Toy IOC Experiments, hierarchical EPICS database

A side product of
"V4" discussions with:

Matthias Clausen,
Bob Dalesio,
Marty Kraimer,
Steve Lewis.

Kay-Uwe Kasemir, June 2006

The Toy IOC

- IOC to try V4 ideas.
  - Java
  - Jython as Shell
  - Reads Databases, processes Records
  - Uses Java Introspection to
    - locate Record implementations
    - locate Link Handlers
      (Constant, Database, CA V3, File, Random).
    - handle Data
      (Scalar Double, String, Boolean, Enum).

Analog Input Records, Links

- Constant (and "PINI"):

```
<record type="ai" name="apx_pi">
   <initial_process>true</initial_process>
   <input><link type='constant'>3.14</link></input>
</record>
```

- Read numbers from a file:

```
<record type="ai" name="data">
   <initial_process>true</initial_process>
   <input><link type='file'>example/data.dat</link></input>
</record>
```

- Database links,
  which can 'process' their source/target ("PP"):

```
<record type="ai" name="copy">
   <scan>1.0 second</scan>
   <input><link type='database' process='true'>data.value</link></input>
</record>
```

Analog Input Records, Links

- Read a CA3 link, process when link updates ("CP"):
  - Type ‘ca3’ or ‘database’ needs to be explicit.

```
<record type="ai" name="excas_fred">
   <input monitor='true'><link type='ca3'>fred</link></input>
</record>
```

- Read random number "device", proc. when link updates ("I/O Intr"):

```
<record type="ai" name="noise">
   <input monitor='true'><link type='random'>1.0</link></input>
</record>
```

String Record

- **String records can use the same links, converting data to string if necessary:**

```
<record type="stringIn" name="data">
   <scan>1.0 second</scan>
   <input><link type='file'>example/data.dat</link></input>
   <forward>copy</forward>
</record>

<record type="stringIn" name="copy">
   <input><link type='database'>data.value</link></input>
   <forward>random</forward>
</record>

<record type="stringIn" name="random">
   <input><link type='random'>10.0</link></input>
</record>
```

- **Forward links as usual.**
  - **More than one is allowed.**

Calc Record

```
<record type="calc" name="counter">
   <description>Ramp Up</description>
   <scan>1.0 second</scan>
   <input name='v'><link type='database'>counter.value</link></input>
   <input name='uno'><link type='constant'>1.0</link></input>
   <calc> v + uno - 0.5 </calc>
   <units>Counts</units>
</record>
```

- CALC inputs
  - have user-defined names
  - have no arbitrary limit to their number.
- Strings in general
  - no arbitrary length limits for record name, description, units, ..

A Link is a Link

- On the API level, "Link" is an interface:
  - read,
  - write,
  - add/remove State and Value listeners.
- The CALC record reads or monitors links
  - Any link type allowed.
  - No need for intermediate AI records to read from "device".

```
<record type="calc" name="sum">
   <scan>1.0 second</scan>
     <input name='v'><link type='file'>example/data.dat</link></input>
     <input name='n'><link type='random'>1.0</link></input>
   <calc> v + n </calc>
</record>
```

Records and their Fields, Channels and their Properties

- EPICS currently distinguishes between
  - Channel "fred == fred.VAL" with Properties { value, status, severity, stamp, units, …}

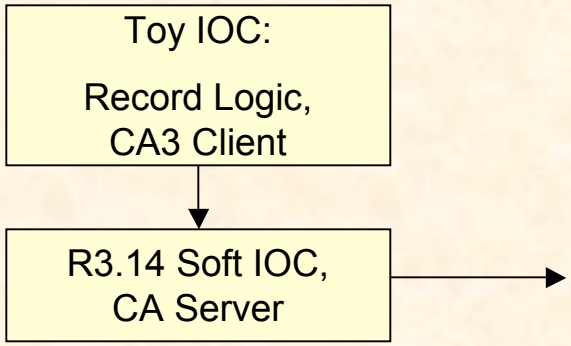
  and
  - Record "fred" with Fields { VAL, STAT, SEVR, TIME, EGU, … }
- Mapping of Fields to Properties
  - Almost only makes sense for channel "*.VAL".
  - Is hidden in record support code.
- Experience from EPICS training and questions:
  - Causes a lot of confusion.

