Ravenscar Java: A high-integrity profile for real-time Java

Jagun Kwon, Andy Wellings and Steve King
University of York, UK.

For many, Java is the antithesis of a high-integrity programming language. Its combination of object-oriented programming features, its automatic garbage collection and its poor support for real-time multi-threading, are all seen as particular impediments.

Topics

- Problems for Java and high integrity systems (HIS)
- The role of the RTSJ
- Learning from the Ada experience
- A framework for high-integrity RTSJ-based systems
- Ravenscar-Java

Java and HIS

- There is difficulty in applying static analysis techniques due to the inherent dynamic nature of Java:
  - run-time dispatching
  - dynamic class loading
- Problems compounded if the analysis has to be done at the byte code level as opposed to language level
- Problems with the Java Memory Model making it difficult to define the semantics of multi-threaded programs (being addressed by JSR - 133)
- Lack of assertions (being addressed by JSR 41)
- Particular minor problems
  - Return values quietly discarded; side effect in expressions
  - No support for subtypes and enumeration types (have to role your own)

Java and Real-Time

- We all know the problems here:
  - poor support for real-time threads and real-time scheduling,
  - lack of confidence in real-time garbage collection,
  - non responsiveness of thread asynchronous interaction mechanisms,
  - inability to interact with interrupts and devices
  - complex virtual machine
  - and so on
- Because of this we have
  - RTSJ
  - J2ME

Problems with RTSJ

- Aimed at the more dynamic soft real-time market rather than the HIS market

- Semantics not yet well defined
- Adds to the complexity of the VM
- Adds to the complexity of the programming model

