Tools and Services at NSLSII

Kunal Shroff, Tasha Summers, Smith Reid, Gabriele Carcassi, Michael Davidsaver (NSLSII)
Ralph Lange (ITER)
Samuel Dallstream (UMich)
Michael Skinner (Marist College)
CS-Studio

• An Integrated platform for controls and physics applications
  • Over 2 dozen releases
  • In production 3.1.6, 3.2.15a, and 3.3.10a
CS-Studio Applications

- BOY
- BEAST
- LogViewer
- DataBrowser
- Pretune
- ShiftViewer
BOY: Best OPI Yet

• Approximately 5000 opi screens
• All hosted in a mercurial repository
• NFS mounted onto all machines in the controls network
BOY Screens

• Engineering:
  Developed by the controls engineers / the developers of the device IOC
  Designed for device control

• Physics:
  Do the real useful stuff..
  Created from a list of required interfaces and discussed in AP group meetings
  Lots of rules, calls to external python scripts
  Task oriented screens

• Operator:
  Combine/Summarize a lot of information
Issues
Interesting Experiences
BOY Screen standards:

- A perfect standards are hard to define
- Following standards adds overhead

Source: http://xkcd.com/927/
Absence of BOY Screen standards:
Solution*

- Guidelines are easier than standards
- Good examples more effective than good standards

* contingent on cooperation between physicist, engineers and operators
Good and Bad Practices:

• PV naming convention + Macros + Linking containers
• Reusable pieces
• Easily modified
Good and Bad Practices:

- PV naming convention + Macros + Linking containers + Rules
  - 2000 + pvs with over 800 rules

- Solution:
  - Multistate LED + formulas
Good and Bad Practices:

• Avoid Scripts/Rules unless absolutely necessary
  • Should this be implemented in an IOC?
  • Is there a PvManager formula I can use?
  • Can I use rules? (Have I allocated a large enough PermGen space)
  • Use a script
Data Connection Layer:

• 3.1.x
  • Use PVManager to address threading issues, source rate throttling

• 3.2.x
  • PVManager with formula functions to provide an alternative to rules and scripts
  • Graphene prototype for displaying large waveforms

• 3.3.x
  • Passive scanning: switch from active polling of the queue to notification model
  • More graphene plots
SWT:

• A limited widget set
  • Using native widget results in the lowest common denominator
  • Poor performance on Linux machines

• JavaFX to the rescue
  • Part of jdk 8
  • Richer and easier to use widget set
    • TreeTable, Table with embedded controls
  • JavaFX and SWT share the UI thread
    • Can be easily embedded into SWT/Jface composites/views
JavaFX in eclipse

- Share the same UI thread
- Relatively easy to embed
- Require e(fx)clipse
JavaFX preliminary comparisons

• 1000 label widgets updating at 1Hz
Motivation and Objectives

• A flat name space restricts seriously:
  • Clients need to know all channel names beforehand
  • Portable generic clients must be simple
  • Apps need full configuration or framework supplied service

• Develop a Directory Service
  • Generic
    • No dependency on installation and local conventions
  • Simple and fast (enough)
    • Use standards wherever possible
  • Provides "query-by-functionality"
Directory Data

- Set of **Channels** (unique names)
- Each Channel has an arbitrary number of **Properties** (name/value pairs) and **Tags** (names)
- Each Channel, Property, or Tag has an **Owner** (group) to allow basic access control
- All names and values are strings
Directory Data example

<channel name="V:1-SR-BI:SUPER{BPM:4}SA:X" owner="cf-update">
  <properties>
    <property name="elemName" value="bpm:4" owner="cf-update"/>
    <property name="elemIndex" value="400" owner="cf-update"/>
    <property name="elemPosition" value="5.208" owner="cf-update"/>
    <property name="pvStatus" value="Active" owner="cf-update"/>
  </properties>
  <tags>
    <tag name="aphla.sys.SR" owner="operator"/>
  </tags>
</channel>
Query Example

• **SR:** *C01*  
  All pvs from storage ring cell 1

• **SR:** *C01** & **elemName=bp**: 4  
  All pvs from storage ring cell 1 belonging to element bpm:4

• **SR:** *C01* & **elemName=bp**: 4 & **pvStatus=active**  
  All pvs from storage ring cell 1 belonging to element bpm:4 and with pvStatus active

• **SR:** *C01* & **elemName=bp**: 4 & **tag=aphla.sys.SR**  
  All pvs from storage ring cell 1 belonging to element bpm:4 with tag aphla.sys.SR
Populating ChannelFinder

• cf-update
  • Adds new channels
  • Manages existing channels
    • Orphaned channels
    • Moved channels

• Python scripts
• cf-properties (under development)
Example st.cmd

dbLoadRecords("gauss.db","P=ktest")

epicsEnvSet("EPICS_HOSTNAME", "dev32new")
epicsEnvSet("EPICS_IOCNAME", "gauss")

iocInit()

