

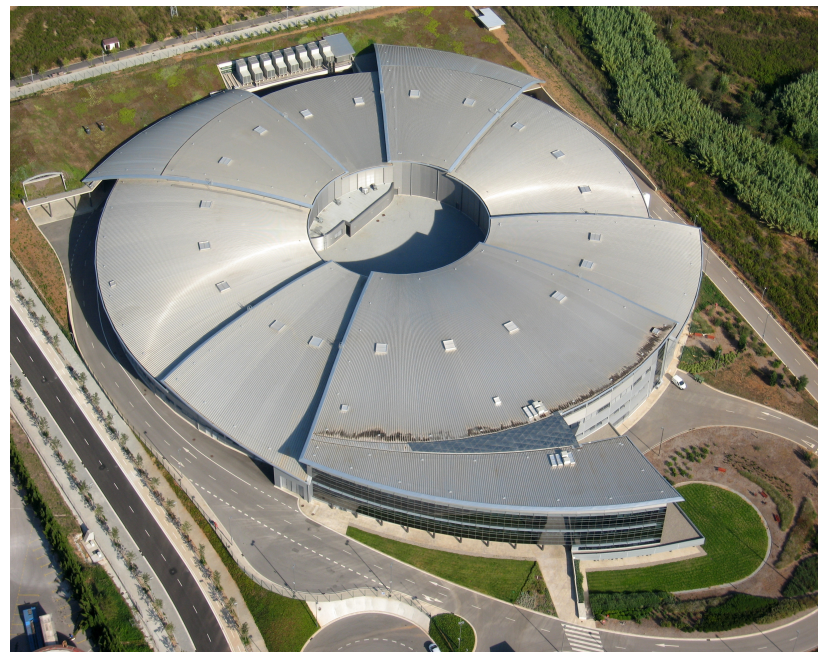
Sardana

– a Python Based
Software Package for Building Scientific
SCADA Applications

Zbigniew Reszela

on behalf of the Alba Controls Group

PCaPAC 2014



Alba Synchrotron Light Facility



Sardana logo

1

Scientific Installations
demands & challenges

2

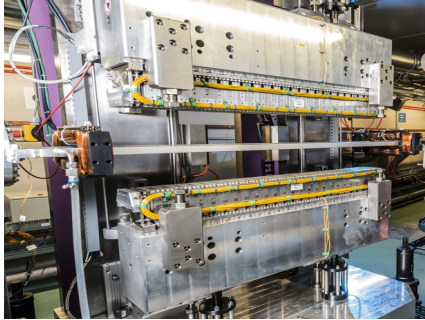
Sardana and its
components

3

Sardana Community

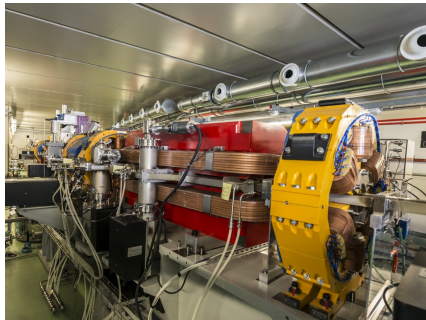
4

Current and future
developments



One of the ALBA insertion devices

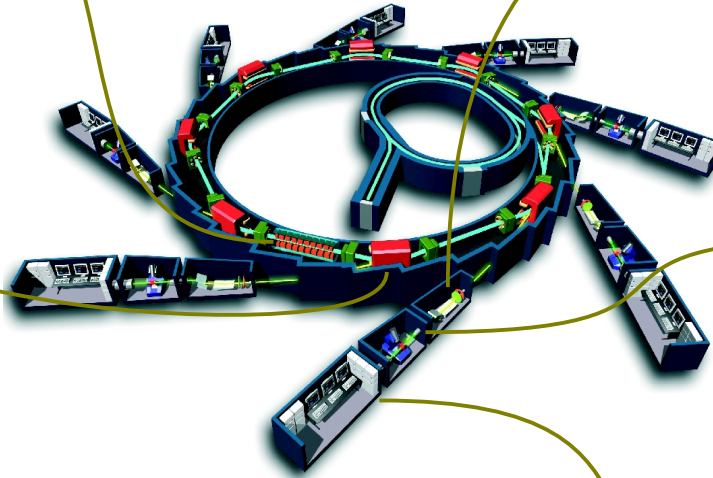
The particle **accelerators** comprises power supplies, vacuum equipment, radio frequency stations, insertion devices and many diagnostics and actuators among others.



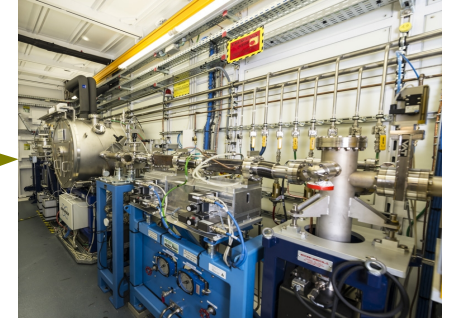
Magnets of the ALBA booster



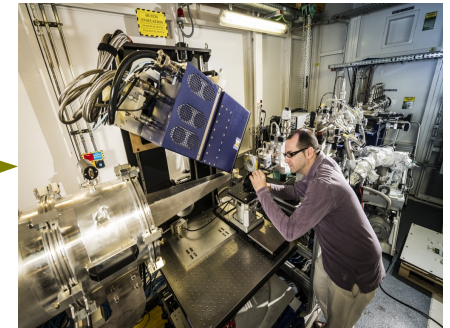
The ALBA control room



The laboratories, e.g. a synchrotron **beamline**, usually consists of even more diverse instruments like, diffractometers, monochromators and sophisticated detectors full of moveable axes and experimental channels.