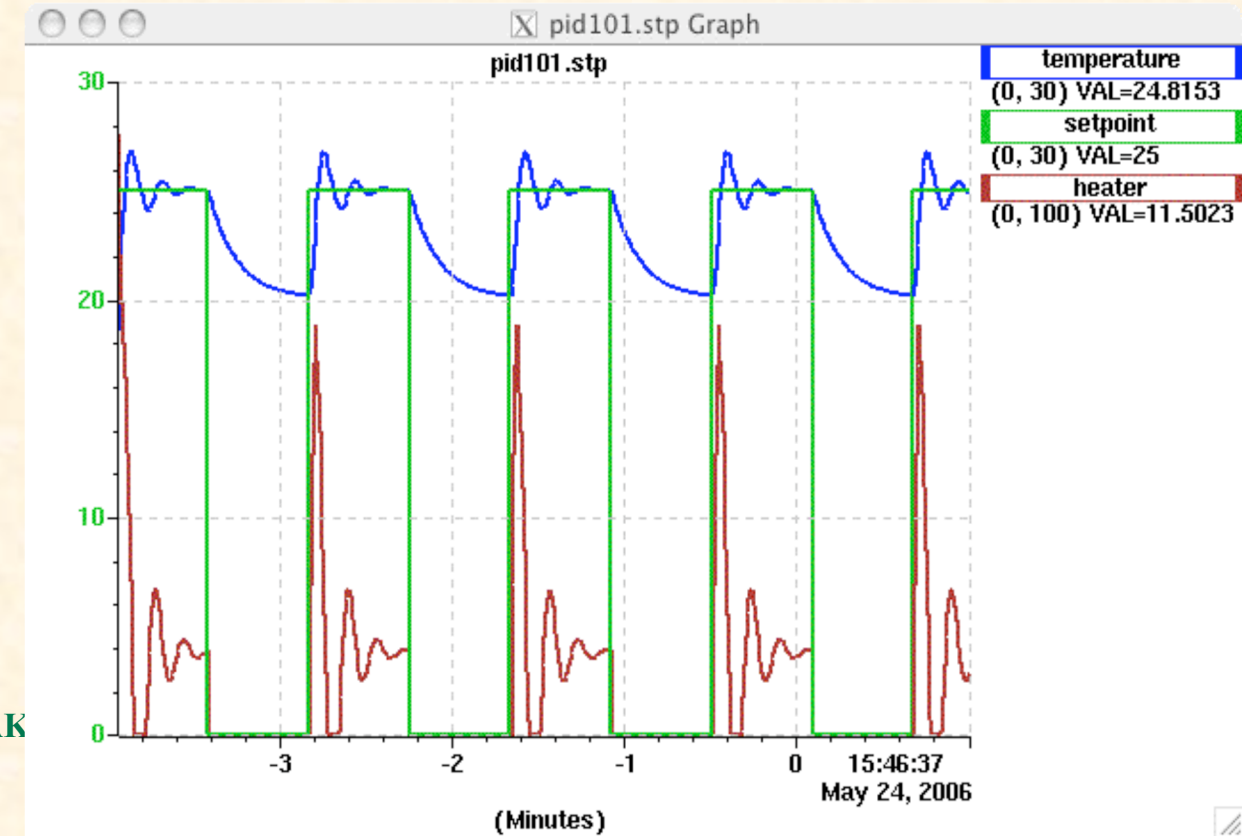
Why not Channel=Record, Prop.=Field?...

- Record provides a list of properties.
- Properties allow read/write/monitor.
- A client who wants DBR_TIME_DOUBLE asks for
  { "status", "severity", "time", "value" }.
- Subscribers get whatever changes.

