5. Real-Time Programming

5.2 Real-Time Java

Roadmap of Section 5.2

- Real Time Specification for Java
- RTSJ Features
  - RealtimeThreads
  - Memory Management
  - RawMemoryAccess
  - Asynchronous Transfer of Control
  - Asynchronous Event Handling
- Reference Implementation / Available Impl.
  - Status of RTSJ
- J Consortium

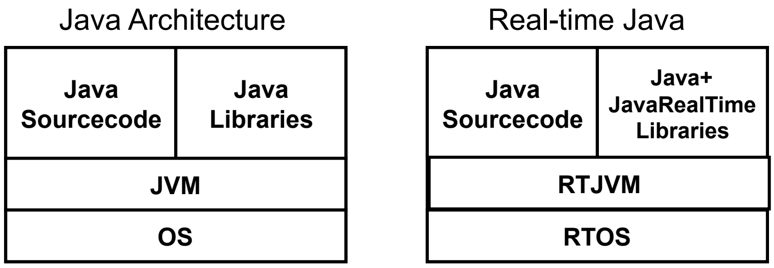
History

- Dec. 1998 java specification request for real time extension for java
- Expert group – SUN, IBM, QNX Software Lab, Nortel, Rockwell, Timesys ..
- Greg Borella (IBM) first specification lead
- Sept. 1999 first public review of specification
- Late in 2001 Timesys volunteered to create the reference implementation
- Final Specification 12/11/2001
- 2003 Sun announced Mackinac project: first commercial implementation of RTSJ

Motivation

- Usage of advantages of Java
  - Cross-platform capabilities
  - Object orientation, Type Safety
  - Developers and Tools available, Rapid Application Development
- Improve real-time properties of java
  - Deterministic execution times
  - Specify real-time scheduling / known start / stop times of threads
  - Specify sufficient memory management
  - Direct access to hardware / memory

Real Time Specification for Java (RTSJ)



- Standard Java API + Real-time Extensions :
- javax.realtime.*

RTSJ: Major Specification Features

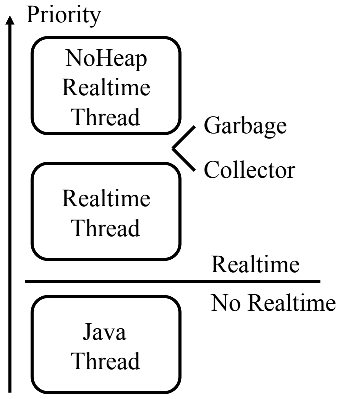
- Real-time threads with precise defined scheduling
- Mechanisms that support writing code that is not influenced by garbage collection
- Asynchronous event handlers to handle events from outside the virtual machine
- Asynchronous transfer of control
- Mechanisms that allow to control where objects will be allocated in memory
- Direct memory access

RTSJ Scheduling

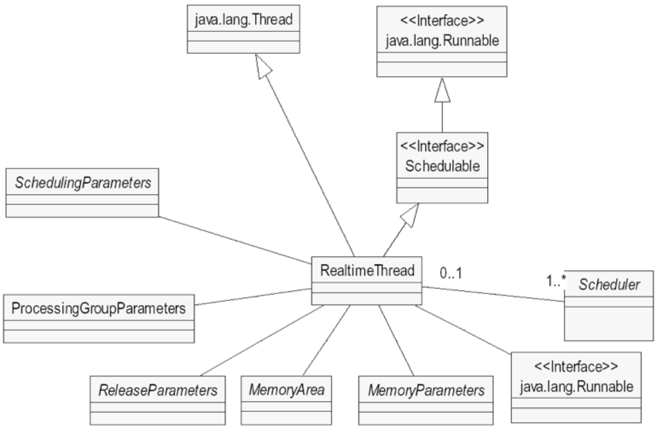
- Scheduling manages scheduling / dispatching of schedulable objects
- Schedulable object – implements *Schedulable*
- RTSJ specifies default scheduling algorithm
  - Fixed-priority preemptive scheduling
  - FIFO
  - At least 28 scheduling priorities
  - Highest priority thread always runs
- Custom scheduler can be implemented