- Result is that static analysis of RTSJ programs is even more problematic

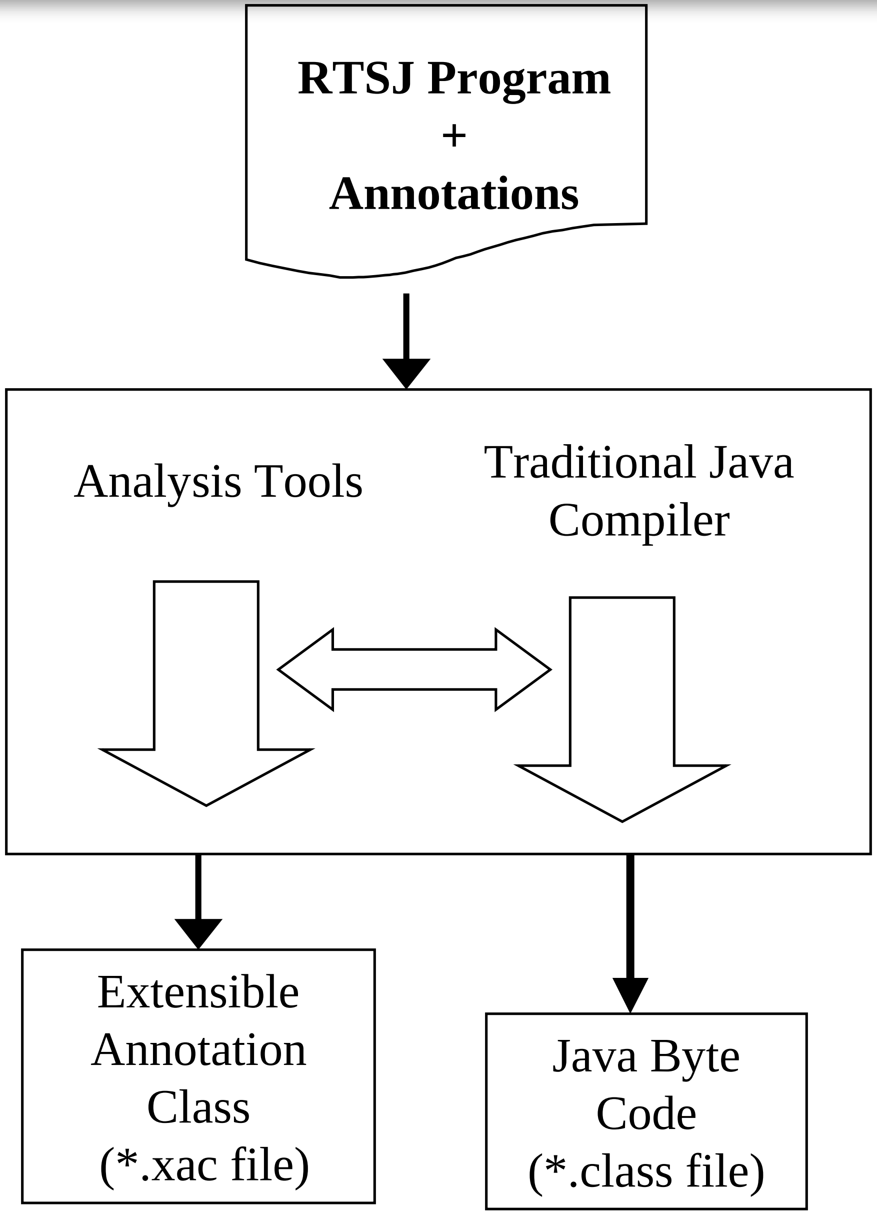
Learning from Ada

- Although Ada does have many faults, it does have a good technical solution for both real-time and high-integrity systems
- The SPARK Ada subset is very conservative, in particular
  - no tasking
  - no OOP
- In recent years, the Ravenscar Ada Tasking profile has become a de facto standard for HIS
  - programs can be analysed for their schedulabilty
  - the run-time support is simple and can be engineered to a high-level of integrity (e.g. Aonix's Raven)
  - will soon become part of the ISO standard

Ravenscar Java

- Goal
  - Produce a high integrity subset (profile) of Java and the RTSJ
  - Fit within the J2ME framework
  - Provide a high integrity Ravenscar VM (RVM)
