Database & Bootscript Generation @ BESSY

T. Birke

BESSY

Control System Status

T. Birke

BESSY

Project Status Control System

T. Birke

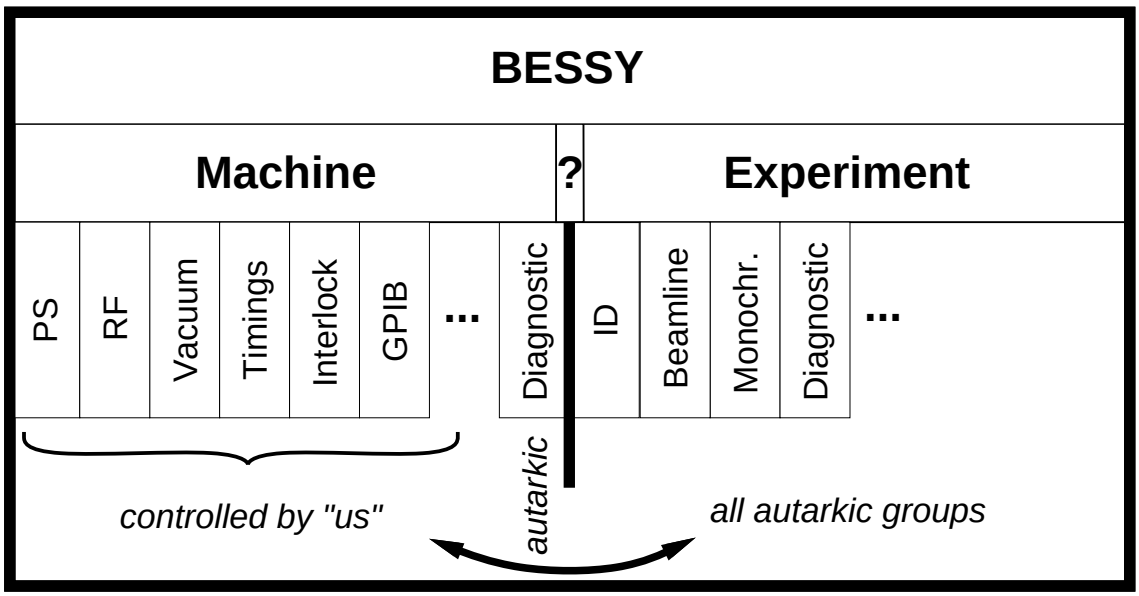
History

- Project BESSY II started '92
- experiences with BESSY I
  - small machine, home-made control system
  - all configuration in files
  - maintaining configuration was an *experts only* task
  - configuration system not transferable to BESSY II
- RFC for a naming convention to be used
- early decision to use central database for configuration management (mid '94)
- main type of device to control: power supply
  - early specification of interface
  - *all interfaced by same I/O-card / embedded controller*
  - *all interfaced by CAN-bus*

  → ~400 PS of 12 different types interfaced identically

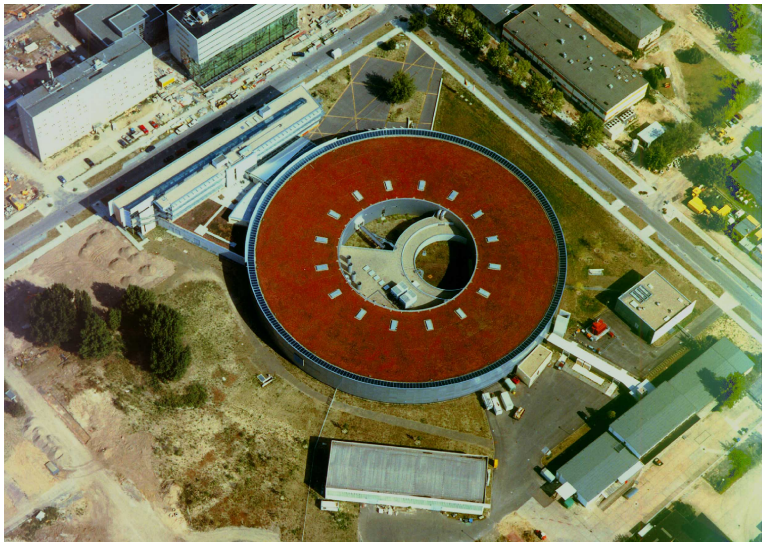
  → idea of templates filled with individual data
  - first entry of data late '96
  - slightly modified over the last 3 years

Controls at BESSY



- Two separate worlds *M* and *E*
- No controls group in *E* division
  every group does it's own specs, interfacing, controls. . .
- Cooperation on *lower* levels of hierarchy works quite well
- Decision was made, to develop a division independent superseding structure for
  - host configuration
  - software distribution (installation procedures)
  - data transparency (online data, measurements and archived data as well as configuration data → RDB)

Some figures



BESSY in operation since September '98

- 3rd generation light source
- 1.7 GeV electron storage ring (240m circ.)
- 50 MeV microtron (gun)
- 10 Hz booster (96m circ., full energy injection)
- 16 straight sections (13 for IDs, 8 inst., 2 in prog.)

