Java implementation of Channel Access (CAJ)

Matej _ekoranja  (presented by Mark Ple_ko)
Funded by DLS (M.Heron) and DESY (M.Clausen)

EPICS Meeting – Japan, 2004
What is CAJ?

- **Channel Access in Java (CAJ)** is a CA client library written completely in Java.
- It “plugs” into JCA 2 interfaces.
- Written as a result of detailed analysis of existing CA library to provide better *stability* and *performance* opposed to the current JCA JNI implementation.
- Since it was written from scratch, code is clean, follows OO design and uses lots of design patterns.
- No problems with native libraries (no recompilation needed).
Achieving stability

• The main reason CAJ was written is stability - JCA JNI was not hard to crash with our ControlDesk application for the DLS (extensive concurrent connect, monitor creation and value retrieval)

• Profound testing during the whole development cycle
  • ~ 90% of code coverage!

• Code simplicity helped a lot (simple code leads to less bugs)

• ‘TCP Reno’-like UDP congestion control

• Until now “only” 3 CAJ bugs were discovered
  • 2 by Ken Evans
Achieving performance

• Latest concurrent, network communication design patterns used to implement efficient event demultiplexing, minimize context switching and maximize CPU cache affinity (Leader/Followers design pattern used)

• Asynchronous I/O used (Java NIO package)
  • new epoll-based selector supported, which is improved select(); available in the latest Linux 2.6 kernel

• (Some performance measurements will be shown later)

• Due to OO design light CAJ version is possible (one communication thread), convenient for light CA clients (handhelds)
Immediate JCA JNI to CAJ migration

Simply change (example):

jca.createContext(JCALibraryJNI_SINGLE_THREADED);

OR

jca createContext(JCALibraryJNI_THREADSAFE);

jca createContext(JCALibraryCHANNEL_ACCESS_JAVA);

... and take care of configuration.

Note that CAJ can not use system environment variables like EPICS_CA_ADDR_LIST (not available in Java 1.4, but available again in Java 1.5).
OO Usage of JCA
Performance measurements

- Client on the same host as server, Pentium IV 1.6GHz, 1GB RAM, Red Hat 9
- "no bulk" means calling flushIO() after each get request

Note: this is only synthetic performance test and doesn’t reflect performance in practice!
Comparing to C Version of CA

• Completely different approaches:
  • C pointer versus Java object creation
  • Java is clean, C is dirty but quick 😊

• Based on CA 4.11, should be compatible down to 4.0
  • But has not been tested

• Backward compatibility might be an issue
  • several undocumented features in the C version, might have missed one
User Experiences

• Control Desk
  • was limited to some 300 PVs before

• JoiMint
  • just got ported to JCA 2 so it can use CAJ

• JProbe (Ken Evans ported it to JCA 2).
  • “I have tested it on all the dbt types. Like my performance test, it is an application that requires a large subset of the features provided by CA. I think CAJ is looking good.”
  • “Moreover, my test program, which accesses 100 PVs updating at 10 Hz, worked *much* better. Before, there were dropouts. Now, it seems to keep up. Cool!”
Conclusions

• Needs some time (production usage) to confirm maturity
• JCA still has much room for performance improvements, now that JNI isn’t the bottleneck anymore
• Open possibilities for more user friendly applications based on Java
  • develop a CAJ server to integrate other Java Applications?
• Proves that
  • other-than legacy CA implementation can be done …
  • CA documentation is useful and useable and that CA is not something mysterious
• Cosylab needs a new task (and funding 😊)