- Approach
  - Apply the U.S. NRC guidelines on the use of languages in HIS (currently, it does not consider Java)
  - Expand the guidelines to be more positive about concurrency
  - Use the Ravenscar computational model to guide the definition of the RTSJ profile
  - Where necessary extend the language via annotations to allow: WCET and other static analysis techniques

Offline Target Independent Architecture



RTSJ Program
+
Annotations
Analysis Tools
Traditional Java Compiler
Extensible Annotation Class (*.xac file)
Java Byte Code (*.class file)

Analysis Tools: E.g.

- Check profile conformance
- Perform high-level WCETA
- Escape analysis
- Model checking

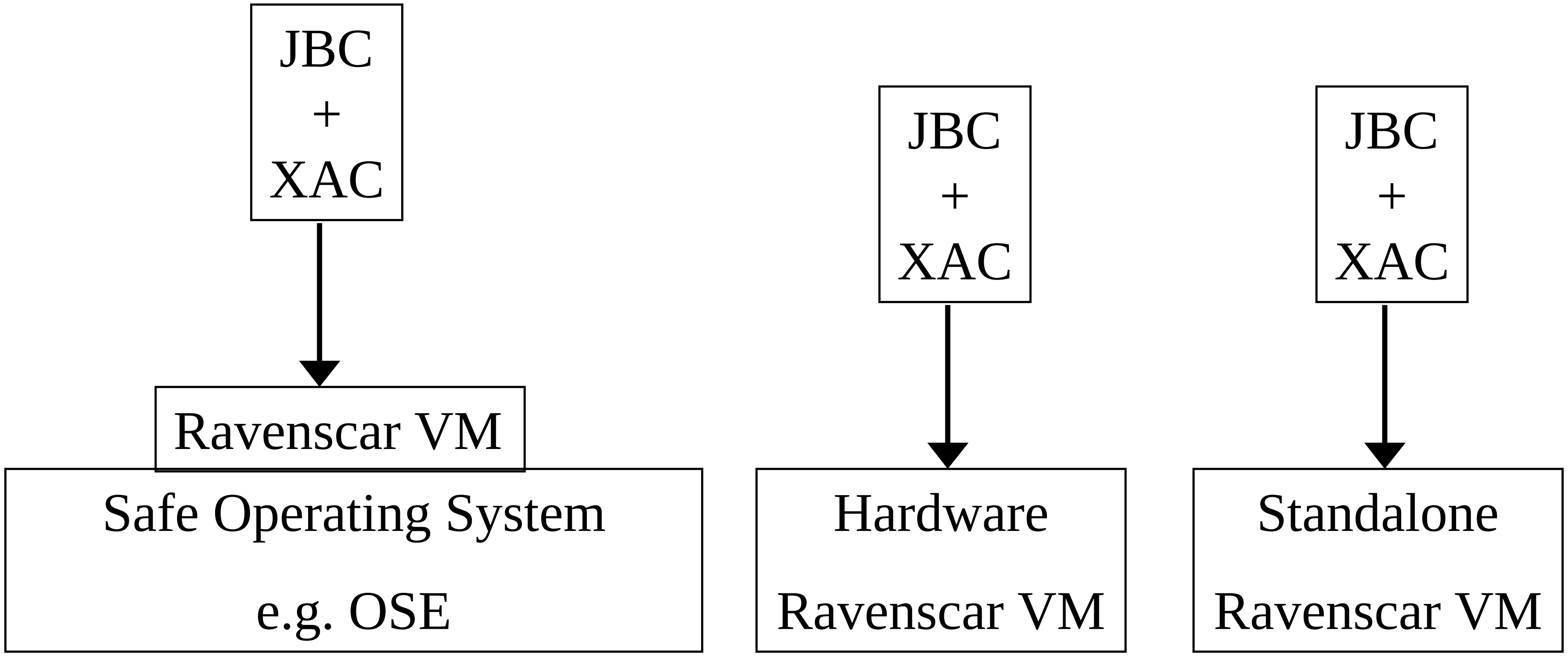
Results of analysis stored in xac file. In particular, JBC frequency vectors