Machine Control System as of Sep 1 2000

- 25 IOCs
- 27296 records
  - 6511 doing hardware I/O (~24%)
  - 6151 of which I/O over the CAN-Bus (~95%)
- 1881 *devices*
- 573 CAN-nodes
- 58 CAN-segments

Device Architecture
Variety of Combinations

Unique & Complex I/O partitioned/documented

- RF system (I/O interface to PLC)
  5 devices, 1875 channels
- WLS
  2 devices, few hundred channels

Intermediate Systems solved

- Scraper controls
  4 stepper motors
- GPIB devices
  1 oscilloscope, 1 master clock

High multiplicity / Simple I/O solved

- Power Supplies
  ~400 devices with ~15 channels each

Non conformal systems isolated

- Feedback systems
  blackbox system / turnkey solution
- Machine diagnostics
  autarkic group, seperate from CS group

Naming Convention

Schema `<DEVICENAME>:<channel>`

- e.g. `S4P3D6R:rdbk` → [S4] [P] [3] [D6] [R] : [rdbk]
- *device* models an abstract unit
- consists of a set of channels
- devices of same class share same set of channels

genome ↓

[HS4] [M P] [3] [D6] [R] — Ring Lattice

Horizontal corrector (windings) in Sext 4 of...

[HFB] [M P] [2] [T4] [R] — Local Orbit Feedback

Horizontal corrector of the feedback system in...

[(P)KI] [K] [3] [D1] [R] — Injection Kicker

if K *then*^P ⇒ power supply

[UE56] [I] [D6] [R] — Insertion Device

ID UE56 as a complex device with lots of *internal* channels

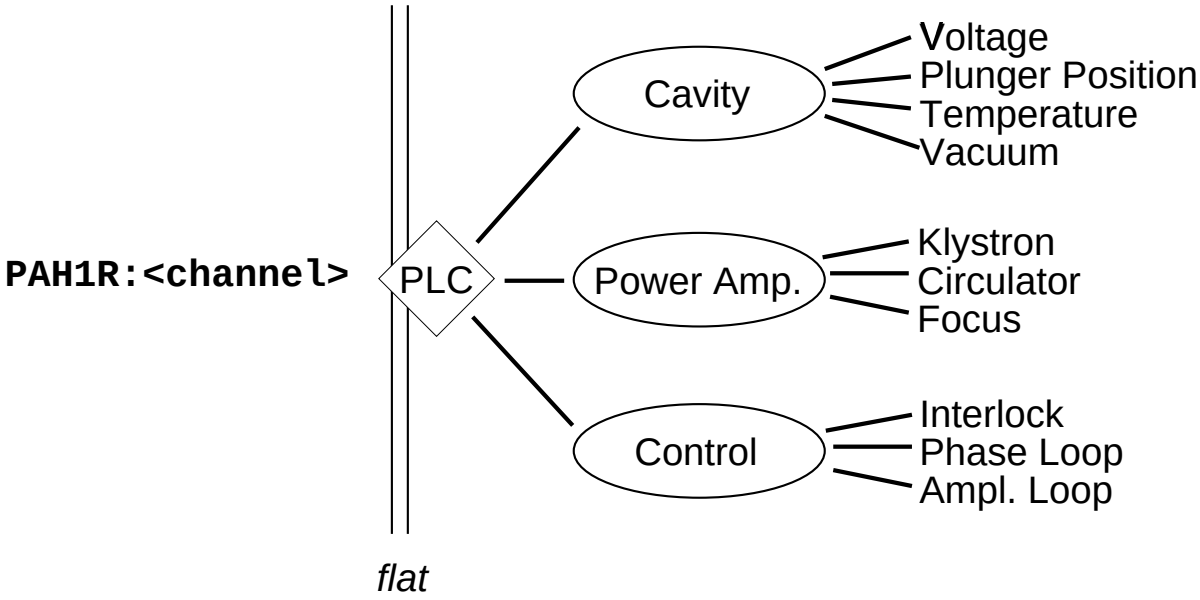
[(P)HB UE56] [I] [2] [D6] [R] — Insertion Device

if I *then*^P ⇒ power supply of

HB ⇒ horizontal bending magnet of UE56 *external!*

Overlapping: A device is **part of** another device!

