Java Takes Flight: Time-portable Real Time Programming with Exotasks

Joshua Auerbach, David F Bacon, Daniel T Iercan, Christoph M Kirsch, VT Rajan, Harald Roeck, Rainer Trummer
High Level Summary

- **Exotasks**: a programming model for Java real time
  - development tools in Eclipse
  - specialized support in Java VM
- **JAviator**: a quad-rotor model helicopter controlled by exotasks
- Measurements collected during actual JAviator flights
  - non-interference from the global heap GC
  - maintain time-portability when moved to different hardware
- And some other measurements
Goal: Improve Real Time Java Programming

RT Java Feasible (as in IBM WebSphere Real Time) because of
- Real-time garbage collection (metronome)
- Ahead-of-time compilation
- Real-time thread scheduling via RTSJ

Some remaining Problems
- Java’s “write once, run anywhere” promise doesn’t apply to timing behavior
  - Exotasks: provide time portability
- RT GC: gives up throughput to get latency and hits inherent limitation. Instead, tradeoff programming convenience for latency
  - NHRTs (part of RTSJ)
  - Eventrons (PLDI 2006)
  - Reflexes (VEE 2007)
  - Exotasks: support the largest subset of Java to date
Exotasks are Time Portable

Logical Execution Time (LET, inspired by Giotto)

- Exotasks are isolated tasks communicating via typed channels
  - Channels perform deep copy of Java objects
- Those tasks that participate in I/O are assigned explicit real-time deadlines
  - Don’t execute early even if their inputs are ready
- All other tasks are scheduled according to data dependencies
  - Isolation keeps scheduling differences from affecting the outcome
Exotasks Escape GC “Less Restrictively”

- Each exotask has a private heap
- Collection of that heap is scheduled by the exotask scheduler
- Independent of the main heap collector
- Each exotask uses the Java memory model exactly as intended
- Only restrictions:
  - No use of static to communicate across threads
  - No thread creation, finalization, weak/soft references
A Simple Exotask Program - Editing Begun
Exotask Program - Graphical Phase Complete

Period = 5ms

Offset = 0ms

Type = SensorData

Offset = 2ms

Type = ActuatorData
Initial Generated Code For Compute Task

```java
public class Compute implements Runnable {
    private ExotaskInputPort<javiator.util.SensorData> in0;
    private ExotaskOutputPort<javiator.util.ActuatorData> out0;

    public Compute(ExotaskInputPort<javiator.util.SensorData> in0,
                   ExotaskOutputPort<javiator.util.ActuatorData> out0) {
        this.in0 = in0;
        this.out0 = out0;
    }

    public void run() {
        // TODO Auto-generated method stub
    }
}
```
A Bigger Program (JAviator Controller)
1. Validate, 2. Instantiate, 3. Schedule & Execute

Exotask Graph

Exotask Code

```java
public class Controller implements Runnable {
    private ExotaskInptPort in0;
    private ExotaskOutputPort out0;

    public Controller(ExotaskInputPort in0, ExotaskOutputPort out0) {
        this.in0 = in0;
        this.out0 = out0;
    }

    public void run() {
        double[] sensorData = (double[]) in0.getValue();
        out0.setValue(new Double(control(sensorData)));
    }

    private double control(double[] sensors) {
        // the actual control algorithm ...
    }
}
```

Exotask Graph Specification

Exotask Runtime System
Code Validation

- Use RTA to conservatively approximate call graph
- All classes initialized at time of analysis
  - actual objects occupying all reachable statics are known
- Rule out `putstatic`
- Rule out `getstatic` of mutable objects
  - Infer immutability using `final` and `private` ("effectively final")
- `Getstatic` of immutable objects allowed but
  - such objects are pinned (as with Eventrons and Reflexes)
  - sharing is rendered invisible by disabling monitor locking
  - Thus, semantics are "copy on read"
- Thread creation, finalization, soft/weak references disallowed
- All else is allowed
- By permitting immutable static objects to be read we enable many JDK classes (HashMap, HashSet, etc) to be used
1. Validate, 2. Instantiate, 3. Schedule & Execute
The Program Runs

Period = 5ms

Offset = 0ms

Offset = 2ms
The Program Runs

- Period = 5ms
- Offset = 0ms
- Offset = 2ms
The Program Runs

- **Period = 5ms**
- **Offset = 0ms**
- **Offset = 2ms**
The Program Runs

- **Period = 5ms**
- **Offset = 0ms**
- **Offset = 2ms**
The Program Runs

- **Period = 5ms**
- **Offset = 0ms**
- **Offset = 2ms**
The Program Runs

- **Period = 5ms**
- **Offset = 0ms**
- **Offset = 2ms**
The Program Runs

Period = 5ms

Sensor

TT Single Mode

Offset = 0ms

Compute

Offset = 2ms

Actuator

Offset = 0ms

Offset = 2ms
The Program Runs

- Period = 5ms
- Offset = 0ms
- Offset = 2ms
The Program Runs

Period = 5ms

Offset = 0ms

Offset = 2ms
The JAviator
Logical Hookup

Control Terminal

Exotask Controller
Experiments

TuningFork

Instrumented Exotask System
Freedom From GC Interference

No GC in Progress
N=30913

Mean=19.9, StdDev=1170.6

GC in Progress
N=1381

Mean=20.0, StdDev=902.4
Time Portability

4-way 2GHz
N=33548

Mean=19943.5, StdDev=1236.2

1-way 1.4GHz
N=32444

Mean=19939.1, StdDev=1159.1
After Recent Improvements (Grounded)

4-way 2GHz
N=89156

Mean=20000.0, StdDev=62.2

600 mhz
N-34094

Mean=20000.0, StdDev=174.5
Gumstix Processor

400mhz N=127508

Mean=20002.0, StdDev=830.3
In Paper but Not Discussed

- Advanced concepts: modes, conditions, communicators
- Timing grammars are pluggable
  - The “time triggered” grammar is shown in examples
  - The HTL grammar is a complete injection of the HTL programming language (superset of Giotto)
- Schedulers are pluggable
- Provision is made for distribution across machines
  - Also pluggable
- More about related work and future work