Elettra Sincrotrone Trieste

Elettra
Sincrotrone
Trieste

TANGO

School on TANGO Controls system

Basics of TANGO

Lorenzo Pivetta
Claudio Scafuri
Graziano Scalamera

http://www.tango-controls.org

Prerequisites

To better understand the training a background on the following arguments is desirable:

- Programming language
- Object oriented programming
- Linux/UNIX operating system
- Networking
- Control systems

Outline

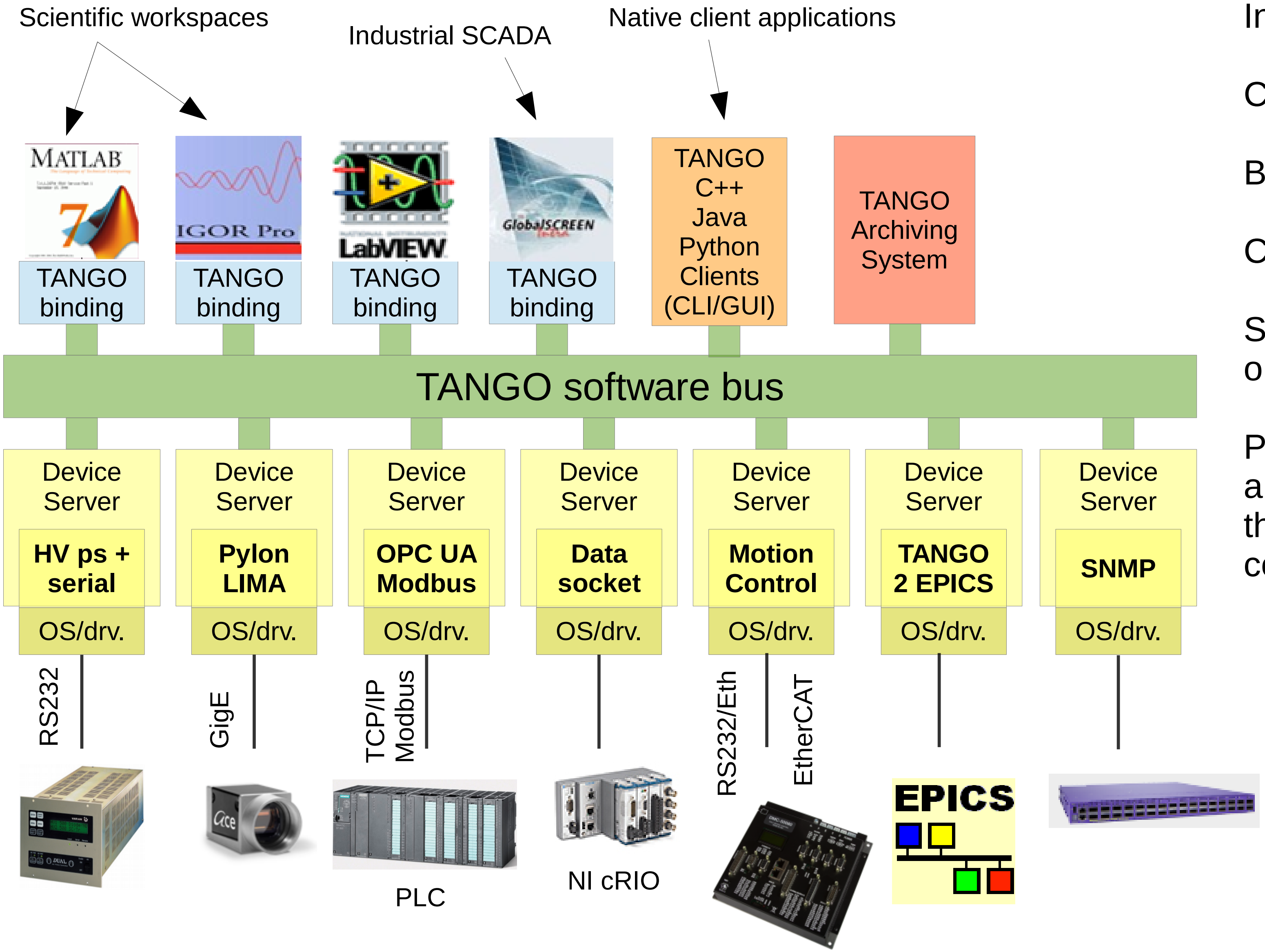
1 - What is TANGO?
Language/OS/Compilers
CORBA and ZeroMQ
TANGO device and device server
TANGO Database
Communication models
Multicast
Polling
Events
Alarms
Groups
TANGO ACL
Logging system
Historical DataBase

2 - TANGO architecture
Device hierarchy
TANGO domains

3 - TANGO configuration/tools
Jive
Starter/Astor
Pogo
TANGO installation
Client basics

4 – Examples
Test device

What is TANGO?