# pipe the output of the dbl command to a file
# the file name should follow the convention 'myHostName.myIOCName.db'
# write the file to a well know directory on which the cfmonitor deamon is running
dbl > $(CF_UPDATE_DIR)/$(EPICS_HOSTNAME).$(EPICS_IOCNAME).db
Under the Hood

• The cf_monitor daemon monitors $(CF_UPDATE_DIR) directory and envokes an update task when
  • A new *.dbl file is created in the dir
  • An existing *.dbl file is modified

• The update task handles
  • Uses the filename to obtain the hostName, iocName property values
  • New channels – creates new channels with hostName, iocName, pvStatus and time properties
  • Orphaned channels – pvStatus property is updated
  • Moved channels – ensures that the hostName, iocName properties are update when channels are moved
  • Unchanged channels

• In all cases existing properties(excluding hostName, iocName, pvStatus, time) and tags are left unaffected
<table>
<thead>
<tr>
<th>Channel Name</th>
<th>SR:C02-MG:G04A{HFCor:FM1} Fld-I</th>
<th>SR:C02-MG:G04A{HFCor:FM1} Fld-SP</th>
<th>SR:C02-MG:G04A{HFCor:FM1} Fld-I</th>
<th>SR:C02-MG:G04A{HFCor:FM1} Fld-SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>READBACK</td>
<td>SETPOINT</td>
<td>READBACK</td>
<td>SETPOINT</td>
</tr>
<tr>
<td>elemName</td>
<td>FXM1G4C02A</td>
<td>FYM1G4C02A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>elemType</td>
<td>HFCOR</td>
<td>VFCOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>elemField</td>
<td>x</td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>devName</td>
<td></td>
<td>FM1G4C02A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sEnd</td>
<td>65.5222</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>C02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>girder</td>
<td>G4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>symmetry</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>length</td>
<td>0.044</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ordinal</td>
<td>263</td>
<td></td>
<td>264</td>
<td></td>
</tr>
<tr>
<td>tags</td>
<td>eget</td>
<td>eput</td>
<td>eget</td>
<td>eput</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>y</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>sys.SR</td>
<td></td>
</tr>
</tbody>
</table>
## ChannelViewer

The ChannelViewer interface is shown in the image. It appears to be part of a software tool for monitoring and controlling systems, possibly related to a control system studio (NSLSII). The interface includes a table with columns for `Channel Name`, `cell`, `girder`, `elemType`, `elemName`, `elemPosition`, `elemField`, `elemIndex`, and `system`. Each row in the table represents a channel with specific attributes.

### Table Headers
- **Channel Name**: A unique identifier for the channel.
- **cell**: The cell identifier.
- **girder**: The girder identifier.
- **elemType**: The type of element associated with the channel.
- **elemName**: The name of the element.
- **elemPosition**: The position of the element.
- **elemField**: The field associated with the element.
- **elemIndex**: The index of the element.
- **system**: The system identifier.

### Example Data
- **V.1-SR:C01-MG:G2(QL1:134)FldSP**: Channel with specific attributes and identifiers.
- **Copy PV name to clip-board**: Option for copying data.
- **EPICS PV Tree**: Explorer tree for EPICS PVs.
- **Probe**: Option for probing data.
- **PV Table**: Table for PVs.
- **OPI Probe**: Option for OPI probes.
- **Data Browser**: Browser for data.
Auto-complete
PVManager Formula

- $\text{cf("\* elemType=HFCOR handle=SETPOINT tags=sys.SR")}$

Queries ChannelFinder and returns a VTable consisting of all of channels for the setpoint pvs for all horizontal fast correctors in the storage ring
Even more
Interesting Experiences
ChannelFinder data

• Over 180k channels
• Over 1.2 million instances of channel + tag/property

• Service response (Query + Parsing) in the order of seconds
• Embedded (denormalized) data model
  • Ideal for one-to-many relationships between entities.
  • Provides better performance for read operations, as well as the ability to request and retrieve related data in a single database operation.
  • Embedded data models make it possible to update related data in a single atomic write operation.

// channels
{
  name: “channel name”,
  owner: “channel owner”,
  properties: [{name: “property name”,
    owner: “property owner”,
    value: “property value”,
    _id: properties._id},.....]
  tags: [tags._id,.....]
}

// tags
{
  name: “Tag name”,
  owner: “Tag owner”,
  _id: ID
}

// properties
{
  name: “Property name”,
  owner: “Property owner”,
  _id: ID
}
Performance Environment

• Database
  • 150k channels
  • 15 million channel-properties/channel-tags

• Queries
  • Search based on channel names and property values
Mysql Vs MongoDB

Mysql

MongoDB
Questions?
Links:

Control System Studio
  • http://controlsystemstudio.org/

ChannelFinder
  • http://channelfinder.sourceforge.net/
  • https://github.com/ChannelFinder/ChannelFinderService

Performance Tests
  • https://github.com/shroffk/cf-mongo-java-test
  • https://github.com/mskinner5278/cf-mongo-test

JavaFx
  • https://github.com/sjdallst/FXvsSWTProfiling
  • http://www.eclipse.org/efxclipse