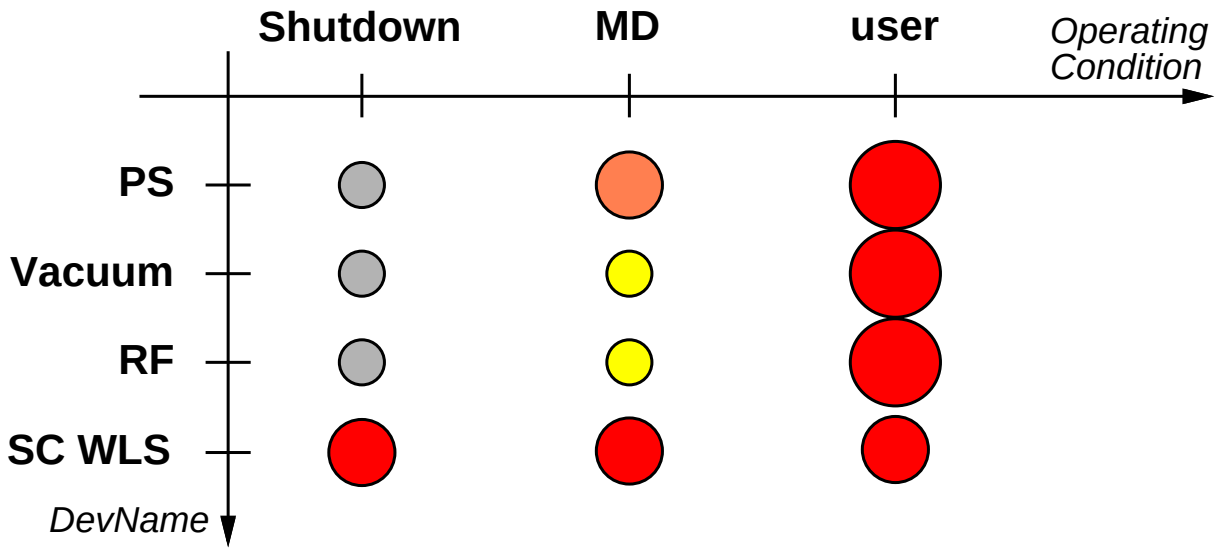
Naming Convention Partitioning



e.g. RF devices

- 5 devices, that appear flat in the control system
- PLC has internal structure
- structuring takes place in `channel` and is not part of the naming convention
  → doesn't follow common rules
- structure does not appear in EPICS DB
- internal structure of PLC remodeled in RDB to make e.g. *alh* useable

Roles of Devices



- Value of Information depends on Operating Condition and/or Information about other devices
- Applications have to behave different depending on the state/mode of the whole machine

  Examples:

Save/Restore	load condition on restore of snapshots
Model	active/inactive elements
Alarms	transient/conditional
Archiving	active period, frequency/monitor

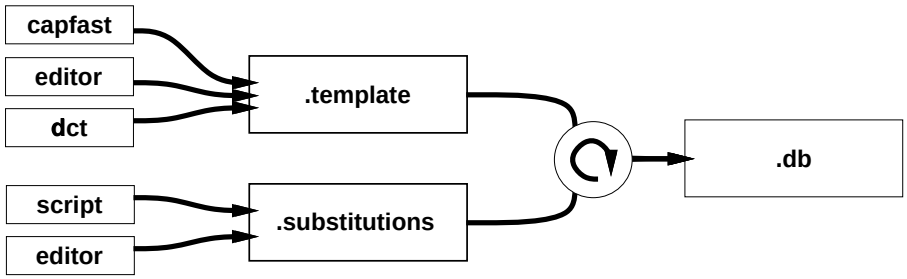
→ Coding of these roles in every Application / Configuration is error-prone (if possible at all)

→ Definition of these roles needed but experience with machine necessary

→ How to model these roles at all?

Database Generation Current Procedure

- database templates from either
  - capfast drawings
  - mini templates (dct or by hand)
- substitution files from either
  - tcl/perl scripts with RDB access
  - tcl/perl/shell scripts



→ expansion to db-file either on host or IOC

- same mechanism used for
  - power supplies
  - vacuum system
  - triggers / timings
  - RF PLC-interface
  - misc. I/O
  - ...