Threads

- NoHeapRealtime Threads
  - Hard real-time
  - Higher priority that gc
  - No references to heap memory
- Realtime Thread
  - Soft real-time
  - Can be interrupted by gc
  - References to heap allowed



RealTimeThread



Periodic Threads

```
int pri = PriorityScheduler.instance().getMinPriority()+10;
PriorityParameters prip = new PriorityParameters(pri);
RelativeTime period = new RelativeTime(20 /* ms */,0 /* ns */);
PeriodicParameters perp = new         PeriodicParameters
   (null,period,null,null,null,null);
RealtimeThread rt= new RealtimeThread(prip,perp) {

public void run() {
   int n=1;
   while (waitForNextPeriod() && (n<100)) {
        System.out.println("Hello "+n);
        n++;
   }
}
};
rt.start();
```

Scheduler

- Default Scheduler : *PriorityScheduler*
  - No change of priority during runtime
- Performs feasibility analysis for sets of schedulable objects
- Cost overrun handler / missed deadline handler per process
- Controlled via *SchedulingParameter*
- Additional Scheduler must implement abstract base class *Scheduler*
- *Can be installed via : RealtimeThread.*

  public void **setScheduler**(Scheduler scheduler)

$T_1, T_2, \ldots, T_n$ – Tasks to be performed in the real time system

$C_1, C_2, \ldots, C_n$ – Cost of each task (how long it takes to run each task)

$R_1, R_2, \ldots, R_n$ – Release time for each task (time that task becomes available to run)

$D_1, D_2, \ldots, D_n$ – Deadline for each task (when each task needs to be complete)

Asynchronous Event Handling

- Real-time and embedded systems are coupled to the real world
- Events in the real world are asynchronous
- RTSJ specifies a mechanism to bind a schedulable object to the occurrence of an event
- When the event occurs the object's run state changes to ready-to-run and is scheduled according its parameters
- Implementation should support hundreds of ev.

Asynchronous Event Handling

- An instance of AsyncEvent represents something that can happen
- AsyncEventHandler implements Schedulable
  - RealTimeThread / NoHeapRTThread
- Default Constructor : All properties inherited from current thread
- An instance of AsyncEventHandler has a method handleAsyncEvent() which contains the logic that should execute when the event occurs
- Method *run()* invokes *handleAsynchEvent()*

AynchEvent Class

- `public synchronized void addHandler ( AsynchronousEventHandler handler)`
  - Adds a handler to the set defined for this AsynchEvent
- `public void bindTo(String happening)`
  - Binds this AsynchEvent to an external event (a happening)
  - Happening is an implementation dependent value that binds this AsynchEvent to some external event
- `public synchronized void fire()`
  - Schedules the run() method of each handler associated with this event

Interrupt Handling Example

```
import java.realtime.*;
public class HardwareInterruptExample extends AsyncEvent{
   private int interruptNum;
   public HardwareEventExample(int num) {
       interruptNum = num;
}
public void setHandler(AsyncEventHandler h) {
   super.addHandler(h);
   super.bindTo(interruptNum);
}
class HardwareEventHandler extends AsyncEventHandler{
   private int interruptCount = 0;
   public void handleAsyncEvent(){
   interruptCount++;
   // Driver code follows}
}
```

Embedded Operating Systems

15

Time

- „Allow *description of a point in time* with up to *nanosecond accuracy and precision* (actual accuracy and precision is dependent on the precision of the underlying system).“
- „Allow *distinctions between absolute points in time*, times relative to some starting point, and a new construct, *rational time*, which allows the efficient expression of occurrences per some *interval of relative time*.“
- Abstract HighResolutionTime implements Comparable
- RelativeTime, AbsoluteTime, RationalTime

Embedded Operating Systems

16

Timers

- Triggers behaviour at a particular point in time
- Special form of asynchronous events
- *OneShotTimer*
  - Fires off once at the specified time
- *PeriodicTimer*
  - Fires off at the specified time and then
  - periodically with a specified interval
- Clock : interface to the system's real-time clock

