Type checking for the Real-Time Specification for Java

Daniel Tang
Professor Jan Vitek
Real-time Systems

- Systems that rely on some time constraint for complete correctness
  - Not necessarily speed, but timeliness
- Examples:
  - Pacemakers
  - Nuclear power plant control
  - AI tournament chess player
Real-time Java

- Java’s widespread use and relative simplicity make it a desirable language to program in.
- Java is not suitable for real-time systems by default.
  - Thread priority inversion
  - Garbage collection

![Graph showing run time and iteration number]
First specification to be introduced with the Java Community Process (JSR-1)

Goal was to improve real-time programming in seven areas

- Thread scheduling/real-time threads
- Memory management/hierarchy to avoid GC
RTSJ Memory Areas

Immortal Memory “a”

Scoped Memory “b”

Scoped Memory “e”

Scoped Memory “c”

Scoped Memory “d”
Unsafe memory accesses will result in exceptions being thrown

Unacceptable in safety-critical applications, yet hard to eliminate

Java annotations can aid in automated detection of errors

› @Override, @Deprecated, e.g.
RTSJ Annotations

- **@Immortal**
  - Singleton instance where all scoped memories reside

- **@ScopeDef(name="b", parent="a")**
  - Identifies a memory location by name and parent scope

- **@Scope("a")**
  - Declares a class to be within a scope
Memory regions must form a tree
Class A can perform limited operations on a class in a parent scope: R/W primitives or annotated types
Safety Rules

- Class A cannot access class B unless B is declared in the same scope.
- Class A cannot allocate objects of type B unless B is in the same scope or is a non-annotated type.
Implementation

- Utilizes Apache’s Byte Code Engineering Library (BCEL) and Java 5
- All previously listed rules accomplished—not all rules enumerated
  - Casting a scope-annotated type to a non-annotated type (Object) is illegal
  - Method calls must be checked against implicit up-casts
Future Work

- Code refactoring to use BCELs provided
- Visitor class
- Implement stack emulation—necessary for type checking in method invocations
- Support for several unmentioned annotations
- Support for local variable annotations