BL04 optical hutch



BL11 experimental station

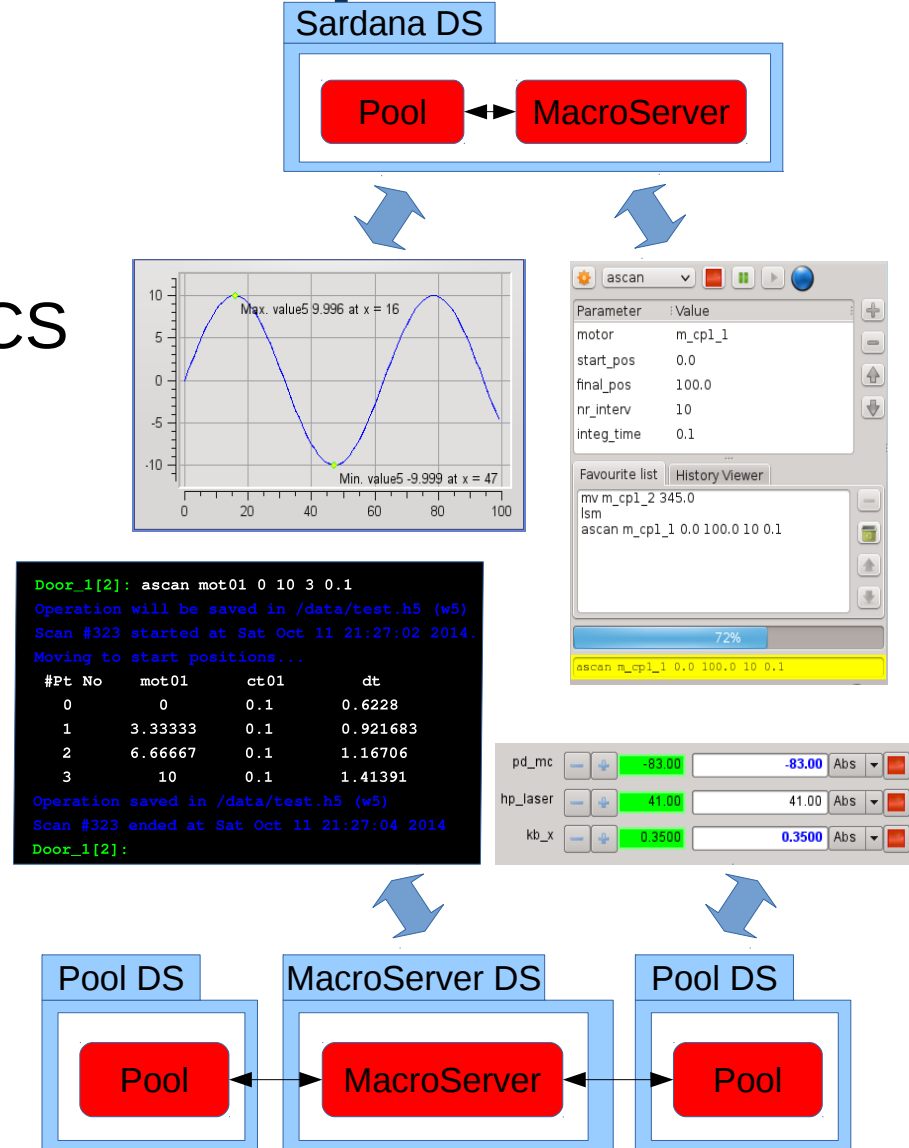


BL13 control hutch

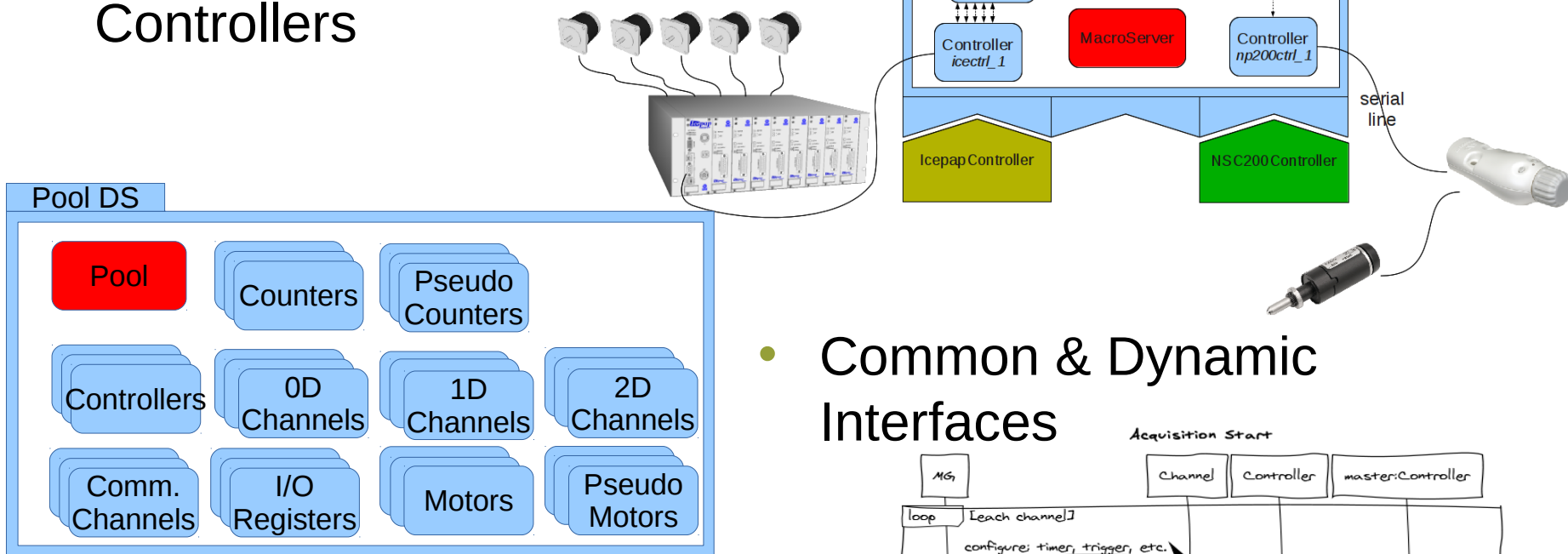
Commercial SCADA (typically oriented to industrial applications)	Scientific SCADA
Does not require high flexibility. (industrial processes do not change their configuration frequently)	Beamlines require high flexibility . (experiment processes changes their configuration frequently)
Provide primitive sequencer.	Require powerful sequencer oriented to the laboratory instruments.
Favor PLC hardware solutions.	Handles heterogeneous hardware using different communication protocols and of different vendors.
Data acquisition is used mostly for monitoring and is stored in DB (data-timestamp pairs)	Performs synchronized data acquisition processes and stores data in the specific file formats .
Most control actions are performed automatically by RTUs or PLC.	Most control actions are performed on operators/scientists request .
	Requires specific capabilities e.g. diffractometer control.

