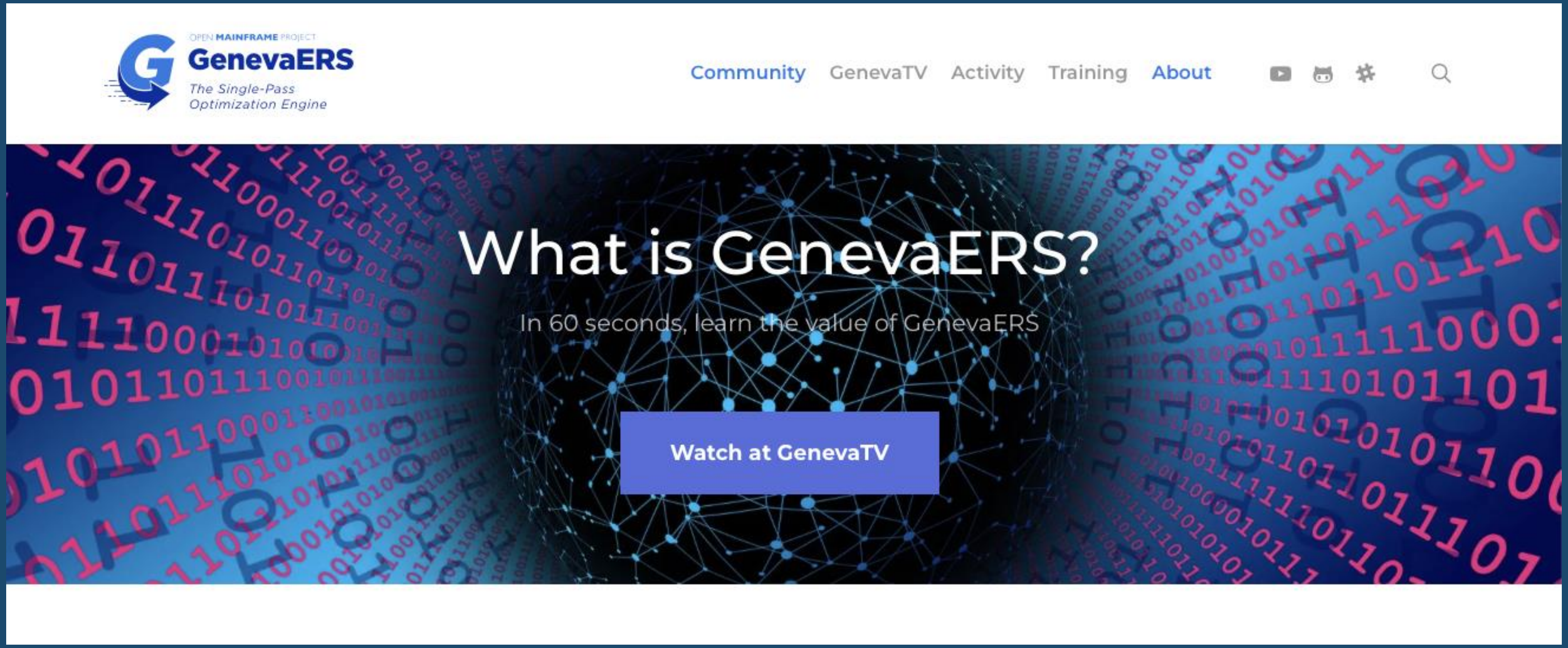
GenevaERS Annual Review
z/OS Java Inter-Language R&D Results

