Transition of GM/CA-CAT data acquisition software from BluIce to JBlulce

Mark Hilgart

GM/CA CAT at the Advanced Photon Source
Biosciences Division of Argonne National Laboratory
USA
Control Software Requirements

- GM/CA is a macromolecular crystallography beamline
- Our users require:
  - Familiar interface
    - Crystallographers are highly mobile
    - Need to use software at different beamlines with little training
  - Rapid, reliable feature development
    - Crystallographers have many choices of beamlines
    - Advances from other beamlines must be incorporated quickly
    - New features must be released quickly before others do – yet must be high quality
History of JBlulce

Blulce Architecture (SSRL)  
- SSRL Blulce: 1997  
- GM/CA replaced SSRL Blulce backend: 2003-2005  
- GM/CA developed JBlulce: 2008-2010

Blulce-EPICS Architecture

JBlulce architecture
JBlulce-EPICS: 100% Java in June 2010
What changed from BluIce to JBlulce?

- Multiple languages -> single language
- C++ -> Java
- Architecture changes
- Features
Single language: Advantages

- Easier debugging
  - Full stack trace is always available
  - Step-through debugging always works
- Higher reliability
  - Thanks to easier debugging and JVM protection from memory errors
- Faster development
  - No protocol layer means changes are much easier
What changed from BluIce to JBlulce?

- Multiple languages -> single language
- C++ -> Java
- Architecture changes
- Features
Java: JVM prevents memory errors

- Java removes uncertainty about memory corruption
  - Tools like valgrind and electric fence, and classes like checked pointers, are not required
- Tracking down the sources of crashes was a major issue in 2007-2008
What changed from BluIce to JBluIce?

- Multiple languages -> single language
- C++ -> Java
- **Architecture changes**
- Features
Architecture: Thread-based actions

- BluIce-EPICS had some threaded commands, but now all are converted
- Single 1-2 page methods tell the whole story of an action
  - In a thread, the line number tells you a lot about what’s happening
  - In a callback-based class, which timers are running, and which callback will be called next?
- Perl scripts are written this way, so porting methods (and keeping in sync) is straightforward
Architecture: Separation of logic, GUI & devices

- SSRL Blulce mixes logic in GUI widgets
- JBlulce usually makes one-line calls to model objects
- Only devices talk to EPICS (except for temporary calls)
**Architecture: 3x LOC reduction**

- JBlulce has more (relevant) functionality in less LOC than just the Bluice front-end
- Java libraries largely replace Stanford libraries
  - Sockets
  - HTTP
  - Math
- Simplification
  - Replaced sockets with simple HTTP
  - DCS abstraction layer

### LOC Breakdown

<table>
<thead>
<tr>
<th>Stanford C++ Libraries</th>
<th>69</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ DCS</td>
<td>33</td>
</tr>
<tr>
<td>Tcl Widgets</td>
<td>8</td>
</tr>
<tr>
<td>Tcl Base</td>
<td>46</td>
</tr>
<tr>
<td>Bluice</td>
<td></td>
</tr>
<tr>
<td>JBlulce</td>
<td></td>
</tr>
<tr>
<td>Java Other</td>
<td>14</td>
</tr>
<tr>
<td>Java Models</td>
<td>3</td>
</tr>
<tr>
<td>Java Widgets</td>
<td>18</td>
</tr>
<tr>
<td>Java Devices</td>
<td>5</td>
</tr>
<tr>
<td>Java Devices</td>
<td></td>
</tr>
<tr>
<td>Java Models</td>
<td></td>
</tr>
<tr>
<td>Java Other</td>
<td></td>
</tr>
<tr>
<td>Stanford C++ Libraries</td>
<td></td>
</tr>
<tr>
<td>C++ DCS</td>
<td></td>
</tr>
<tr>
<td>Tcl Widgets</td>
<td></td>
</tr>
<tr>
<td>Tcl Base</td>
<td></td>
</tr>
<tr>
<td>Bluice</td>
<td></td>
</tr>
<tr>
<td>JBlulce</td>
<td></td>
</tr>
<tr>
<td>Java Other</td>
<td></td>
</tr>
<tr>
<td>Java Models</td>
<td></td>
</tr>
<tr>
<td>Java Devices</td>
<td></td>
</tr>
<tr>
<td>Java Other</td>
<td></td>
</tr>
<tr>
<td>Stanford C++ Libraries</td>
<td></td>
</tr>
<tr>
<td>C++ DCS</td>
<td></td>
</tr>
<tr>
<td>Tcl Widgets</td>
<td></td>
</tr>
<tr>
<td>Tcl Base</td>
<td></td>
</tr>
<tr>
<td>Bluice</td>
<td></td>
</tr>
<tr>
<td>JBlulce</td>
<td></td>
</tr>
<tr>
<td>Java Other</td>
<td></td>
</tr>
<tr>
<td>Java Models</td>
<td></td>
</tr>
<tr>
<td>Java Devices</td>
<td></td>
</tr>
<tr>
<td>Java Other</td>
<td></td>
</tr>
</tbody>
</table>