Timer Example

```
PeriodicTimer pt = new PeriodicTimer(
   new RelativeTime(200,0),
   new RelativeTime(200,0),null);

ReleaseParameters rp  = pt.createReleaseParameters();


pt.addHandler(new AsyncEventHandler
   (null,rp,null,null,null) {
   public void handleAsyncEvent()  {
   System.out.println("Timer went off ");
}
 });
pt.start();  // start the timer
```

Asynchronous Transfer of Control

- Allows interrupting a thread by raising interrupted exceptions
- One thread can throw an exception into another thread
- Better way of notifying the application about the occurrence of a significant event
- Behaves like Thread.stop(deprecated) but is safer
- Can be used as a time-out mechanism
- Asynchronous exception deferred if thread is in synchronized block or uninterruptible method
  - Methods can be made interruptible if *AsynchronouslyInterruptedException* is added to throw clause

Asynchronously Interrupted Exception

- A thread that wants to be interrupted when significant events occur, should mark its methods as throwing *AsynchronouslyInterruptedException*
- The thread would not be interrupted if it is executing a method that is not marked as throwing *AsynchronouslyInterruptedException*
- Triggered when *RealtimeThread.interrupt()* is called

Memory Management

- Definition of memory areas for object allocation
- Heap memory – no real-time
  - Standard Java Heap (one per Virtual Machine)
- Immortal memory – real-time capable
  - Allocated objects exist until the end of the application
- Scoped memory – real-time capable
  - Manual memory management (defined scope)
- Physical memory areas

Scoped Memory

- Activated using the method enter
  - public void enter(Runnable r)
- All allocation in run-method of runnable are done in ScopedMemory
- All objects in Scoped memory will be finalized and collected if :
  - Last real-time thread referencing the scoped exits
- Reference counting of real-time thread using the scope
- Single Parent rule for Scope Stacks
  - No cycles in scope dependencies

Memory Management
Scoped Memory - Types

- VTMemory
  - Allocation may take a variable amount of time
  - Not subject to garbage collection
- LTMemory
  - Not subject to garbage collection
  - Guarantees linear execution time for object allocations from the area
- (CTMemory) in jRate
  - Allocation in constant time

ScopedMemory Example

```
Final ScopedMemory myScope = new VTMemory();
myScope.enter(new Runnable()
{
   public void run()
   {
       …            // all new calls here are
                    // allocated to myScope
   }
}                   // end of run
)                   // end of enter
```

ScopedMemory Example 2

```
final ScopedMemory s = new LTMemory (16,1024);

RealtimeThread t = new RealtimeThread (null,
                       null,
                       new MemoryParameter (s),
                       null,
                       new Runnable ()
                       {
                           public void run ()
                           {
                                // ...
                           }
                       }
```

Embedded Operating Systems

25

Nested Scoped Memory

```
Runnable nestedLogic = new Runnable() {
   public void run() {
      MemoryArea ma2 = new LTMemory(…);
       Runnable moreNestedLogic = new Runnable(){
           public void run() {A a = new A();}
       ma2.enter(moreNestedLogic);
       }};}
};
MemoryArea ma1 = …
ma1.enter(nestedLogic);
```

Embedded Operating Systems

26

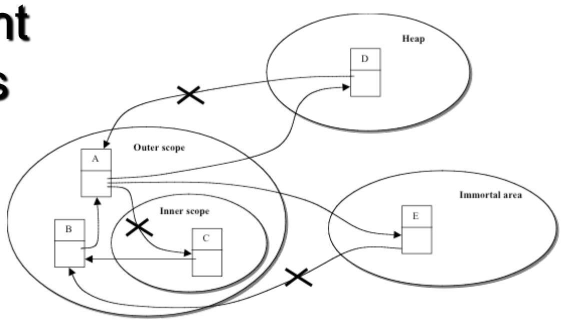
Immortal Memory

- Shared among all threads
- Objects allocated within ImmortalMemory live until the end of the application
  - Objects still exist without any reference to it
- Can be scanned by garbage collector, but not collected itself
- Singleton class
- ImmortalPhysicalMemory