By Neil Beesley and Kip M. Twitchell

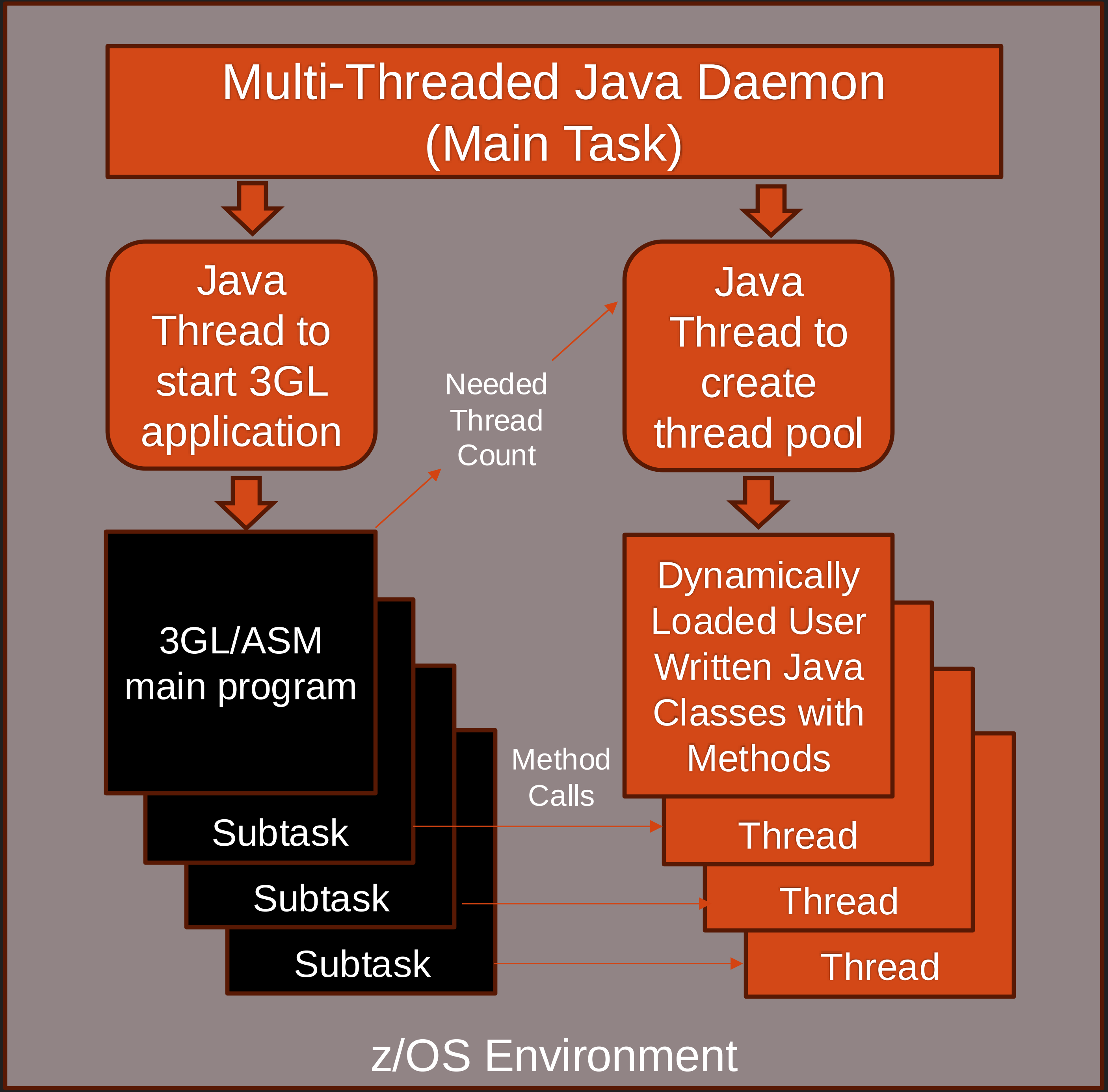
September 26, 2024



GenevaERS R&D Inter-language Framework POC

Enables Java to be called from z/OS ASM or 3GL applications

- JVM is started only once
- Supported applications:
  - Single
  - Multi-threaded
- Supports languages:
  - Assembler
  - C++
  - COBOL
  - PL/1
- Success criteria:
  - Prove Viability
  - Run-time Efficiency


Multi-Threaded Java Daemon (Main Task)
Java Thread to start 3GL application
Java Thread to create thread pool
Needed Thread Count
3GL/ASM main program
Subtask
Subtask
Subtask
Method Calls
Dynamically Loaded User Written Java Classes with Methods
Thread
Thread
Thread
z/OS Environment

Execution Environment and Code Sample

OPEN MAINFRAME PROJECT GenevaERS The Single-Pass Optimization Engine

Sample JCL

Batch job that executes Java daemon (JVM starts)

```
/JAVA EXEC PROC=JVMPRC16,
// JAVACLS='GvbJavaDaemon -D'
//STDENV DD *
LIBPATH="$LIBPATH":/safr/mf_build/lib
export LIBPATH="$LIBPATH":
for i in $(find $APP_HOME -type f);do
    CLASSPATH="$CLASSPATH":"$i"
    done
export CLASSPATH="$CLASSPATH":
# Set JZOS specific options
IJO="-Xms64m -Xmx1g"
IJO="$IJO -Dfile.encoding=ISO8859-1"
export IBM_JAVA_OPTIONS="$IJO ”
//*
//DDEXEC   DD *
PGM=TSTUR70,PARM='TASKS=20,NCALL=20000’
//*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
```

