Real-Time Software

Programming Real-Time Abstractions

René Rydhof Hansen

12 October 2010

Today's Goals

- Programming real-time abstractions
  - Periodic Tasks
  - Aperiodic/Sporadic Tasks
- Real-Time Event Handlers
- Controlling I/O jitter

Real-Time Tasks

Definition

Tasks with... real-time (timing) constraints (duh!)

Ada and C/POSIX Real-Time Support

Low-level primitives; build-your-own.

Real-Time Java Support

Explicit (library) support: class of real-time threads with special attributes:

- Release parameters
- Scheduling parameters
- Memory parameters
- Processing group parameters

Warning: little or no enforcement of constraints/guidelines

Real-Time Java: RealtimeThread class

```
package javax.realtime;
public class RealTimeThread extends Thread
  implements Schedulable {
    public RealTimeThread(
              SchedulingParameters  scheduling,
              ReleaseParameters  release,
              MemoryParameters  memory,
              MemoryArea  area,
              ProcessingGroupParameters  group,
              Runnable  logic);

  public void release();
  public void start();
  public boolean waitForNextPeriod();
  public boolean waitForNextRelease();
}
```

Real-Time Java: ReleaseParameters class

```
package javax.realtime;
public class ReleaseParameters implements Cloneable {
  protected ReleaseParameters(RelativeTime cost,
                              RelativeTime deadline,
                              RelativeTime blockingTerm,
                              AsyncEventHandler overrunHandler,
                              AsyncEventHandler missHandler);

  public Object clone();
  public RelativeTime getCost();
  public void setCost(RelativeTime cost);
  ...  // getters and setters
  public boolean setIfFeasible(RelativeTime cost,
                               RelativeTime deadline,
                               RelativeTime blockingTerm);
}
```

Real-Time Java: ReleaseParameters class

```
package javax.realtime;
public class ReleaseParameters implements Cloneable {
  protected ReleaseParameters(RelativeTime cost,
                              RelativeTime deadline,
                              RelativeTime blockingTerm,
                              AsyncEventHandler overrunHandler,
                              AsyncEventHandler missHandler);

  public Object clone();
  public RelativeTime getCost();
  public void setCost(RelativeTime cost);
  ...  // getters and setters
  public boolean setIfFeasible(RelativeTime cost,
                               RelativeTime deadline,
                               RelativeTime blockingTerm);
}
```

Programming Periodic Tasks

Periodic Tasks

The fundamental, bread and butter, tasks in an RTS.

Periodic Tasks in Ada

```
task body Periodic_T is
   Next_Release : Time;
   Release_Interval :
      constant Time_Span := Milliseconds(...);
begin
   -- read clock and calculate Next_Release
   loop
      -- sample data or do calculation
      delay until Next_Release;
   end loop;
end Periodic_T;
```

Programming Periodic Tasks

Periodic Tasks in C/Real-Time POSIX

```
#include <signal.h>
#include <time.h>
#include <pthread.h>
#include <misc/timespec_operations.h>
void periodic_thread()
{
  struct timespec next_release, remaining_time;
  struct timespec thread_period;
  /* read time and calculate next_release */
  while(1) {
    /* sample data or do calculation */
    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,
      &next_release, &remaining_time);
    add_timespec(&next_release,
      &next_release, &thread_period);
  }
}
```

Programming Periodic Tasks

Library Support for Periodic Tasks: PeriodicParameters class

```
package javax.realtime;
public class PeriodicParameters
  extends ReleaseParameters {
  public PeriodicParameters(HighResolutionTime start,
            RelativeTime period, RelativeTime cost,
            RelativeTime deadline,
            RelativeTime blockingTerm,
            AsyncEventHandler overrunHandler,
            AsyncEventHandler missHandler);

  public RelativeTime getPeriod();
  public void setPeriod(RelativeTime period);

  ... // getters and setters
}
```

Programming Periodic Tasks

Periodic Tasks in Real-Time Java

```
public class Periodic extends RealtimeThread {
  public Periodic(PeriodicParameters P) {...}

  public boolean run() {
    boolean deadlineMet = true;
    while(deadlineMet) {
      // code to be run each period
      ...
      deadlineMet = waitForNextPeriod();
    }
  }
```

- Similar to Ada/C but no explicit computation of `NextRelease`
- Typical Java: lots and lots of writing

Programming Aperiodic and Sporadic Tasks

Aperiodic vs. Sporadic

- Sporadic tasks have minimum inter-arrival time > 0, aperiodic tasks do not
- Impossible to perform RTA for aperiodic tasks

Handling aperiodic (and sporadic) events

- Put events in queue
- Queue overflow?
  - Throw an exception
  - Ignore
  - Replace last
  - Extend queue