Budgeted allocation

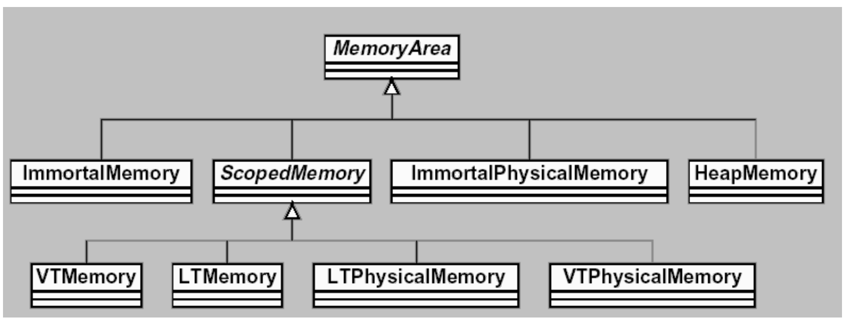
- RTJS provides limited support for memory allocation budgets
- Maximum memory area consumption and maximum allocation rates for real-time threads
- Definition in MemoryParameter of RealTimeThread constructor

Assignment restrictions



	Reference to Heap	Reference to Immortal	Reference to Scoped
Heap	Yes	Yes	No
Immortal	Yes	Yes	No
Scoped	Yes	Yes	Yes, if same, outer, or shared scope
Local Variable	Yes	Yes	Yes, if same, outer, or shared scope

Memory Area - Classes



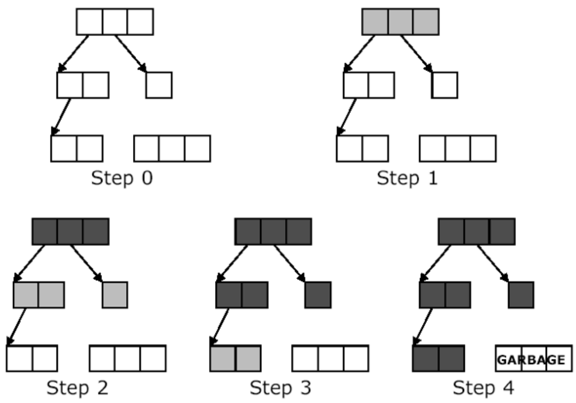
Garbage Collection

- Reference Counting
- Mark-and-Sweep
  - Distinguish live objects from garbage
    - Start in local variable array, operand stack
    - Mark all referenced objects alive
  - Remove all unmarked objects
- Mark-and-Compact
  - Adds de-fragmentation to mark-and-sweep algorithm

Real-time garbage collection

- Fine-grained incremental garbage collection
- Garbage collection should run interleaved with normal threads – not atomic !
- Incremental tracing collectors
  - Objects traversed through as a graph
  - Marking like mark-and-sweep, but using 3 colours (white, grey,black)
- Generational garbage collectors
  - Objects that have been alive for a long time will probably stay for some time more
  - Objects grouped as generations based on creation times

Incremental Collector
Tri-Color Marking



Automatic garbage collection in RTSJ

- "Garbage collector is independent and can be changed"
  - RTSJ does not specify any GC, but gives 2 examples of how GC should be implemented
- "Allow the program to precisely characterize an implemented GC algorithm's effect on the execution time, preemption, and dispatching of real-time Java threads."
  - GC algorithm should be configurable (scanning rates, CPU usage, priorities ..)

Physical Memory Access

- Embedded applications often require direct memory access for
  - Device drivers
  - Memory-mapped I/O
  - Battery-backed RAM
  - Flash memory
- *RawMemoryAccess* contains methods to create/ access a range of physical memory
  - Read-/Write Methods
  - Access based on byte,short, long, float

Synchronization

- Java : *synchronized* keyword
- Communication between NHRT and regular threads needed
- NHRT can not wait for full queues
- Wait free queues
  - Wait-Free-Write-Queue
  - Wait-Free-Read-Queue
  - Wait-Free-Double-Ended-Queue