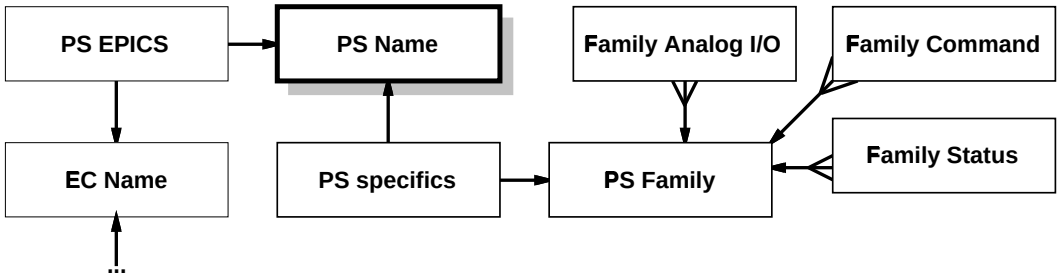
Database Generation Example: Power Supplies (I)

Precondition

- about 400 PS of 12 different types (families)
- all interfaced with same I/O-card, embedded controller and CAN-interface

Data in RDB

- PS specific data
  name, max. current, family, . . .
- EPICS specific data
  DRV[HL], [HL]OPR, PREC, ASG, . . .
- Fieldbus specific information
  embedded controller, CAN segment, CAN node ID, . . .
- Definition of Families
  Analog I/O, Commands, Status-Information. . .



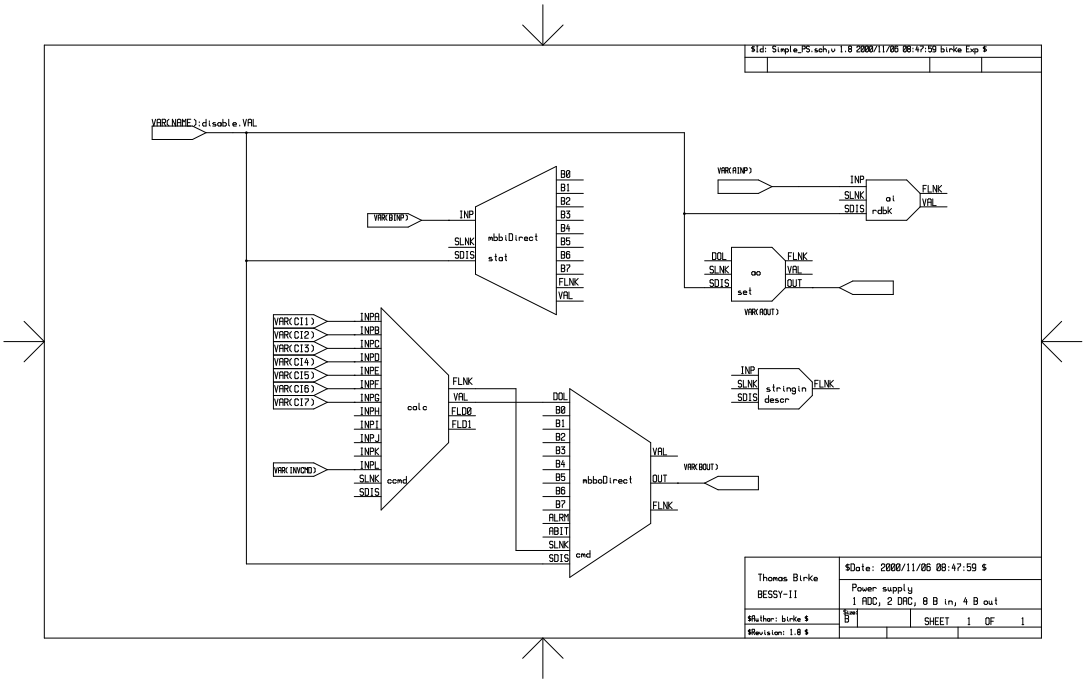
Resposibilities

- PS specifics and all Family data maintained by device responsible (usually ;-)
- Name and EPICS specifics maintained by controls group

Database Generation Example: Power Supplies (II)

Creating db files

- capfast schematic → db-template



- misc. stuff in mini-templates (dct)
- Tcl script (oratcl) does RDB-queries and creates per-IOC substitution files
- some special cases not covered by information stored in RDB!

→ "some" information is located in the Tcl code (Script has about 800 lines!)

