JSR-282

RTSJ version 1.1

RTSJ Timeline

RTSJ 1.1

- First JSR to update the RTSJ
- Goals
  - Fill some minor gaps in the RTSJ
  - Try to use a fast spec development cycle
  - 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 memory

Scheduling I

- Improve support for aperiodic threads
  - Currently Sporadic and Aperiodic release parameters only work well with asynchronous event handlers
  - The RealtimeThread class has no general release mechanism
- Version 1.1 will add
  - waitForNextRelease() and release() methods

Scheduling II

- Make per-thread CPU time available
  - Currently, cost monitoring but no way to determine how much CPU time actually used
- Version 1.1 will allows
  - the current, maximum and minimum CPU consumption of a schedulable object (SO) to be determined
  - the current, maximum and minimum CPU consumption of a processing group to be determined

Scheduling III

- Make blocking factor available for feasibility analysis
  - Currently there are no mechanisms that allow the program to give (to the feasibility analysis) the time each schedulable objects spends waiting for locks
- Version 1.1 will
  - Add a blocking time to the ReleaseParameters class

Scheduling IV

- Support processor pinning
  - Currently, the spec tries not to undermine implementations on multiprocessor (SMP) systems but provides no direct support
- Version 1.1 may
  - allow real-time threads and bound asynchronous event handlers to have a processor affinity set

Scheduling V

- Reduce execution eligibility inversion when using asynchronous events that are attached to external happenings
  - Currently, the releasing of ASEHs as a result of: interrupts, signals or the passage of time occur at the highest execution eligibility
- Version 1.1 will
  - allow a scheduling parameter to be associated with an external asynchronous event

Memory Areas I

- Support pinnable scopes
  - Currently there is no way to keep a scope memory area alive unless a SO is active within it
  - This led to the "wedge thread" pattern
- Version 1.1 will
  - restructure the MemoryArea hierarchy so that the main classes implement interfaces
  - The hierarchy will then be extended to allow the concept of scoped memory areas that can be pinned
  - This allows a scope to stays alive even when there are no active SOs within it

Memory Areas II

- Add weak scoped references
  - Currently no equivalent of Java's weak references for scoped memory area
- Version 1.1 is considering
  - whether to allow explicit references between objects that would violate the RTSJ reference rules
  - Assumes the program knows the lifetime of objects
  - JVM is ware of these references and checks safety

Memory Areas III

- Relax bi-directional reference rule
  - Currently it is not permitted for an SO to use a parameter object unless that object can hold a reference to the SO
  - This restricts the parameter object to be in the same scope as the SO which reduces the opportunities for sharing or recycling parameter objects
- Version 1.1 will
  - remove this restriction

Memory Areas IV

- Allow "Melding" scopes
  - Currently, it is necessary to know the size of a scope before it is created
  - This means that components developed independently cannot share the same scope
- Version 1.1 is considering
  - allowing scopes that are adjacent to each other on the scope stack to be melded into a single scope
  - this would allow references between objects in this scopes without violating the RTSJ reference rules

Others

- Pass an object ref from fire() to handleAsyncEvent()
- Allow an ASEH's default memory area to be cleared before each call to handleAsynEvent()
- Conserve immortal memory
- Improve raw/physical memory
- Etc.

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 memory

Progress

- We maintain a document for each project. They are visible on the JSR-282 community update page on jcp.org.
- Most of the projects are ready to try implementation.
- We hope to release an alpha version of RTSJ 1.1 before the end of 2006, and wrap up the JSR in early 2007.

References

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