Java daemon starts ASM/3GL application, “DDEXEC” statement

Sample Code

ASM/3GL application performs INIT call telling Daemon how many Java threads are needed (one per sub-task for ASM/3GL application)

```
MVC    UR70FUN,=CL8'INIT'   Set number of threads
LH     R0,WKTASKS           Number of subtasks
STH    R0,UR70OPNT          Number of threads needed
LAY    R1,UR70LIST
L      R15,WKUR70A
BASR   R14,R15
```

ASM/3GL application performs “method” calls which are dynamically loaded and executed in the Java thread pool

```
MVC    UR70FUN,=CL8'CALL’
MVC    UR70CLSS,=CL32'MyClass’
LAY    R1,UR70LIST
L      R15,WKUR70A
BASR   R14,R15
```

When ASM/3GL application finishes the Java daemon also terminates, end of job

Exchanging Data Between Java and Application

OPEN MAINFRAME PROJECT GenevaERS *The Single-Pass Optimization Engine*

Data can be passed between ASM/3GL application and Java in either be 31-bit or 64-bit addresses

The following code set the length of send buffer and max length of receive buffer

```
MVC    UR70FUN,=CL8'CALL’
MVC    UR70CLSS,=CL32'MyClass’
MVC    UR70METH,=cl32'Method1’
MVC    UR70LSND,SNDLEN
MVC    UR70LRCV,RECLEN
LAY    R1,UR70LIST
L      R15,WKUR70A
BASR   R14,R15
```

A typical Java user written method would be

```
Public ReturnData MyMethod(byte[] bytesin) {
…
ReturnData returnData = new ReturnData(rc, bytesout);
Return returnData;
}
```

```
ASCII -> Java: new String(bytes, "ASCII")
EBCDIC -> Java: new String(bytes, "Cp1047")
Java -> ASCII: string.getBytes("ASCII")
Java -> EBCDIC: string.getBytes("Cp1047")
```

- Single argument supplied of type byte[], which is the data send by ASM/3GL application. Bytesin.length determined by send length
- Object of type ReturnData is returned comprising a return code and variable array of bytes to ASM/3GL program
- Sent and received data is by arrays in native ZOS format
- Packed, Binary full words and half words, Zoned numeric and EBCDIC and sent/received by ASM/3GL application
- User written Java method must know data to expect, so it can interpret input byte array into Java data types using JZOS (get functions)
- Before returning it must format the output byte array into native ZOS format using JZOS (put functions)
- ASCII/EBCDIC is handled using String() with encoding identifier

Performance Engine Java Subsystem: Measurements and APA

OPEN MAINFRAME PROJECT
GenevaERS
The Single-Pass Optimization Engine

Using GenevaERS as our specific application, we found the following performance characteristics.

The initial results were:

Scenario	Event records	Lookups	CPU	Elapsed
Bare ASM Exit (no JVM)	76580	76580	0.02	0.01
ASM Exit not calling Java (with JVM)	76580	76580	0.34	1.05
ASM Exit calling Java (with JVM)	76580	76580	1.47	2.21
JVM Start-up cost			0.32	1.04
Cost for calling Java 10,000 times (minus start-up)			0.15	0.15

After analysis, we made changes to avoid using Token Services each call with better results

Scenario	Event records	Lookups	CPU	Elapsed
Bare ASM Exit (no JVM)	76580	76580	0.02	0.01
ASM Exit not calling Java (with JVM)	76580	76580	0.36	1.05
ASM Exit calling Java (with JVM)	76580	76580	1.18	2.05
JVM Start-up cost			0.34	1.04
Cost for calling Java 10,000 times (minus start-up)			0.11	0.13

Conclusions

OPEN MAINFRAME PROJECT GenevaERS The Single-Pass Optimization Engine

We attempted to use the following features:

- z/OS environment
- Multi-threading
- Multi-language
- With JVM started only once
- Primarily focused on Run-time Efficiency

- Results of Efforts
  - Multiple levels of interfacing possible
  - Data transmission between applications is possible
  - Use of jzos is integral to success of the efforts
  - Although of course ASM-Java was less efficient than just ASM, the overhead was surprisingly low
- Area of further research includes performance implications of ASCII to EBCDIC conversion processes for a high-performance engine like GenevaERS