JSR 302 Public Review
Safety-Critical Java

April 13, 2021

Doug Locke, Ph.D. (Specification Lead)
representing The Open Group

Agenda

- Background
- Expert Group
- Brief Technical Overview
- Publicity, Collaboration, Participation, and Transparency
- Intellectual Property

Background – What is a Safety-Critical System?

- What is a Safety-Critical System (SCS)?
  - Any system that MUST have an extreme level of reliability
  - An SCS failure may result in loss of life or property
  - An SCS is subject to formal certification (e.g., DO-178C)
  - Formal certification is very expensive (ca $60-80/SLOC)

Background – History of SCSs

- Originally, SCSs were rare, small, and simple
  - E.g, aircraft autopilot (ca 1975)
  - SCSs now found in increasing numbers and complexity
    - Aircraft, spacecraft, air traffic control, automotive, rapid transit, medical devices, power generation and transmission, industrial controls, military vehicles, UAVs, weapons, etc.
  - Until about 1980, all SCSs written in Assembly
  - 1980 – present, most SCSs written in C
  - 1995 – present, subset of Ada also used (Ravenscar profile or SPARK)
  - No dynamic memory allocation in SCSs until recently
  - No OO SCSs until 2012

Background – JSR-302

- SCSs represent a new technology domain for Java
  - Application code must be as simple as possible
  - Certification required for both application and infrastructure
  - Almost all SCSs have “hard real-time” characteristics
  - Provably correct memory management is critical
- Around 2004, The Open Group (TOG) started a High Assurance Software initiative
  - TOG is a consortium of about 400 companies, government agencies, and other consortia creating open standards
    - For example, TOG manages the Single Unix Specification (SUS) that governs all UNIX implementations
  - Members wanted a modern, robust language for use in such S/W
  - Therefore, TOG started an effort for Safety-Critical Java
  - JSR 302 was approved in 2006.

Java Community Process

Business/Marketing/Ecosystem Justification

- Why do this JSR?
  - Permit SCSs to exploit major Java strengths for safety, reliability, portability
- What's the need?
  - Existing SCSs are overly expensive, and difficult to certify
  - They tend to duplicate infrastructure capabilities (e.g., drivers, memory management, scheduling)
- How does it fit in to the Java ecosystem?
  - Built upon the RTSJ (JSR 1, JSR 282) to maintain compliance with J2SE – currently requires Java 8.
- Is the idea ready for standardization?
  - Yes. Multiple organizations in TOG are looking for this.

Expert Group

- The EG has consisted of the following members:
  - Industrial: aicas, IBM, Boeing, Rockwell Collins, Siemens, DDC-I
  - Academic: Andy Wellings, Martin Schoeberl
  - Others: Ben Brosgol, Scott Anderson, Joyce Tokar
- The EG has met weekly over it’s lifetime by teleconference (currently uses Zoom)
- The EG communicates internally using e-mail, and via a shared SVN repository

Java Community Process

Brief Technical Overview (1 of 3)

- Introduces three Compliance Levels (Level Zero, One, and Two)
  - Higher levels permit more complex applications
  - Higher levels require more expensive infrastructure
- Introduces Mission concept
  - Application consists of one or more Missions
  - Missions can be sequenced arbitrarily
  - At Level Two, multiple Missions are possible simultaneously
- Mission consists of
  - Non-GC memory area (however, GC not prohibited)
  - One or more Schedulable Objects (from RTSJ)
- RTSJ-subset memory management (e.g., can't share private memory across Schedulable Objects)

Brief Technical Overview (2 of 3)

- Simple I/O using JME Connectors and Connections
  - No file management)
- Supports RTSJ Interrupt Service Routines
- Supports RTSJ Raw Memory (e.g., DMA, memory-mapped I/O)
- Supports RTSJ Clocks and Timers, including user-defined clocks
- Simple JNI support
  - Limited reflection
  - Specification defines supported JNI interfaces
- Exception support is a subset of RTSJ

Brief Technical Overview (3 of 3)

- Specific Java SCJ Annotations are required
  - E.g., SCJAllowed(level) means that a method is allowed for an SCJ application at Level "level" or below, and that it is executable on any SCJ infrastructure supporting Level "level" or above.
- Specification defines SCJ-supported Java library classes and methods from
  - java.io
  - java.lang
  - java.microedition.io
  - javax.realtime
  - javax.realtime.memory
  - javax.realtime.device
  - javax.safetycritical
  - javax.safetycritical.annotate
  - javax.safetycritical.io

Publicity

- Open Group Real-Time and Embedded Forum
  - regular updates have been presented at TOG meetings
- Java Technology for Real-time and Embedded Systems (JTRES)
  - More than 100 papers have been peer-reviewed, published, and presented on SCJ topics
- SCJ Presented at the 2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems
  - *Java for Safety-Critical Applications*, Proceedings of SafeCert 2009, York, UK, 2009

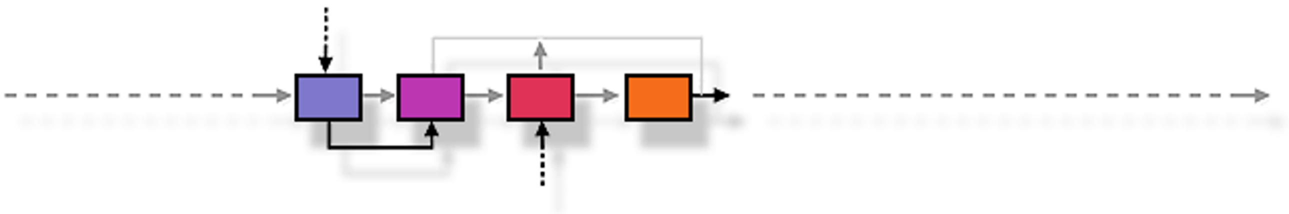
Collaboration with other community groups

- We have been continuously collaborating with JSR 282 to ensure compatibility between the specifications.
  - Issues forwarded to JSR 282 EG
  - JSR 282 updates then returned to the JSR 302 EG
  - Accommodations regularly made to ensure that SCJ is implementable on an RTSJ (JSR 302) base
- Several EG members are also JSR 282 members
- We also collaborate with the Open Group Realtime and Embedded Forum.

IP flow

- The SCJ Specification uses an open license:
  - https://www.jcp.org/aboutJava/communityprocess/licenses/jsr302/JSR-302SpecificationLicense.txt
- The SCJ RI and TCK use an open source license:
  - https://www.jcp.org/aboutJava/communityprocess/licenses/jsr302/302RILicense.txt
- We have received a number of comments and contributions from outside the JCP. The EG has reviewed all contributions and incorporated them when possible.
- All collaboration tools are open source
- We do not currently have a contributor agreement
- We are not aware of any legal concerns

Java Community Process



Thank you!