Offline Target-Dependent Analysis

- Takes a main class and a Java Path and constructs the tree of objects created by the program
- Takes the timing characteristics of the target RVM
- Performs low-level WCET on the real-time threads
- Performs schedulability analysis

- In a semi static environment, this could be done by the RVM after loading the main class

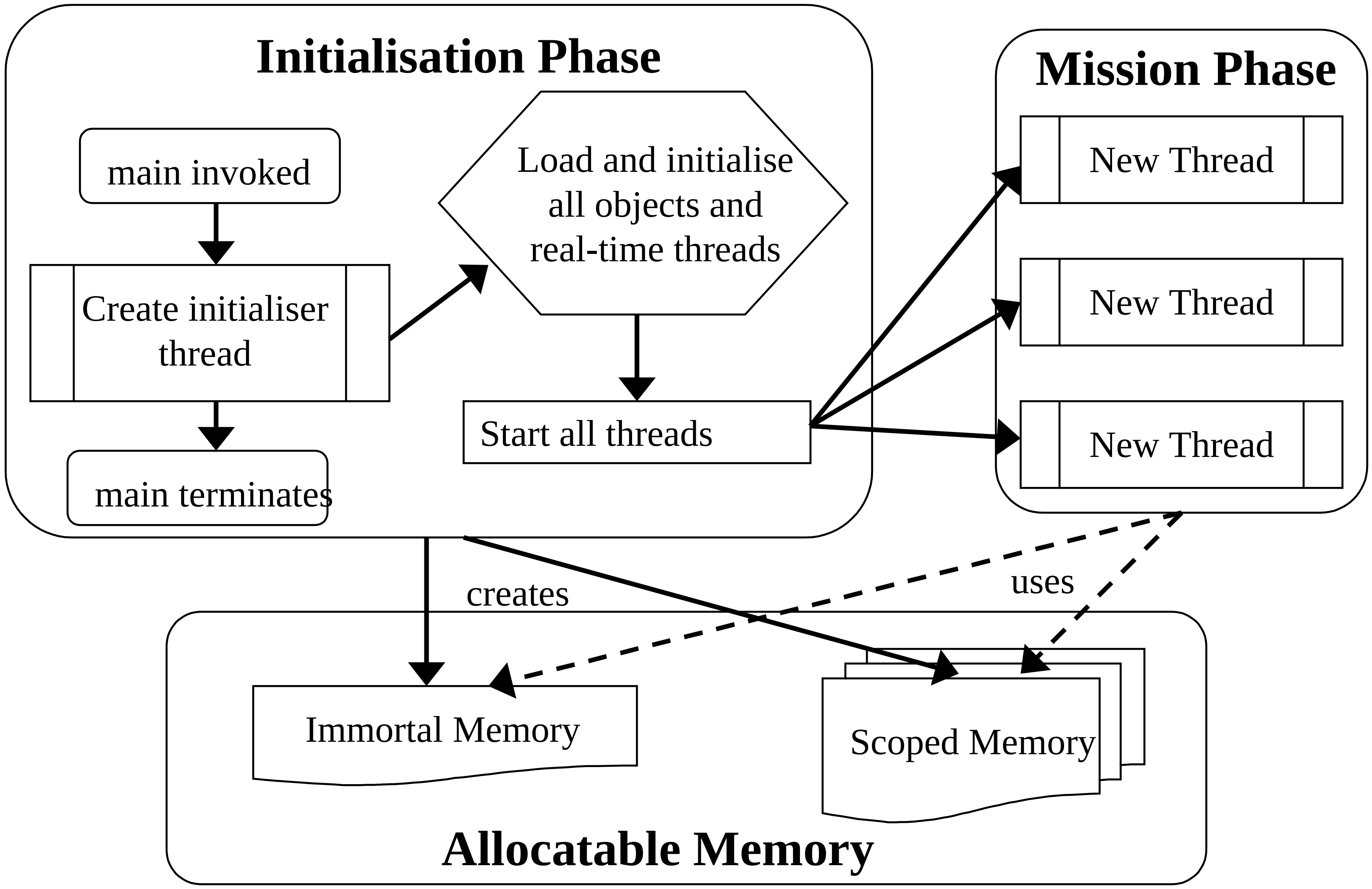
Online Architecture



Computational Model

- Fixed priority pre-emptive scheduling
- Priority ceiling inheritance
- Periodic no-heap real-time threads
- Sporadic no-heap event handlers
- Two phases of program execution

Program Execution



The Profile

- Predictability of memory utilization
- Predictability of timing
- Predictability of control and data flow (not covered here)

- The profile changes some of the access modifiers of the classes, constructors, and methods in order to ensure they cannot be used directly by the programmer. The changes are always more restrictive and, hence programs, will always execute on non-Ravenscar implementations.

The Initializer Thread

```
package ravenscar;
import javax.realtime.*;

public class Initializer extends RealtimeThread
{
  public Initializer()
  {
    super( new PriorityParameters(
         PriorityScheduler.MAX_PRIORITY), null, null,
         ImmortalMemory.instance(),
         null, null);
  }
}
```

Typical Application

```
import ravenscar.*;

public class MyApplication extends Initializer
{
  public void run()
  {
    // Create memory areas (LTMemory areas)
    // Create no heap real-time threads
    // All objects required by application must be
    // created here or in the constructors of objects
    // created.
    // Start all threads.
  }

  public static void main (String [] args)
  {
    MyApplication myApp = new MyApplication();
    myApp.start();
  }
}
```

Memory Area Restrictions

- No nested calls to enter LTMemory areas
- No LTMemory areas shared between threads (all communications via Immortal memory)

- Goal: to significantly reduce the complexity of the VM in this area and (with escape analysis) to eliminate the run-time checks
- Implication: memory management is done at the thread level rather than the object level

MemoryArea Class

```
package ravenscar;
public abstract class MemoryArea
{
  protected MemoryArea(long sizeInBytes);
  protected MemoryArea(javax.realtime.SizeEstimator size);

  public void enter(java.lang.Runnable logic);
  public void executeInArea(java.lang.Runnable logic)
         throws InaccessibleAreaException;

  public static MemoryArea getMemoryArea(java.lang.Object object)

  public long memoryConsumed();
  public long memoryRemaining();
  public java.lang.Object newArray(...)
     throws IllegalAccessException, InstantiationException;

  public java.lang.Object newInstance(java.lang.Class type)
    throws IllegalAccessException, InstantiationException;
  public java.lang.Object newInstance(...)
    throws IllegalAccessException, InstantiationException;
  public long size();
}
```

Subclasses

```
public final class ImmortalMemory extends MemoryArea
{
  public static ImmortalMemory instance();
}

public abstract class ScopedMemory extends MemoryArea
{
  public ScopedMemory(long size);
  public ScopedMemory(SizeEstimator size);
  public void enter();
  public int getReferenceCount();
}

public class LTMemory extends ScopedMemory
{
  public LTMemory(long size);
  public LTMemory(SizeEstimator size);
}
```

Predictability of Timing

- Computation model support only periodic threads and sporadic event handlers
- No on-line scheduling analysis
- No overrun or deadline miss handlers