(*) = kLOC
What changed from BluIce to JBlulce?

- Multiple languages -> single language
- C++ -> Java
- Architecture changes
- Features
Features: Raster

- 2008-2: Initial release
- 2009-2: Fl. raster
- 2010-1: Multiple run tabs
- Two uses
  - Find invisible crystals
  - Find the best diffracting areas of large crystals
- Diffraction vs. fluorescence
  - Diffraction is always relevant
  - Fluorescence is 4x faster and causes less radiation damage
Features: Vector collect

- Data collection along an arbitrary 3D vector
  - To define, center endpoints and click “set”
- Uses
  - Collect along rod-shaped crystals
  - Raster crystals that don’t fit the rigid raster tab grid
- Visualization over live video
- Parameters for:
  - Frames per site
  - Site spacing
  - Overlap
Features: Integrated motion tracking

Tracking output

[14.530] estimateMoveTime(): motionTime=2.1s accelTime=200.00s accelCurve=100.00s ...
[14.531] elapsed= 0.0s inPos=1 runPrg=0 actPos=-0.0000 ...
[14.633] elapsed= 0.1s inPos=1 runPrg=0 actPos=0.0281 ...
...
[15.959] elapsed= 1.4s inPos=0 runPrg=1 actPos=87.2386 ...
[16.061] elapsed= 1.5s inPos=0 runPrg=1 actPos=89.9528 ...
[16.163] elapsed= 1.6s inPos=0 runPrg=0 actPos=89.9997 ...
[16.265] elapsed= 1.7s inPos=1 runPrg=0 actPos=89.9998 ...
[16.367] elapsed= 1.8s inPos=1 runPrg=0 actPos=89.9998 ...
[16.469] elapsed= 1.9s inPos=1 runPrg=0 actPos=89.9998 ...
[16.478] Tracking ended. Real in-motion time for motor gonio_omega was 1.9s (92.3% of est.)

- Tracking guarantees motors are in position, and handles errors and timeouts
- In Tcl/C++, tracking was added in the background
  - DCS protocol didn’t support tracking
  - Tracking was done in the background but could be ignored
- In Java, tracking determines when a move finishes
  - Moves always complete, by timeout if necessary
  - Move results are more informative because they incorporate tracking
Features: Editable spreadsheet and Weblce integration

- New features built in to the initial converted screening tab:
  - Spreadsheet can be edited and saved without using OpenOffice or Excel
  - Screening results are sent automatically to Weblce
- 2010-1: Weblce scores can be loaded back in to Blulce
Features: 5x startup time reduction

Blulce: 16 seconds

JBlulce: 2.8 seconds

- Tests were performed on the same computer
- Improvement due mostly to connecting PVs in parallel at startup
  - PV requests are sent at once
  - Callbacks are sent as PVs connect
Tcl/Java integration

- Raster was an initial test
- 1-3 tabs were converted per run
- TkXext embedded Java windows in Tcl
- Named pipes were used for sending commands between processes
Release

- 2.5 years of development
  - Raster: 2008-2
  - Fl. raster: 2009-2
  - Screening: 2009-3
  - Collect: 2010-1
  - Hutch, Sample, Scan: 2010-2

- Testing
  - Fully functional 4 weeks before users
  - Change cutoff 2 weeks before users
  - Crystallographers tested daily for the last 4 weeks

- Smooth release so far
  - No serious bugs found in 1.5 weeks of users
Conclusion

- Users’ demands guide development
  - Familiar interface
    - Throughout development, interface is kept the same
  - Rapid, reliable feature development
    - Architecture has been streamlined to enable fast development and high reliability

- Future plans: concentrate on features