"Water Heater" from EPICS Training

- **Implemented as Toy IOC Database:**
  - **Prop. & Integral Controller**
    - `<record type="ai" name="setpoint"> …`
    - `<record type="calc" name="error"> … "error_sum" … "output" …`
  - **Water Heater & Water Tank Temperature Simulation**
    - `<record type="calc" name="heater"> … "temperature" …`
  - **'Push' data into V3 soft-IOC for e.g. StripTool**
    - `"ao" records …`





UT-BATTELLE

OAK

So far so good

- Quite similar to V3.
- What else could one do?
- What do others do?

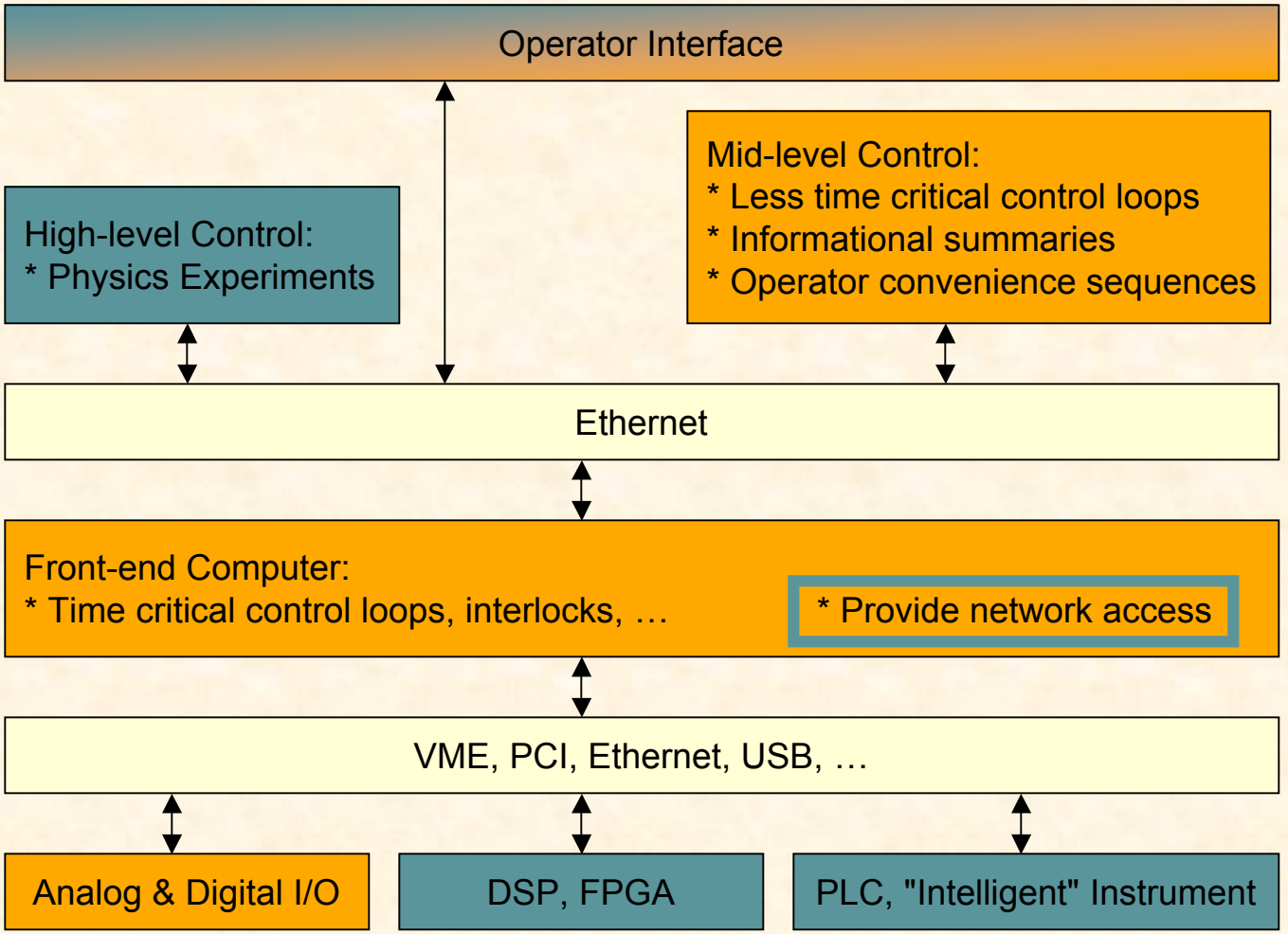
What's the difference between...