Sardana and its components

- SCADA software suite
- It is open source (LGPL)
- Allows building distributed CS
- Uses Client - Server model
- Communicates via Tango
- Server:
Device Pool (HW access)
MacroServer (sequencer)
Sardana (all in one)
- Client: Taurus HMIs

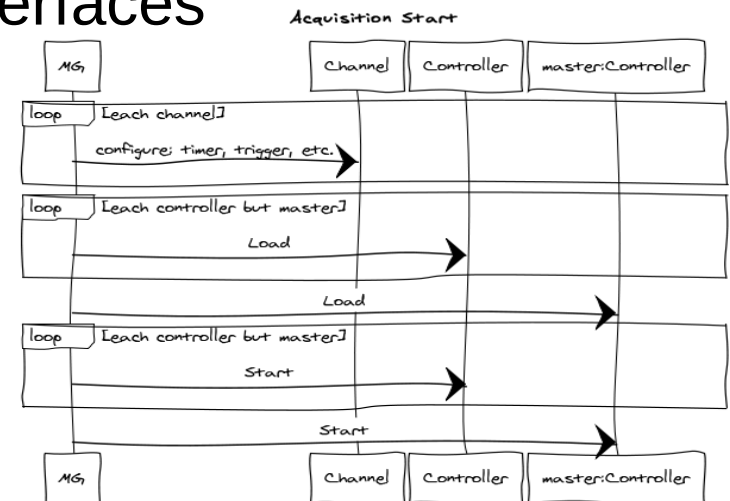


- Hardware Access via Controllers



- Control algorithms: Motion, Acquisition

- Common & Dynamic Interfaces



- Sequencer capable to plan and execute user procedures - **macros**
- MacroServer & Door(s) (via Door different client applications access the MacroServer engine)
- Allows sequential & simultaneous execution of macros
- Various GUI and CLI clients can control the macro execution

MacroServer DS

MacroServer

Doors

Macros

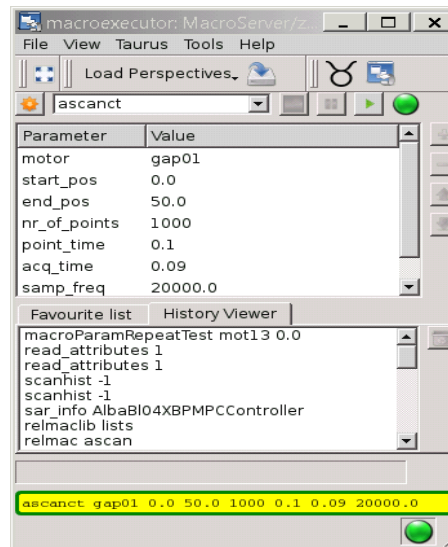
```
Door_zreszela_1 [13]: lsmeas
Active Name Timer Experi. channels
-----
* mg_odedtest oned01 oned01
  mntgrp01 ct01 ct01, ct02, ct03, ct04
  mntgrp02 ct01 ct01, ct02
  mntgrp03 ct01 ct01, ct02, ct03, ct04, oned01

Door_zreszela_1 [14]: lsm
Name Type Controller Axis
-----
gap01 PseudoMotor slitctrl01 1
icepap1302 Motor icepap13ctrl 2
mot01 Motor motctrl01 1
mot02 Motor motctrl01 2
mot03 Motor motctrl01 3
mot04 Motor motctrl01 4
mot05 Motor motctrl01 5
offset01 PseudoMotor slitctrl01 2
soprolec1 Motor soprolec_ctrl 1

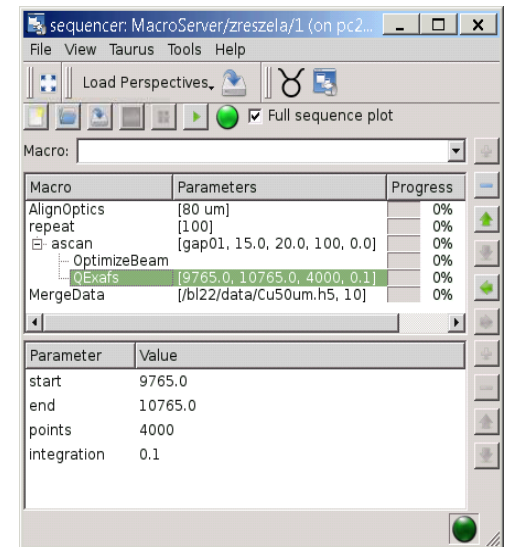
Door_zreszela_1 [15]: %ascan mot01 0 1 4 0.1
Operation will be saved in /home/zreszela/tmp/test.h5 (w5)
Scan #329 started at Sun Oct 12 13:43:27 2014. It will take at least 0:00:00.694422
Moving to start positions...
#Pt No mot01 ct01 ct02 ct03 ct04 dt
0 0 0.1 0.2 0.3 0.4 0.085824
1 0.25 0.1 0.2 0.3 0.4 0.249444
2 0.5 0.1 0.2 0.3 0.4 0.410941
3 0.75 0.1 0.2 0.3 0.4 0.570931
4 1 0.1 0.2 0.3 0.4 0.730435
Operation saved in /home/zreszela/tmp/test.h5 (w5)
Scan #329 ended at Sun Oct 12 13:43:28 2014, taking 0:00:00.845693. Dead time 40.9%
(motion dead time 29.5%)

Door_zreszela_1 [16]: []
```

