



# RTSJ Status: Autumn 2007

---

The JSR 282 Expert Group



# RTSJ 1.1

---

- First JSR to update the RTSJ
- Goals
  - Fill some minor gaps in the RTSJ
  - Be backward compatible



# Laundry List

---

Aperiodic threads	Visible CPU time	Schedulable AEs	Blocking factor
Cost enforcement improvement	Processor pinning	AE.fire(obj)	AEHs for resource overflow
Timed.reset	getCurrentSO	Add methods with return ref	Params for enter/executeInArea
Compare relative time to zero	Relax bi-directional rule	New getters for Timer and RTT	Weak scoped refs
Pinned scopes	Melding scopes	Conserve immortal	Physical/Raw memory



# Status

---

- Most of the spec issues have been identified and written up in “Specification Issue” notes
  - These are available on [jcp.org](http://jcp.org)
  - Some scope reduction inevitable
- The RI is currently being updated (by Peter Dibble) to prototype the required changes
- Once experience with this has been obtained, the revision will be finalized



# Reference Implementation

---

- The latest versions of the reference implementation have been alpha releases for RTSJ 1.1
  - Each alpha release has added 1.1 features
- and they have targeted Linux with NPTL (starting with Fedora 6.)
  - This was a surprisingly challenging migration, but Linux has been improving as a real-time platform and the RI can now run correctly on plain Linux.



# Alpha 1.1 Highlights

---

Aperiodic/Sporadic realtime threads	Updated raw memory API
AE.fire(argument)	Access to consumed CPU time for SOs
Pinned scopes	Blocking time for feasibility analysis



# Coming Soon to the Alpha RI

---

- Highlights
  - Cost overrun notification
  - Remove bi-direction reference rule
  - Processor pinning
- (This work may already be done)



# Selected Topics

---

- A few of the RTSJ 1.1 features stand out because they are surprising extensions to RTSJ 1.0, or for the amount of discussion they've engendered.
  - Processor pinning
  - The CPU Time APIs
  - Pinnable scopes
  - The new raw memory API



# RTSJ and Multiprocessor Systems

---

- RTSJ tries to be neutral on shared memory multiprocessor systems
- It is silent on how the VM decides which tasks to execute on which processor



# SMPs and the RTSJ

---

- Given the prevalence of SMPs, the current RTSJ position is
  - not tenable: better schedulability can potentially be obtained by allowing static configuration
  - not scalable to multicore architecture which may be NUMA
- MP issues are practical problems. We can no longer ignore them.



# Problem

---

- As of yet, there has been no standardisation of support for multiprocessor systems in the operating system community
- If RTSJ is being implemented on top of a real-time operating system, it is difficult to know what facilities it can rely on



# Platform (OS) Variability

---

- A program may be allocated the full set of the processors in the system or a subset of them
- The OS may only support global scheduling of threads or it may allow threads to be constrained to one or more processors in the set allocated to the program



# Platform (OS) Variability

---

- The OS may dynamically change the set of processors allocated to a program during the program's execution. If it does this, it is done in a safe manner.
  - Mechanisms may be provided by the OS to inform the application (if the OS supports task to processor allocation) or they may not (if it only supports global scheduling)



# Challenge

---

- To provide a set of mechanisms that can be both expressive enough to support a wide range of application requirements and yet can be implemented (possibly with degraded services) on a wide range of operating systems



# Any solution

---

Should degrade if the program is executing

- on a single processor system
- under an operating system which imposes a global partitioning approach
- under an operating system that does not change the processor set allocated to a program



# Minimal Support

---

- The minimum functionality is for the operating system to allow the VM to determine how many processors are available to it



# API: RealtimeSystems Class

---

```
public static java.util.Bitset  
    availableProcessors ();  
  
public static boolean setAffinitySupported ();  
  
public static boolean  
    affinityChangeNotificationSupported ();  
  
public static AsyncEvent ProcessorRemoved,  
    ProcessorAdded;
```

--



# API: RealtimeSystems Class

---

```
public final static java.util.BitSet setDefaultAffinity(  
    java.util.BitSet Processors)  
    throws ProcessorAffinityException;
```

```
public final static java.util.BitSet setDefaultNoHeapAffinity(  
    java.util.BitSet Processors)  
    throws ProcessorAffinityException;
```

```
public final static java.util.BitSet getDefaultAffinity()  
    throws ProcessorAffinityException;
```

```
public final static java.util.BitSet getDefaultNoHeapAffinity()  
    throws ProcessorAffinityException;
```

Add **new** Exception

```
public class ProcessorAffinityException extends Exception;
```



# API: RealtimeThread/BASEH

---

```
public java.util.Bitset setAffinity(java.util.BitSet Processors)  
    throws ProcessorAffinityException;
```

```
public java.util.BitSet getAffinity();
```



# CPU Time

---

- The EG has extensively debated CPU time APIs.
  - One concern is that we don't know how to characterize the dependability of CPU time measurements
  - We are not entirely happy with any (implementable) approach to cost enforcement



# Solution

---

- We're mainly punting on dependable CPU time measurements. We've added a method that returns something like expected error, but it is not a satisfying solution.
- We are adding a cost overrun notification facility. This feels like a “policy/mechanism separation” that may eventually get us out of the cost enforcement business.



# Pinnable Scopes

---

- Applications now pin scopes by leaving an execution context in the scope. This works, but is not elegant.
- RTSJ 1.1 adds new scoped memory types that have a pinned attribute.
- The need for this facility was not controversial, but the design took much work and discussion.



# Raw Memory

---

- Goal:
  - Make the raw memory access classes into a practical way to access non-memory-mapped devices; e.g., the X86 I/O space.
- Approach:
  - Raw memory accessor factories
- Result:
  - It works, even for raw memory classes added outside `javax.realtime`.



# Summary

---

- The latest versions of the RTSJ RI include Alpha RTSJ 1.1 features.
- Those features are, of course, subject to change.
- In fact, we wish “the community” (that includes you) would try them and give us feedback so we can improve them while they are relatively flexible.



# References

---

- [jcp.org](http://jcp.org) -- JSR-282 pages (only accessible to JCP members)
- [www.rtsj.org](http://www.rtsj.org)
- [www.timesys.com/java](http://www.timesys.com/java) RI download page