- Tango, ACS, DOOCS, NIF, RHIC, CDEV, ….
  and EPICS?

  - EPICS
    - "IOC" Application Framework: Database, Sequencer.
    - Channel Access: Generic, flat list of channels.

  - "The Rest"
    - More DIY front-end coding, using C/C++/Java libs.
    - Network protocol: OO Interfaces to "Device" Objects.

Where do EPICS or "The Rest" excel?



What EPICS V3 does very well…

- EPICS Database handles scheduling, I/O, conversion, smoothing, limits, and network updates for simple devices (VME register, ADC/DAC channel):

```
record(ai, "temp") {
      SCAN  "1 second"
      DTYP  "XY ADC"
      INP   "#C2 S3"
      LINR  "typeKdegC"
      EGU   "deg C"
      HIGH  "40.0"
      SMOO  "0.5"
      MDEL  "0.01"
      …
   }
```

- Database configuration instead of C/C++/Java code creation avoids many pitfalls.

What EPICS V3 does less well...

- Data on FPGA, DSP, PLC, Intelligent Device already has its own units, limits, conversion parameters, alarms.
  - How to map those to fields of an ai or ao record?
  - Often, you don't, and spread the data across many records: ai "value", mbbi "status", ao "low_limit", …
- Do custom record types help?
  - Are there record types "Pump", "Valve", "BPM"?
  - Record type creation is hard, the results are rarely shared.
  - PID, ePID, cPID, a few "Magnet" records.
- Lack of structure above PV level:
  - A PV is just a named number.
  - No way to find the "status" PV for a "value" PV, or the "setpoint" for a "readback".

What is "The Rest" up to?

- Object-Oriented "Device" interfaces instead of PVs
  - MagnetPS:
    current setpoint, current readback, on/off status,
    'on' command, 'off' command.
  - CurrentLimitedRampingMagnetPS:
    MagnetPS + current limit, current ramp rate.
- Advantages:
  - OPIs can auto-configure based on the device type.
  - Higher-level applications:
    Command-completion, compile and run-time type safety.
  - Physics Applications can search for
    "all BPMs in sector 7"
    (A name server as well as device type issue).

Issues with "The Rest"

- Limited IOC development support:
  - Network library, maybe a code skeleton generator, but below that it's do-it-yourself C/C++/Java code.
- Lack of runtime database means
  - Possibility of bugs related to scanning, threading.
  - More difficult to understand somebody else's application, since one can't peek into known records.
  - No runtime changes unless anticipated in your custom code.

Can one have it all?

- A runtime database?
- "Devices" on the network?
- Simplified creation of new record types?

Create New Record Types?

- **Problem with PID, ePID, cPID, Magnet record:**
  - **There's typically *something* inside that you *don't* like.**
  - **C/C++ Code.**
    Hard to modify by "IOC database" person.
- **Compare: LEGO**
  - **You hardly *want* to create new LEGO blocks.**
  - **The simpler the blocks, the better.**
  - **Important: They can be combined.**

Idea: Simpler Records...

- **Make existing records simpler**
  - **BO with 'HIGH'**
    ⇒ remove from BO, have separate DELAY record.
  - **CALCOUT**
    ⇒ 2 CALC, new CONDITION record, AO.
  - **MOTOR, PID, …**
    ⇒ a whole bunch of records.
- **… but allow combination of those simple records.**

Condition Record

```
<record type="calc" name="toggle">
   <scan>1.0 second</scan>
   <input name='v'><link type='database'>toggle.value</link></input>
   <calc> 1.0-v </calc>
</record>
<record type="condition" name="The_Decider">
   <input monitor='true'><link type='database'>toggle.value</link></input>
   <if_true>yes</if_true>
   <if_false>no</if_false>
</record>
<record type="base" name="yes"> </record>
<record type="base" name="no">  </record>
```

- Compare to CALCOUT
  - More obvious, yet in principle more versatile.
  - Current implementation only "if true", "if false"; no "on change", "on transition to 0" etc. as in CALCOUT.