Spock (based on IPython)



MacroExecutor



Sequencer

Hooks

```
Door_1 [8]: loop 0 10 3
Starting loop
At step 0
running hook with hints=['pre-acq']
En hook 1
At step 3
running hook with hints=['pre-acq']
En hook 1
At step 6
running hook with hints=['pre-acq']
En hook 1
At step 9
running hook with hints=['pre-acq']
En hook 1
Finished loop
```

Input parameters & results & data

SPEC like commands

```
MS_zreszela_1
Macro libraries
  -> addmac lib test
  -> communication
  -> demo
  -> env
  -> expert
  -> ioregister
  -> libmac lib
  -> log test
  -> macrostatus_demo
  -> mca
  -> motor
  -> pcapac2014
  -> AlignOptics
  -> ask_number_of_points
  -> captain_hook
  -> hooked_dummyscan
  -> hooked_scan
  -> JO_plot
  -> MergeData
  -> OptimizeBeam
  -> QExafs
  -> pn test
  -> scan
  -> a2scan
  -> a2scanct
  -> a3scan
  -> a3scanct
  -> a4scan
  -> a4scanct
  -> anultiscan
  -> ascan
  -> ascanct
  -> ascanct
  -> ascanh
  -> d2scan
  -> d2scanct
  -> d3scan
  -> d3scanct
  -> d4scan
  -> d4scanct
  -> multiscan
  -> dscan
  -> dscanct
  -> fscan
  -> mesh
  -> meshic
  -> scanhist
  -> sequence
  -> standard
  -> _vm
  -> _vum
  -> ct
  -> mstate
  -> mv
  -> mvr
  -> pva
  -> pvm
  -> report
  -> set_lim
  -> set_lm
  -> set_pos
  -> set_user_pos
  -> settimer
  -> uct
  -> unv
  -> unvr
  -> wa
  -> vm
  -> vu
  -> vum
  -> tools

13
14
15 @macro([["moveable", Type.Moveable, None, "moveable to move"],
16         ["position", Type.Float, None, "absolute position"] ])
17 def move(self, moveable, position):
18     """This macro moves a motor to the specified position"""
19     moveable.move(position)
20     self.output("Motor ended at ", moveable.getPosition())
21
22 class Loop(Macro, Hookable):
23     """A macro that executes a for loop. It accepts hooks.
24     hints = { 'allowsHooks': ['pre-move', 'post-move', 'pre-acq', 'post-acq']
25     param_def = [['start', Type.Integer, None, 'start point'],
26                  ['stop', Type.Integer, None, 'end point'],
27                  ['step', Type.Integer, 1, 'step']]
28     result_def = [['result', Type.Integer, None, 'result']]
29
30     def hook1(self):
31         self.output("In hook 1")
32
33     def run(self, start, stop, step):
34         self.info("Starting loop")
35         self.hooks = [ (self.hook1, ["pre-acq"])]
36         for i in xrange(start, stop, step):
37             self.output("At step %d" % i)
38             self.flushOutput()
39             for hook, hints in self.hooks:
40                 hook()
41         self.info("Finished loop")
42         return i
43
44 class hooked_scan(Macro):
45     """An example on how to attach hooks to the various hook points of a scan.
46     param_def = [
47         ['motor', Type.Moveable, None, 'Motor to move'],
48         ['start_pos', Type.Float, None, 'Scan start position'],
49         ['end_pos', Type.Float, None, 'Scan final position'],
50         ['nr_interv', Type.Integer, None, 'Number of scan intervals'],
51         ['integ_time', Type.Float, None, 'Integration time']]
52
53     def hook1(self):
54         self.info("hook1 execution")
55
56     def run(self, mot, start, end, nr, intt):
57         self.ascan, pars = self.createMacro("ascan", mot, start, end, nr, intt)
58         self.ascan.hooks = [(self.hook1, ["pre-acq"])]
59         self.runMacro(ascan)
60
61     @property
62     def data(self):
63         return self.ascan.data
64
65 @macro()
66 def ask_number_of_points(self):
67     """Asks user for the number of points"""
68     nb_points = self.input("How many points?", data_type=Type.Integer)
69
70 @macro()
71 def JO_plot(self):
72     """Sample JO at linspace(0, 20, 200)"""
73     x = linspace(0, 20, 200)
74     y = j0(x)
75     x1 = x[::10]
76     y1 = map(j0, x1)
77     self.pyplot.plot(x, y, label='J0(x)')
78     self.pyplot.plot(x1, y1, 'ro', label='J0(0*(integ)(x))')
79     self.pyplot.title(r'Verify J0(x)=\frac{1}{\pi}\int_0^{\pi}\cos(x\sin\phi)d\phi')
80     self.pyplot.xlabel('x')
81     self.pyplot.legend()
82
83
84
85
86
87
```