Scheduling

```
package ravenscar;
public interface Schedulable extends java.lang.Runnable
{
}

public abstract class Scheduler
{
}

public class PriorityScheduler extends Scheduler
{
  public static final int MAX_PRIORITY;
  public static final int MIN_PRIORITY;
}
```

Release Parameters

```
package ravenscar;
public class ReleaseParameters
{
  protected ReleaseParameters();
}

public class PeriodicParameters extends ReleaseParameters
{
  public PeriodicParameters(AbsoluteTime startTime,
                            RelativeTime period);
  protected AbsoluteTime getStartTime();
  protected RelativeTime getPeriod();
}

public class SporadicParameters extends ReleaseParameters
{
  public SporadicParameters(RelativeTime minInterarrival);
  protected RelativeTime getMinInterarrival();
}
```

Threads

```
package java.lang;
public class Thread implements Runnable
{
  Thread();
  Thread(String name);

  void start();
}
```

Real-time Threads

```
package ravenscar;
public class RealtimeThread extends java.lang.Thread
             implements Schedulable
{
  RealtimeThread(PriorityParameters pp,
         PeriodicParameters p);
  RealtimeThread(PriorityParameters pp,
         PeriodicParameters p, MemoryArea ma);
  public static RealtimeThread currentRealtimeThread();
  public MemoryArea getCurrentMemoryArea();
  void start();
  static boolean waitForNextPeriod();
}
```

No Heap Real-time Threads

```
public class NoHeapRealtimeThread extends RealtimeThread
{
   NoHeapRealtimeThread(PriorityParameters pp,
         MemoryArea ma);
   NoHeapRealtimeThread(PriorityParameters pp,
         PeriodicParameters p, MemoryArea ma);
   void start();
}
```

Periodic Threads

```java
package ravenscar;
public class PeriodicThread extends NoHeapRealtimeThread
{
  public PeriodicThread(PriorityParameters pp,
         PeriodicParameters p, java.lang.Runnable logic);
  public void run();
  public void start();
}
```

Example Implementation

```
public class PeriodicThread extends NoHeapRealtimeThread
{
  public PeriodicThread(...)
  { super(pp, p, ImmortalMemory.instance());
    applicationLogic = logic;
  }
  public void run()
  {
    boolean noProblems = true;
    while(noProblems) {
      applicationLogic.run();
      noProblems = waitForNextPeriod();
    }
    // Deadline missed. If allowed, a recovery routine here
  }
  public void start()
  {
    super.start();
  }
}
```

Asynchronous Event Handlers

```
package ravenscar;
public class AsyncEventHandler implements Schedulable
{
  AsyncEventHandler(PriorityParameters pp,
         ReleaseParameters p, MemoryArea ma);
  AsyncEventHandler(PriorityParameters pp,
         ReleaseParameters p, MemoryArea ma,
         java.lang.Runnable logic);

  public MemoryArea getCurrentMemoryArea();
  protected void handleAsyncEvent();
  public final void run();
}
```

Bound Handlers

```
public class BoundAsyncEventHandler
             extends AsyncEventHandler
{
  BoundAsyncEventHandler(PriorityParameters pp,
         MemoryArea ma, ReleaseParameters p);
  BoundAsyncEventHandler(PriorityParameters pp,
         MemoryArea ma, ReleaseParameters p,
         java.lang.Runnable logic);

  protected void handleAsyncEvent();

}
```

Sporadic Handlers

```
public class SporadicEventHandler
       extends BoundAsyncEventHandler
{
  public SporadicEventHandler(PriorityParameters pri,
                              SporadicParameters spor);
  public SporadicEventHandler(PriorityParameters pri,
                              SporadicParameters spor,
                              java.lang.Runnable);
  public void handleAsyncEvent();
};
```

Async Events

```
package ravenscar;
public class AsyncEvent
{
  AsyncEvent();
  void addHandler();
  void fire();
  void bindTo();
}

public class SporadicEvent extends AsyncEvent
{
  public SporadicEvent(SporadicEventHandler handler);
  public void fire();
}

public class SporadicInterrupt extends AsyncEvent
{
  public SporadicInterrupt(SporadicEventHandler handler,
                           java.lang.String happening);
}
```

Current Status

- Currently producing an annotation aware tool systems
- Investigating the ease with which static analysis can be performed on the subset (e.g. escape analysis, model checking)
- RVM analysis and construction