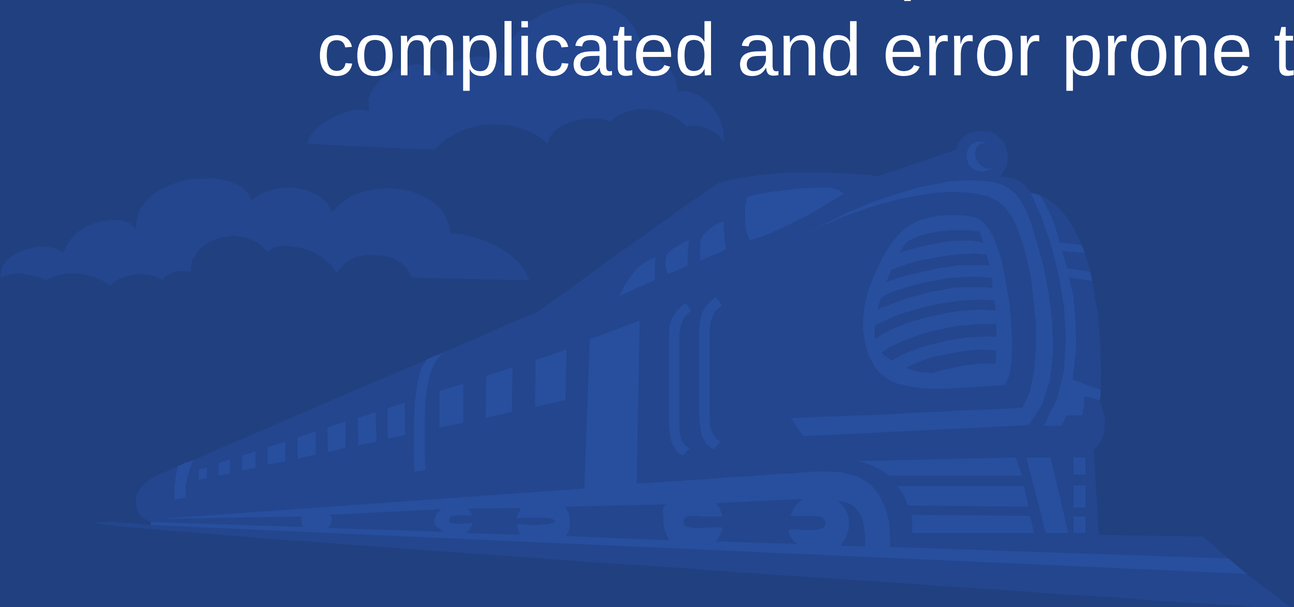
- A barrier can be created by:
  - Creating a protected type
  - Declaring an *entry operation* in that protected type
  - Assigning the *entry condition* to that operation
- Runtime system takes care of the dynamic aspects of enforcing the entry barrier

Barriers in Java

- Java provides low-level methods to produce similar behavior
  - `wait()` — suspends a Thread and places it in the object’s set of suspended Threads
  - `notify()` — “notifies” (wakes up) one Thread in the object’s set of suspended Threads
  - `notifyAll()` — notifies all Threads in the set of suspended Threads

Barriers in Java

- These are primitive operations
  - Have to worry about algorithms that will produce equivalent behavior to barriers
  - These low-level operations are more complicated and error prone to use



Barriers in Java

- Drawbacks to `wait()`, `notify()`, and `notifyAll()`
  - Their low-level nature complicates adding more barriers to a class
  - Exacerbates nested object lock deadlock
  - Inheritance anomaly

Java's Drawbacks

- Thread scheduling in non-real-time Java
  - Arbitrary Thread scheduling
    - Ada's specification defines how to choose tasks in any situation where one needs to be chosen to use resources next
    - Non-real-time Java may choose Threads arbitrarily in some situations
    - The *Real Time Specification for Java* provides virtual machine extensions to support Thread scheduling policies that address this

Java’s Drawbacks

- Thread scheduling in non-real-time Java
  - Priority inversion is not addressed
    - Ada addresses this by using *priority* inheritance when it schedules tasks
    - Non-real-time Java provides no way to address priority inversion
    - The *Real-Time Specification for Java* does, though, through the ability to enable particular Thread scheduling policies

Java’s Drawbacks

- Memory management in non-real-time Java
  - Memory is dynamically allocated
  - Objects cannot explicitly be destroyed
  - The “garbage collector factor”
  - Real-time Java specifications provide remedies involving non-heap memory
    - *Real-Time Core Extensions*
    - *Real-Time Specification for Java*

Java’s Drawbacks

- Operations available to access single bits of data
  - Useful in implementing device drivers
  - **Ada**: can define a record type and map its components onto particular bits within a primitive data type
  - **Java**: provides low-level bit shifting and bit masking operations
    - More complicated to use and error-prone
    - Unintuitive behavior

Java’s Drawbacks

- Class Initialization Code and Class Dependencies
  - Ada compilers
    - Check package initialization code for dependency problems
    - Report any problems
  - Java compilers
    - Don’t check the same for similar class initialization code
    - Class initialization process is more error-prone

Conclusion

- How usable is non-real-time Java in implementing this kind of system?
  - The last two drawbacks can be worked around
  - The other drawbacks make non-real-time Java less than ideal than Ada for this particular embedded real-time application
  - Java is a “work in progress” for embedded real-time applications like this one