Adding, editing macros at runtime

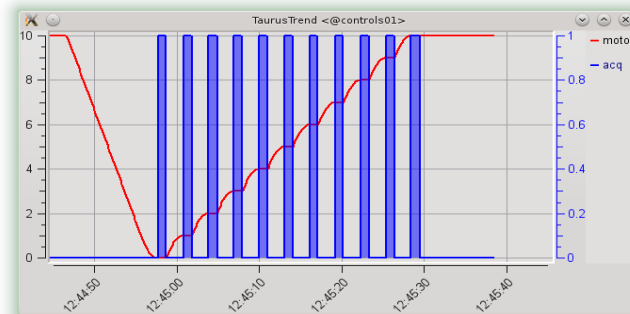
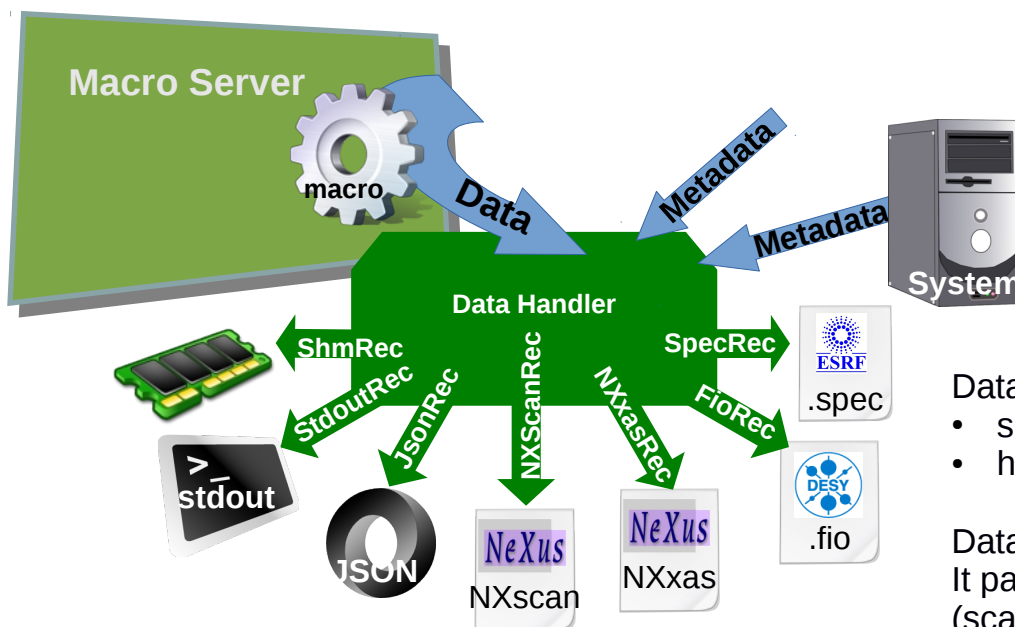
Interactive macros

Plotting

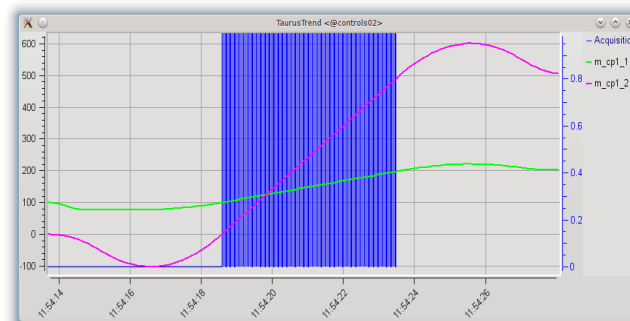
The sardanaeditor widget with some exemplary macros

The scan process consist of **data acquisition** of the involved experimental channels while **varying the scanning variable** (typically a moveable object).

- Sardana offers various scanning modes: step, hybrid, continuous.
- Turnkey n-dimensional scan macros exists.
- Motion, data acquisition & storage are synchronized and optimized.



Motion & acquisition during the step scan.