Priority Inversion

- Default behaviour of synchronized must be: priority inheritance
- RTSJ defines priority ceiling emulation protocol
  - Synchronized segment has a allocated a priority level that indicates the highest possible priority for any thread trying to enter the segment
  - After entering into the segment, the thread's priority is raised to the ceiling value

Handling Posix-Signals

```
public final class POSIXSignalHandler
{
  public static final int SIGABRT;
  public static final int SIGTERM;
  public static final int SIGCANCEL;
  …
public static void addHandler(int signal,
                       AsyncEventHandler handler);
public static void setHandler(int signal,
                       AsyncEventHandler handler);
}
```

Realtime Security

- "System and Options"
- Primarily to check physical memory access
- Check if the application is allowed to set the scheduler

Embedded Operating Systems

39

Realtime System

```
public final class RealtimeSystem
{
  public static final byte BIG_ENDIAN
  public static final byte BYTE_ORDER
  public static final byte LITTLE_ENDIAN
  public static GarbageCollector currentGC
  ()
  public static void setSecurityManager
  ( RealtimeSecurity manager)
}
```

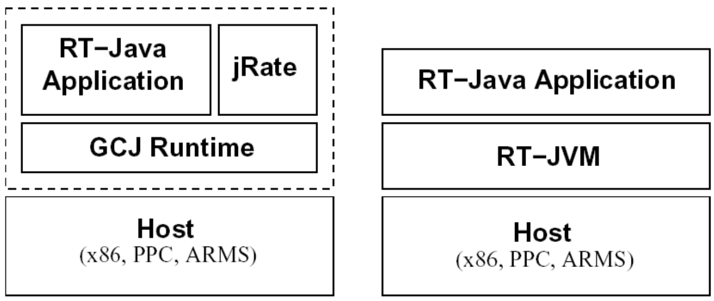
Embedded Operating Systems

40

RTSJ Implementations

- Reference Implementation by TimeSys
  - http://www.rtj.org
  - Based on Timesys Real-Time Linux / x86
- Mackinac: Sparc/x86 running Solaris 10
- Open source implementation, jRate
  - http://tao.doc.wustl.edu/~corsaro/jRate/
  - PhD thesis of Angelo Corsaro
- JamaicaVM - aicas GmbH (Karlsruhe)
- Esmertec Jbed
  - 256 kByte including RTOS
- aJile Systems aj-100
  - Hardware implementation

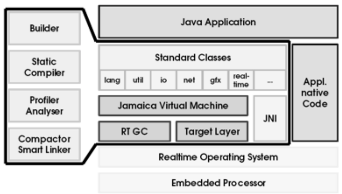
jRate – Overview Precompilation

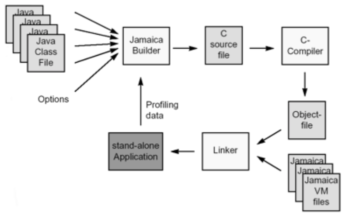


(a) (b)

JamaicaVM – aicas GmbH

- Java bytecode interpreter
- 128 Kbyte minimal footprint
- Real-time garbage collector
- Implements RTSJ exept
  - PriorityCeilingEmulation
  - VTPhysicalMemory
  - LTPhysicalMemory
  - ImmortalPhysicalMemory



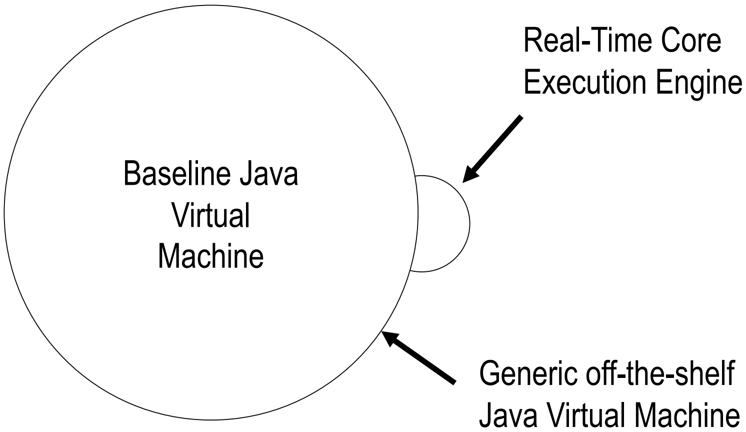


Targets: Sun Solaris, VxWorks, Windows, RTEMS, INTEGRITY

J Consortium

- HP, Aonix, Ericsson, Microsoft, Mitre and NewMonics
- Real-Time Java Working Group
- Core Real-Time Extensions for Java
- Specifies performance like C++
- http://www.j-consortium.org/rtjwg/index.shtml

