JSR 302 Review
Safety-Critical Java

January 13, 2014

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

• Goals

• Information to be gathered
  – Background
  – Technical Scope
  – Expert Group
  – Deliverables: Specification, RI, TCK, IP
  – Schedule
  – Publicity, Collaboration, Participation, and Transparency

• Implementation notes

• Issues

• Questions, discussion, next steps
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 $40-50/SLOC)
• 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 – 1995, most SCSs written in C
  – 1995 – present, subset of Ada used (Ravenscar profile or SPARK)
  – No dynamic memory allocation in SCSs until recently
  – No OO SCSs until 2012
• 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
  – 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.
• 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 RTSJ (JSR-1, JSR-282) to maintain compliance with J2ME and J2EE – currently requires Java 7.

• Is the idea ready for standardization?
  – Yes. Multiple organizations in TOG are looking for this.
• The EG consists of the following members:
  – Industrial: aicas, IBM, Atego, Boeing, Rockwell Collins, Siemens, DDC-I
  – Academic: Andy Wellings, Martin Schoeberl, Anders Ravn
  – Others: Ben Bros gol, Scott Anderson, Joyce Tokar
• The EG currently meets weekly by teleconference
• The EG communicates internally with e-mail, and via an SVN repository
• 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
  – 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)

- 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 subset of RTSJ
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 all SCJ-supported Java library classes and methods from
- java.io
- java.lang
- java.microedition.io
- javax.realtime
- javax.realtime.device
- javax.safetycritical
- javax.safetycritical.annotate
- javax.safetycritical.io
History

• Initial work on SCJ started in The Open Group in 2003
• JSR-302 was approved in July 2006
• First Early Draft Review started 7 January, 2011
  – Completed 7 April, 2011
• Second Early Draft Review started 28 June, 2013
  – Completed 26 September, 2013
• Besides EDR releases, the EG has released many drafts via TOG meetings and academic postings.
  – E.g., https://github.com/scj-devel/doc which is maintained by researchers at the Technical University of Denmark
Other Deliverables

- Other than the Specification, RI, and TCK, the EG does not currently plan other deliverables
- However, the SCJ Specification (currently at 344 pages not counting Javadoc Appendices) provides
  - Detailed semantics descriptions
  - Computational model descriptions
  - Sample application code
  - Rationale for key capabilities and limitations
Publicity

• Open Group Real-Time and Embedded Forum
  – regular updates presented
  – E.g., http://www.opengroup.org/sanfrancisco2014/rtes

• Java Technology for Real-time and Embedded Systems (JTRES)
  – Annual conference dedicated to RTSJ and SCJ issues
  – Has met every year since 2003
  – More than 100 papers published and presented on SCJ topics
  – See jtres2014.compute.dtu.dk/ for 2014 Niagara Falls conference information
Collaboration with other community groups

• We are collaborating with JSR-282 to ensure maximal compatibility between the specifications.
  – Issues forwarded to JSR-282 EG
  – JSR-282 updates then returned to the EG
  – Accommodations regularly made to ensure that SCJ is implementable on an RTSJ base
• Three EG members are also JSR-282 members
• We also collaborate with the Open Group Realtime and Embedded Forum.
Implementations

- The RI is being developed by aicas, GmbH
- An SCJ implementation is currently available from
  - Aalborg University and VIA University College, Denmark
  - http://www.icelab.dk/index.html
- Another SCJ implementation is being created at the Technical University of Denmark
  - http://cj4es.imm.dtu.dk/
## Schedule

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification Draft Complete</td>
<td>1 Mar. 2015</td>
</tr>
<tr>
<td>Final Draft Review Start</td>
<td>1 Apr. 2015</td>
</tr>
<tr>
<td>First RI Release</td>
<td>1 May. 2015</td>
</tr>
<tr>
<td>First TCK Release</td>
<td>1 Jun. 2015</td>
</tr>
</tbody>
</table>
• 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 reviews all contributions and incorporates 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
The JSR-302 RI is being developed by aicas GmbH

The JSR-302 TCK is being developed by Aalborg University and VIA University College, Denmark (from SCJ formal semantics)

Neither of these are from the spec-lead organization.

The RI and TCK will be available for public download

We plan to provide a public source code repository for the RI and TCK
  – We will plan for this before completion of JSR-302
Participation and transparency

• JSR-302 is managed by The Open Group
  – TOG manages many open standards
  – E.g., TOG manages the POSIX standard
• The JCP page is https://www.jcp.org/en/jsr/detail?id=302
• The SCJ project website is http://douglocke.com/SCJ
Adopt-a-JSR

• This is a new requirement that we have not been tracking.
• What do we have to do?
Mailing lists or forums

- This is a new capability for us
- Twitter hashtag: #SCJava
- Spec lead has tweeted several messages on SCJ
- We expect increased message traffic as we near completion
- Pointers to twitter and project website added to SCJ-302 page on jcp.org as well as in twitter messages.
- JTRES community regularly updated on SCJ progress
Issue tracker

• This project is nearly complete
• The project started with 17 major issues
• All major issues have been closed
• At present, 8 minor issues are open, expected to be closed in January
• On average ca 12 issues opened and closed each month.
• Spec Lead keeps issues list, sends to EG list regularly
• Non-EG issues become EG issues immediately, are then reported back to originator by Spec Lead
Document archive

• Public documents consist of public presentations and draft specifications
  – Presentations made at TOG meetings quarterly
  – Draft specifications published on GitHub
  – GitHub also used by community researchers

• Private repository (SVN) used for EG document archives
  – Specification using LaTex, with Javadoc tools
  – Any EG member can build the spec
Other transparency and participation metrics

• The Safety-Critical Java community is an active subset of the real-time Java community
• This community consists of practitioners and researchers involved in safety-critical and Java technology
• There have been over 100 papers published on SCJ
Implementation notes

- Methodologies for creating certifiable infrastructures for safety-critical systems are known
  - Very expensive
  - Each mechanism must be certifiably correct
  - Large number of documented “artifacts” must be created for certification
  - Both implementation and application must be certifiable
  - Rigorous development methodologies and testing must be followed
Questions, discussion, next steps
Thank you!
http://jcp.org