Programming Aperiodic Tasks in Ada

```
protected Aperiodic_Controller is
   procedure Interrupt;
   entry Wait_For_Next_Interrupt;
private
   Call_Outstanding : Boolean := False;
end Aperiodic_Controller;

protected body Aperiodic_Controller is
   procedure Interrupt is
   begin
      Call_Outstanding := True;
   end Interrupt;

   entry Wait_For_Next_Interrupt when Call_Outstanding is
   begin
      Call_Outstanding := False;
   end Wait_For_Next_Interrupt;
end Aperiodic_Controller;
```

Programming Aperiodic Tasks in Ada

```
task body Aperiodic_T is
  begin
    loop
      Aperiodic_Controller.Wait_For_Next_Interrupt;
      -- ...
    end loop;
  end Aperiodic_T;
```

Aperiodic Tasks in C/Real-Time POSIX

Similar to Ada.

Programming Aperiodic Tasks in Real-Time Java

The AperiodicParameters class

```
package javax.realtime;
public class AperiodicParameters
  extends ReleaseParameters {
  public AperiodicParameters(RelativeTime cost,
           RelativeTime deadline,
           AsyncEventHandler overrunHandler,
           AsyncEventHandler missHandler);

  public static final String arrivalTimeQueueOverflowExcept;
  public static final String arrivalTimeQueueOverflowIgnore;
  public static final String arrivalTimeQueueOverflowReplace;
  public static final String arrivalTimeQueueOverflowSave;

  public String getArrivalTimeQueueOverflowBehavior();
  public void setArrivalTimeQueueOverflowBehavior(
                String behavior);
}
```

Programming Aperiodic Tasks in Real-Time Java

An aperiodic task

```
public class Aperiodic extends RealtimeThread {
  public Aperiodic(PriorityParameters PP,
                   AperiodicParameters P) { ... }

  public void run() {
    boolean deadlineMet = true;

    while(deadlineMet) {
      // code to be run each period
      ...
      deadlineMet = waitForNextRelease();
    }
  }
}
```

Real-Time Events and Event Handlers

Tasks vs. Event Handlers

- Tasks
  - Long lived
  - Stateful
  - Periodic activity/inactivity
  - Scheduled
  - Competing with other tasks for resources
- Event Handlers
  - Short lived
  - Stateless
  - One shot
  - "Immediate"
  - Alternative: scheduled

Implementing Real-Time Event Handlers

Ada

- Supported by: `Ada.Real_Time.Timing_Events`
- Most effectively executed directly from clock interrupt handler
- Executed directly by Ada run-time system: minimal release jitter

C/Real-Time POSIX

- Supported through signals
- Support for both one-shot and periodic timers
- Problem with multi-threaded programs: signal sent to whole process

Real-Time Java

- Supported by: `Timer`
- Examples: `OneShotTimer`, `PeriodicTimer`

Input/Output Jitter

Definition (Input/Output Jitter)

The variation in input/output latency

- Output latency = task latency
- Must be controlled
- Often done by (smart) I/O devices themselves

Controlling I/O Jitter with Real-Time Events

- Good for small execution time or minimal jitter
- Always executes at interrupt priority level (interferes with rest of program)
- All I/O handled with same (top-)priority

Example (Timing event in Ada)

```
procedure Timer(Event : in out Timing_Event) is
begin
   -- read from sensor interface
   Data_Available := True;
   Next_Time := Next_Time + Input_Period;
   Set_Handler(Input_Jitter_Control,
     Next_Time, Timer'Access);
end Timer;
```

Controlling I/O Jitter with Separate Tasks

- Three tasks (threads): input, processing, output
- More control
  - Precise timing for signals (often) unknown/unspecified
  - Fine(r) grained priorities for I/O
- More overhead

Controlling I/O Jitter with Separate Tasks

Example (Sensor reading task in C/Real-Time POSIX)

```
void sensor_thread(parameters *params) {
  struct timespec next_release, remaining_time;
  next_release = params -> start;
  clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME,
    &next_release,&remaining_time);
  while(1) {
    /* read sensor */
    add_timespec(&next_release,
      &next_release,&params->period);
    clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME,
      &next_release,&remaining_time);
  }}
```

Controlling I/O Jitter with Periodic Event Handlers

- Supported in Real-Time Java
- Use two periodic event handlers: input, output
  - Little or no jitter
  - Inexpensive
- Use task for processing
  - Busy waiting for data to process

Example (Sensor reader in Real-Time Java)

```
public class SensorReader extends AsyncEventHandler {
  ...
  public void handleAsyncEvent() {
    // read sensor
    synchronised(this) {
      dataAvailable = true;
      notifyAll();
    }
  }
```

Summary

Summary:

- Programming real-time abstractions
  - Periodic Tasks
  - Aperiodic/Sporadic Tasks
- Real-Time Event Handlers
- Controlling I/O jitter