Simpler ... But With Hierarchy

- We usually have hierarchy in the
  - Underlying problem,
  - RDB structures used to create EPICS database,
  - Naming Standards,
  - Capfast or now also VDCT schematics.
- Yet EPICS records are flattened?
- Why not keep the records hierarchical?
  - Database view, accessible at runtime, better matches the problem/RDB/naming/schematics.
  - Eases the mapping to "devices" seen on the network.

Device Record

```
<record type="device" name="thingy">
    <description>Device Test</description>
    <record type="ao" name="setpoint">
        <scan>1.0 second</scan>
        <input><link type='file'>example/data.dat</link></input>
        <units>tics</units>
    </record>
    <record type="ai" name="readback">
        <input monitor='true'><link type='database'>setpoint.value</link></input>
        <units>tocs</units>
    </record>
</record>
```

- Record type "device" contains other (sub-)records.
- Records process as usual, "device" really only a namespace.

IOC Shell View (Jython)

```
>>> ioc.loadRecords("example/device101.xml")
>>> ioc.start()
>>> ioc.ls()
'thingy'
>>> ioc.ls('thingy')
'thingy' :
[scan, initial_process, description, process_active,
  debug, time, status, forward, /setpoint, /readback]
>>> ioc.ls('thingy/readback')
'thingy/readback' :
[input, value, units, scan, initial_process, description,
  process_active, debug, time, status, forward]
>>> ioc.get('thingy/readback.value')
'thingy/readback.value' = 10.0
```