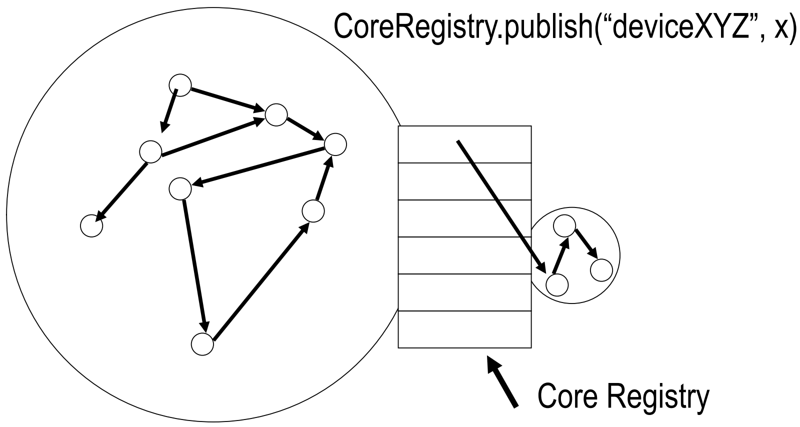
RTJWG View of the Core



Embedded Operating Systems

45

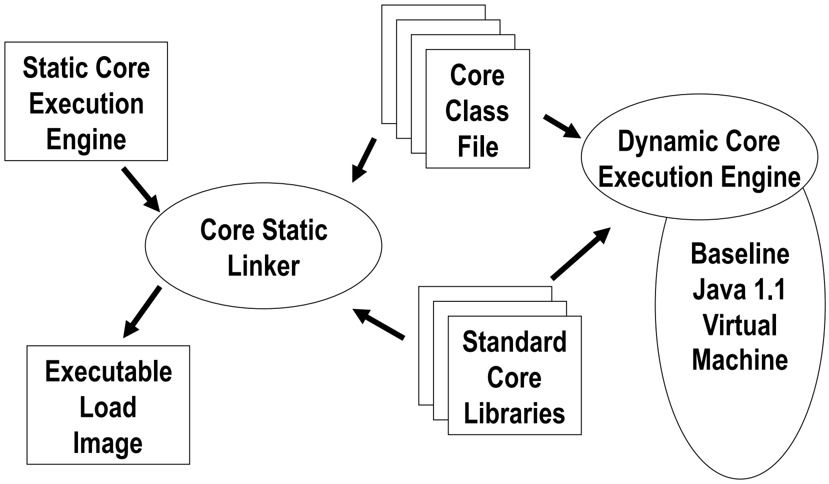
The Core Registry



Embedded Operating Systems

46

Core Deployment



Distributed RTSJ

- State : Java Specification Request
- RMI for Real-Time application communication
  - Predictable end-to-end timeliness
  - Other trans-node properties
- Specification of flow control mechanisms
- “The Distributed Specification for Java – An Initial Proposal”, E. Douglas Jensen
- Following : OMG Dynamic Real-Time CORBA

Literature

- Real-Time Specification for Java
  - http://www.rtj.org/
- The Real-Time Java Platform : Mackinac White Paper
- Seminar on Real Time Linux and Java - Spring 2001
  - http://www.cs.helsinki.fi/u/kraatika/Courses/rt-sem01s.html
- "Real-Time Java Platform Programming", Peter Dribble, Prentice Hall PTR
- Sun Java Real-Time System (Java RTS)
  - http://java.sun.com/javase/technologies/realtime