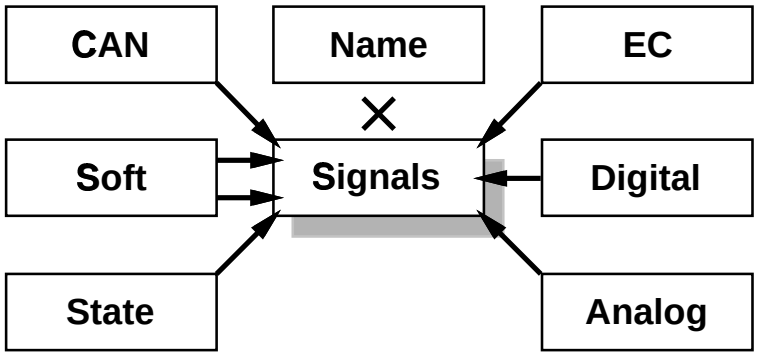
Database Generation Example: flat I/O (I)

Precondition

- flat
- interfaced via CAN-Bus (directly and ADA16)
- RF: 5 identical PLCs with ~ 300 signals each, interfaced via CAN-Bus (directly and ADA16)

Data in RDB

- Devicename
- Signalnames / -types
  analog, binary
- I/O specification
  input/output, location of signal . . .



Resposibilities

- controls group and device-responsible develop interface definition
  Hardware I/O, CAN-Bus. . .
- All information in DB is maintained by controls group
  initial list of signals delivered by device-responsible

Database Generation Example: flat I/O (II)

Creating db files

- usually mini templates (1 or 2 records) created by hand, dct or capfast
- Tcl script (oratcl) or perl (oraperl, DBI) script does RDB-queries and creates per-IOC or per-device substitution files
- scripts are rather primitive (< 200 lines) just gathering data from RDB and formatting output to create substitution file

→ No information located outside RDB

→ Experience:
Data in RDB are harder to maintain

but scripts work for all flat I/O

Database Generation Example: Without RDB

Precondition

- "complex" device → "complex" template
  e.g. GPIB scope → 32 interconnected records
- very few (usually one) instances

Data in RDB none at all!

Resposibilities

- controls group is completely responsible for device
- Interfacing e.g. a oszilloscope is "just needed"

Creating db files

- template usually created with capfast
- script creates per-device substitution files
- script is very primitive (< 20 lines)

Bootscript Generation (I)

Requirements

- which drivers to load
- which databases to load
- which hardware to initialize
- which sequencers to start (and how)
- misc. initialization

Current Procedure

- one Application with sophisticated Makefile.Host and a lot of ASCII-files creates all startup-files
- startup-files consists of blocks
  - environment variables
  - main directory for IOC
  - binary modules to load
  - drivers to initialize
  - dbd to load
  - template/substitutions to load
  - db-files to load
  - initialisation commands
  - sequencers to start
  - extraneous commands after startup
- defaults are provided that can be overridden by IOC-specific block-files

Bootscript Generation (II)

Problems

- relevant information located in "source code"

  e.g. db files are generated on a per-IOC basis from RDB, and the bootscript has to load/initialize the driver and load the database

  → consistency is maintained manually!

Example

```
################################################## Global Stuff
putenv "TIMEZONE=MET::-60:033002:102602"
putenv "EPICS_TS_MIN_WEST=-60"
< ../rshCommands
cd "../.."
################################################## Load Binaries
ld < bin/mv162/iocCore
ld < bin/mv162/StandardSupport
initHookRegister(mv162Hooks)
initHookRegister(mCANHooks)
################################################## Load Databases
dbLoadDatabase("dbd/Standard.dbd")
cd "db"
dbLoadTemplate("IOC-stats.IOCS2G.substitutions")
dbLoadRecords("IOCS2G.db", "IOC=IOCS2G")
dbLoadRecords("disable.IOCS2G.db", "IOC=IOCS2G")
dbLoadRecords("Vacuum.IOCS2G.db", "IOC=IOCS2G")
################################################## Configure IOC Core
iocLogDisable=0
TSconfigure
asSetFilename("/opt/IOC/BII-Controls/db/security.acf")
################################################## Ignition...
iocInit
dbior 0, 1 > /opt/IOC/log/Database/IOCS2G.dbior
dbhcr > /opt/IOC/log/Database/IOCS2G.dbhcr
dbl > /opt/IOC/log/Database/IOCS2G.dbl
################################################## State Machines
```

Conclusion

- Database generation meets our requirements very well

  improvements possible, but cost-benefit analysis forbids activity

- Bootscript generation works well

  but should use RDB to get info about what should be done on which IOC

→ Priority on RDB-ToDo-list is rather low