Hierarchy

- **Current implementation: File System idea for Names.**

  /thingy

  /setpoint

  .value, .units, …

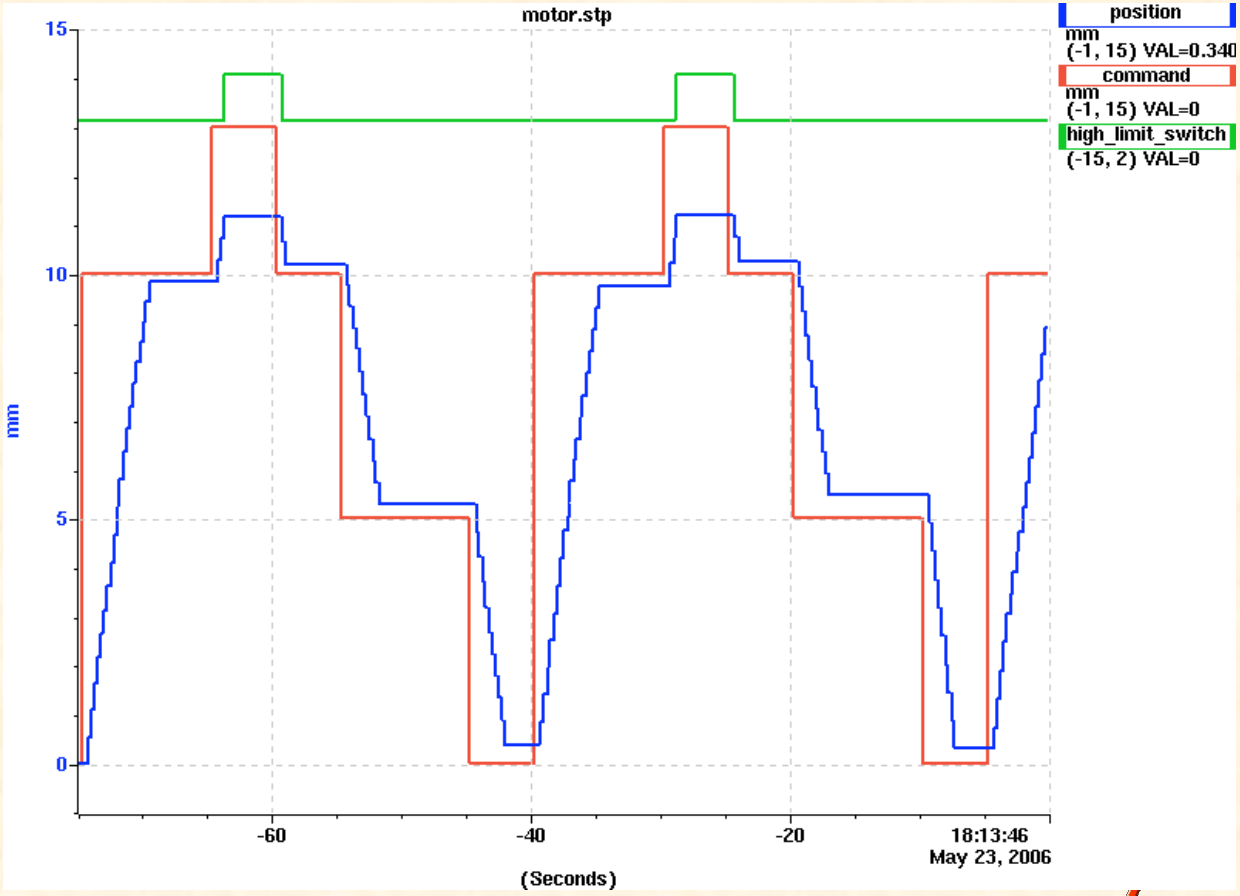
  /readback

  .value, .units, …

- **On the network, this could map to a "device" with properties "setpoint" and "readback"**
  - **May need added "device class" description, so database can assert that a "device record" indeed provides all properties which clients expect.**
  - **Still: Properties for the "device" on the network provided by records, not custom C/C++/Java code.**

Motor Example

- Device record with
  - Command, position, limit switches, …
  - Reads commands from file.
  - Simulates stepper motor.
  - Sends certain values via "ao" record to a V3 soft IOC for StripTool plot.



IOC Shell for Motor Example

```
>>> lsR()
motor
+-- motor/command
    +-- input
    +-- value
    +-- units
    +-- description
    ...
+-- motor/limit_switches
    ...
+------ motor/limit_switches/low
        +-- input
        +-- calc
        +-- value
        +-- units
        ...
+------ motor/limit_switches/high
...
```

IOC Shell for Motor Example...

```
>>> ls('motor')

'motor' :

 [scan, initial_process, description, process_active,
  debug, time, status, forward, /command, /state,
  /limit_switches, /new_state, /position, /ca3_pusher]

>>> cd('motor/position')

>>> ls()

'motor/position' :

 [input, calc, value, units, scan, initial_process,
  description, process_active, debug, time, status,
  forward]

>>> get('time', 'status', 'value', 'units')

'motor/position.time' = 2006/06/06 16:42:55.616000000

'motor/position.status' = ok

'motor/position.value' = 2.494510838231727

'motor/position.units' = mm
```

Summary

- Toy IOC
  - Java platform (with Eclipse IDE) to try V4 ideas.
- Record Types
  - can be simpler (no BO.HIGH, no CALCOUT, no PID …),
  - combining the simpler records gives same functionality.
- Record Hierarchy
  - Often better maps problem, schematics, RDB, … to the EPICS database.
  - Basis for mapping EPICS database to "devices" on the network as now used by everything but EPICS.

More Ideas

- Processing a "Device" record currently does nothing
  - Unless one makes its <forward> point to e.g. one of the records inside the device...
- "Device" record has no meaningful properties of its own
  - Could add input and output ports which map to properties of embedded records:

```
<record type="device" name="thingy">
  <ports>
    <in target="_setpt" process="true">setpoint</in>
  </ports>
  <record type="ao" name="_setpt"> …
```

  - Mapping to "device" on network could be restricted to ports, while remaining sub-records stay ‘private’.
    On the other hand, EPICS V3 does well with making everything network-accessible…