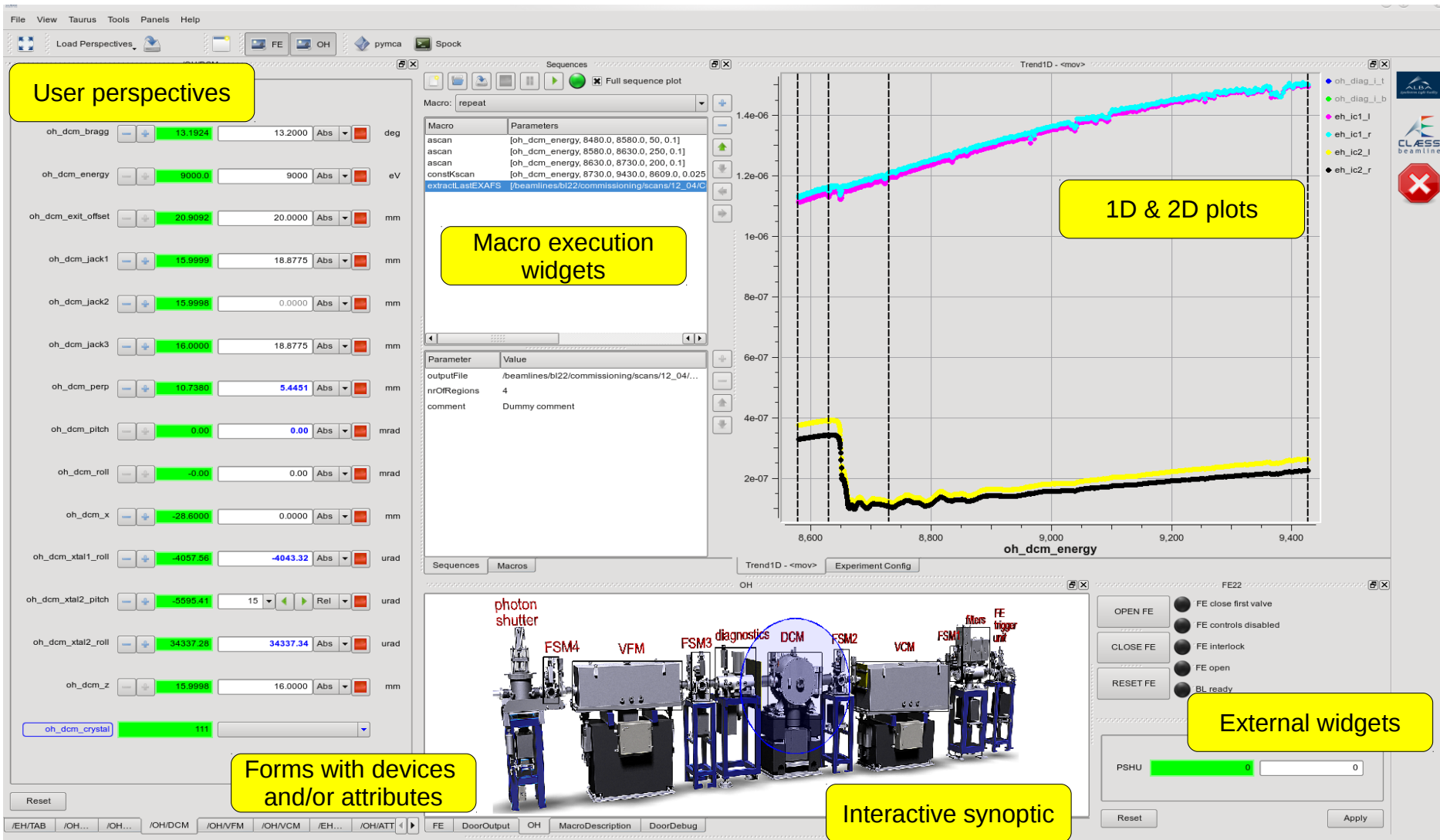


Motion & acquisition during the continuous scan.

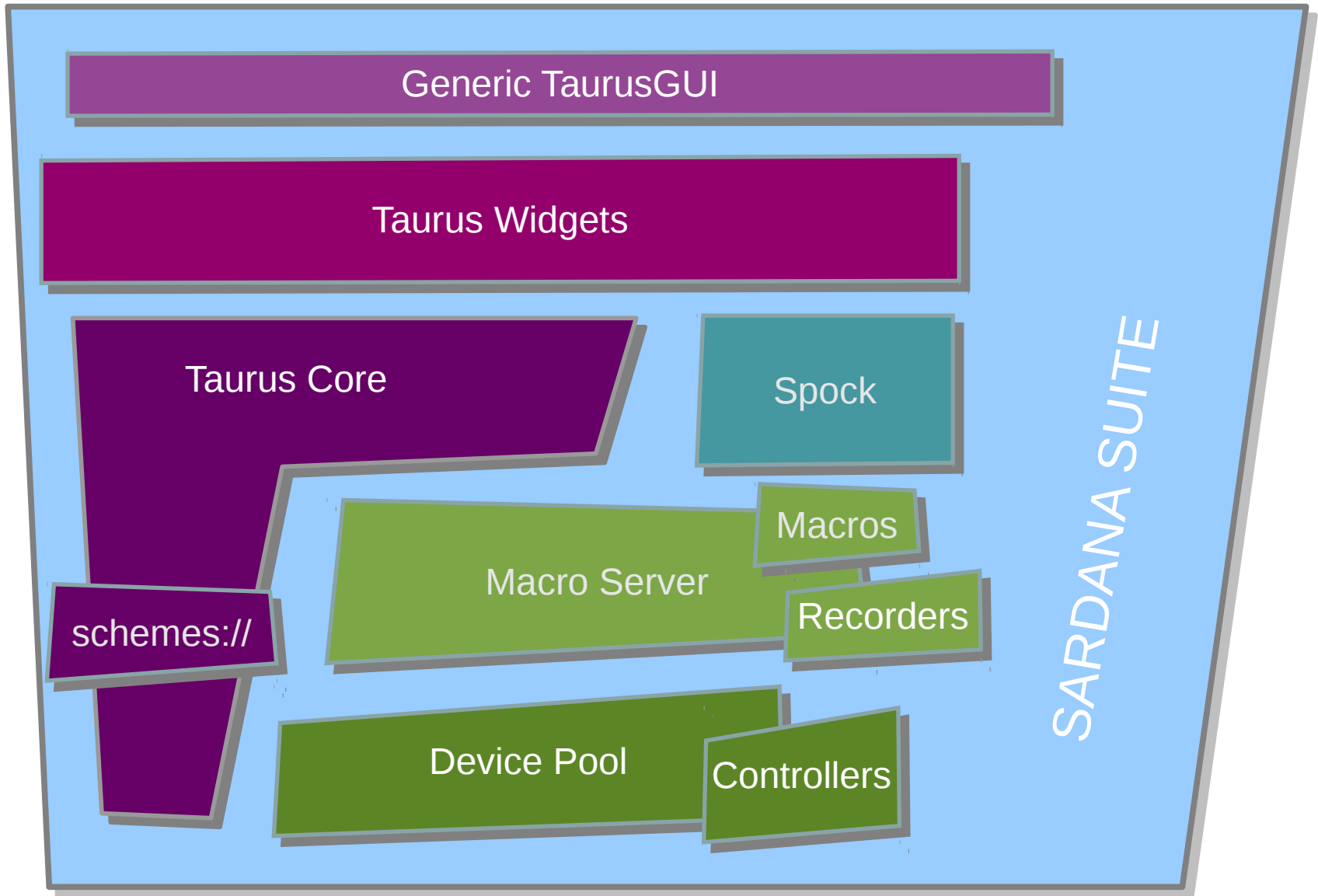
Data handing in Sardana:

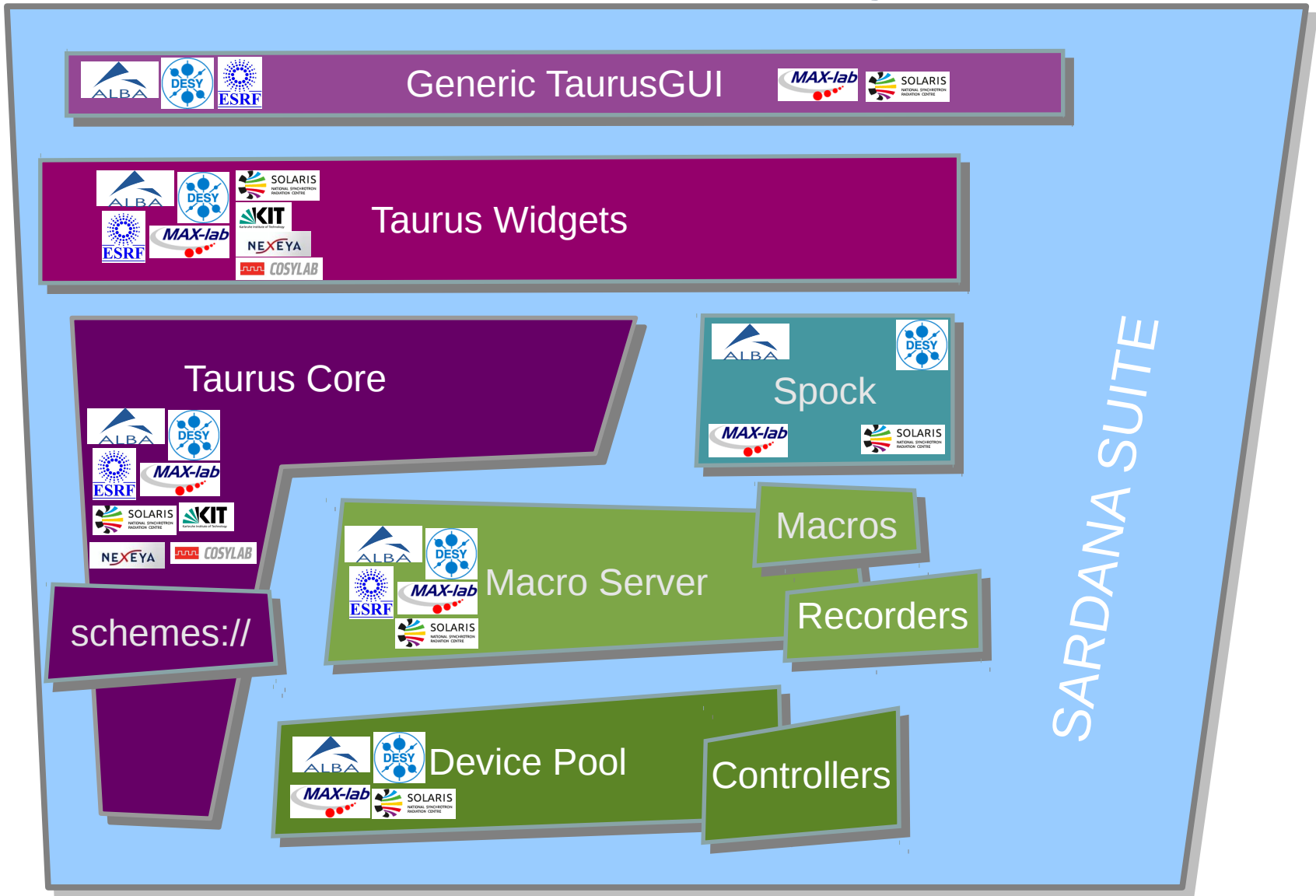
- scan data can be stored in **different formats**
- handled by one or more **optional recorders**

Data handler receives and gathers *metadata*. It passes data and the experiment metadata from the (scanning) macro and despatch them to the destination in the correct format: file (nexus, spec, etc.), console output or data post-processing program, etc.



Beamline 22 GUI developed with Generic TaurusGUI Framework





Current and future developments

General

Plug-in system   

Use ReadThe Docs for documentation   

Create the h5file:// scheme 

Support Python3 

Taurus

Taurus core Tango independent    

Use of Pint Quantities   

Multi-models (adapter pattern)    

Replace Qwt for plots  

Allow external logging (SEP8)    

Use of standard Enum (SEP12)   

Direct registering of Icons (avoid resource files)  

Merge TaurusConfiguration into TaurusAttribute   

Support PySide and Qt5 

Generic support for archiving values   

SVG synoptic view   

Sardana

Generic continuous scans (SEP6)   

Trigger/Gate elements (SEP6) 

Controllers with heterogeneous elements 

Diffractometer control (SEP4)    

Pseudo roles on instance level   

2D detectors: Lima integration (SEP2)   

Configuration tool   

Easier debugging  

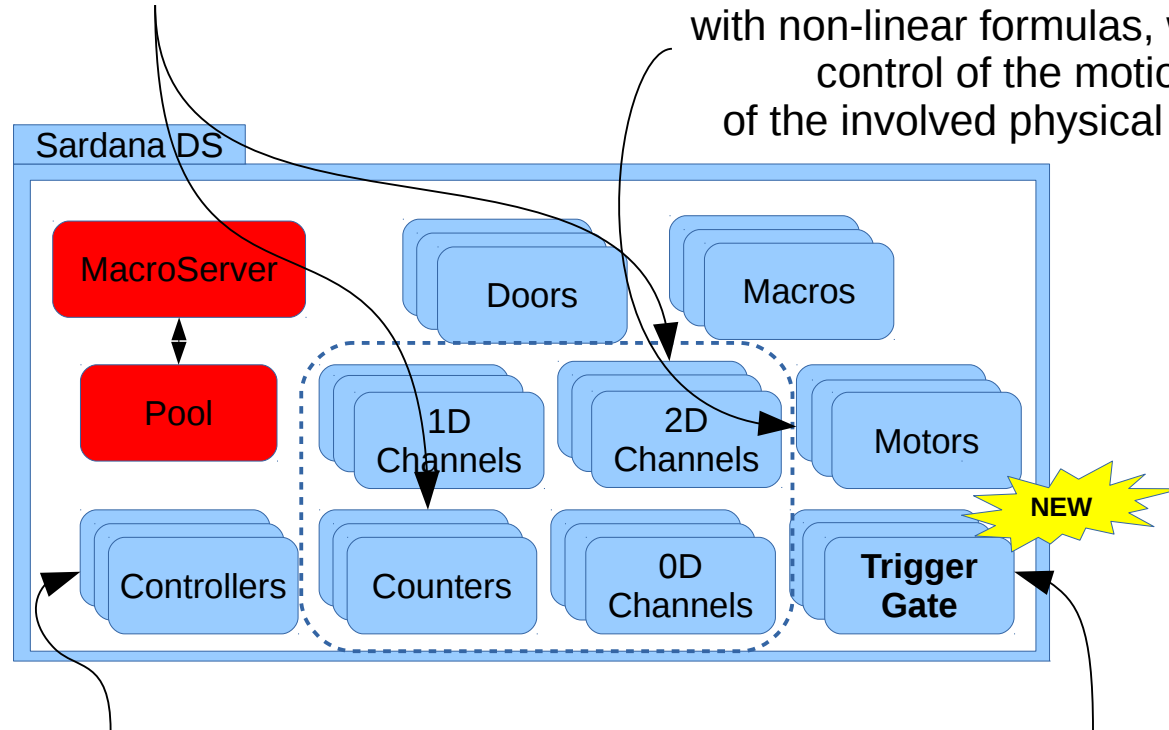
Acquisition with multiple Pools 

Refactor experiment configuration  

Generic Continuous Scans

Transparency between different synchronization modes of the experimental channels must be achieved.
(timestamp used for triggering and data merging)

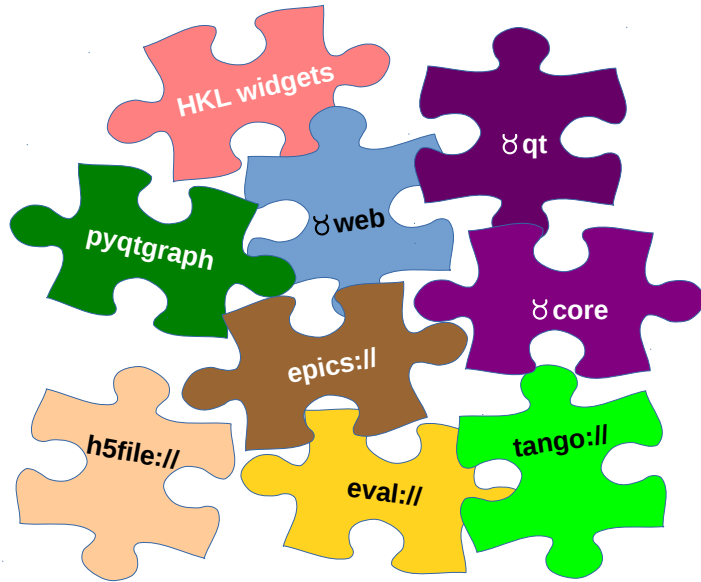
To ensure scans of the pseudomotors with non-linear formulas, with a constant rate, control of the motion trajectories of the involved physical motors is required.



The Controller concept needs to be extended to allow handling elements of different types (multipurpose hardware)

A new element type – trigger/gate is necessary. Its interfaces must allow configurations of equidistant or arbitrary sequences of events in time or distance domains.

Generic plugin system

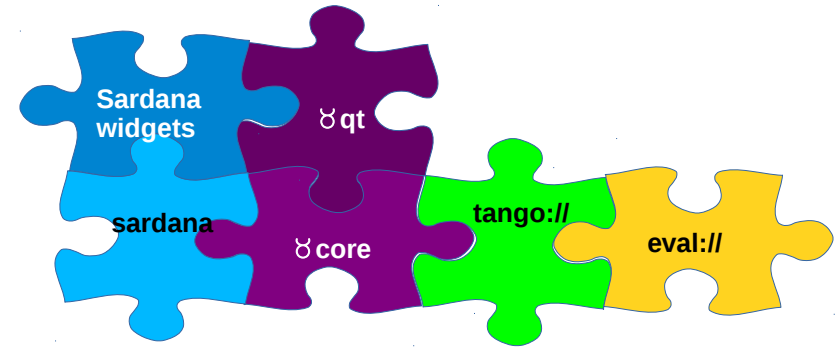


Plugins will make Sardana & Taurus...

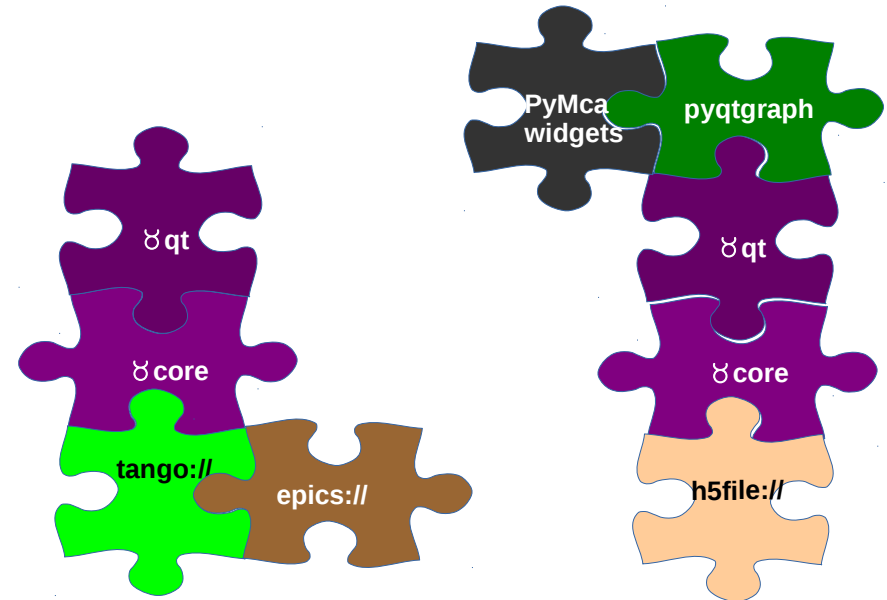
- Light: most dependencies optional
- Extendable for user specific need
- Taurus usable as a library for data analysis GUIs (not connected to control system at all)

What could be a plugin in Sardana & Taurus:

- Schemas, Widgets, Taurus core extensions...
- Macros, Controllers, Recorders...



Example: Taurus+Sardana as we use it now in ALBA



Example: Controlling a mixed Tango+EPICS environment

Example: Taurus for Data Analysis (no control system)

- Alba Controls Group
- Alba scientists
- Contributors & Experts from:
 - Desy
 - ESRF
 - MaxLab
 - Solaris
 - Soleil