Scientific workspaces

Industrial SCADA

Native client applications

TANGO binding

TANGO binding

TANGO binding

TANGO binding

TANGO C++ Java Python Clients (CLI/GUI)

TANGO Archiving System

TANGO software bus

Device Server	Device Server	Device Server	Device Server	Device Server	Device Server	Device Server
HV ps + serial	**Pylon LIMA**	**OPC UA Modbus**	**Data socket**	**Motion Control**	**TANGO 2 EPICS**	**SNMP**
OS/drv.	OS/drv.	OS/drv.	OS/drv.	OS/drv.	OS/drv.	OS/drv.

RS232

GigE

TCP/IP

Modbus

RS232/Eth

EtherCAT

PLC

NI cRIO

In short:

Control system framework

Based on CORBA and ZMQ

Centralized config. database

Software bus for distributed objects

Provides unified interface to all equipments hiding **how** they are connected/managed

The TANGO collaboration

TANGO collaboration history

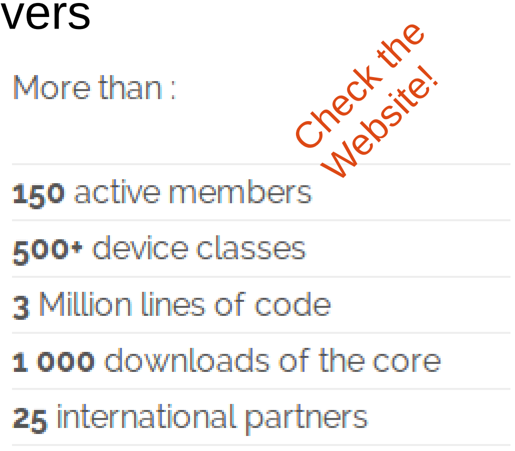
- started in 1999 at the ESRF
- in 2000 SOLEIL joins ESRF to develop TANGO
- end 2003 ELETTRA joins the club
- 2004 ALBA also joins
- 2006 ANKA
- 2007 – 2012 Desy, MaxLab, FRM II, SOLARIS
- 2013 – 2014 ELI-Beamlines, ELI-ALPS, University of Szeged, INAF
- 2015 – 2016 draft, discussion and approval of Collaboration Contract

TANGO Collaboration Contract signed by institute directors in 2016

Yearly basis collaboration meetings (next June, INAF Trieste, Italy)
Committer member: commit source code to TANGO Controls core
Collaborator member: write and share TANGO device servers

Nevertheless, TANGO is **free for anyone to use**

Mailing list, forum, web site...
http://www.tango-controls.org



More than :

150 active members

500+ device classes

3 Million lines of code

1 000 downloads of the core

25 international partners

Check the Website!

Elettra Sincrotrone Trieste

Language/OS/Compilers

TANGO release 9.2.2 (+ patches) (C++98, C++11)

Previous release TANGO 8.1.2.c (+patches)

Languages

Server side: C++, Java, Python
Client side: C++, Java, Python, Matlab, LabView, IgorPro, Panorama

OS – Linux (PREEMPT_RT, Xenomai hard real-time)

Architecture: x86, PPC, ARM
Compiler: gcc 3.3 – gcc 4.8

OS – Windows XP/Vista/7

Architecture: x86
Compiler: VC9, VC10, VC11

OS – MacOSX

Architecture: x86
Compiler: gcc 4.6 – gcc 4.8

Training focus on TANGO 8 with some info on TANGO 9

CORBA and ZeroMQ

CORBA – http://www.omg.org

- Common Object Request Broker Architecture specification
- Defines the ORB and the services available for all objects
- Uses an Interface Definition Language (IDL) and defines bindings between IDL and programming languages
- An Inter-operable Object Reference (IOR) identifies each object
- TANGO adopts omniORB for C++ and JacORB for Java
  - http://www.omniorb.sourceforge.net
  - http://www.jacorg.org

ZeroMQ, ZMQ, 0MQ – http://zeromq.org

- An embeddable networking library that acts like a concurrency framework
- Sockets that carry whole messages across various transports like in-process, inter-process, TCP and multicast
- Used for event-based communication in TANGO ≥ 8

Device

Everything which needs to be controlled is modeled as a Device

The Device is the core concept of TANGO

A Device can represent:

- an equipment (e.g. a power supply)
- a set of equipments (e.g. a set of 3 motors, x-y-z axes, driven by the same controller)
- a set of software functions
- a group of equipments constituting a subsystem

The modeling of the equipment, either hardware or software, is the first fundamental step when writing a TANGO device

- a TANGO device must be self-consistent
- must enable the access to all the features of the modeled device
- the limit of its responsibilities, meaning the separation of concerns, is clearly defined: **1 device = 1 service = 1 element of the system**
- the analogy with object-oriented programming is straightforward

Class/Device/Device Server

Class/Device/Device Server: three concepts closely related

- **TANGO Class**: a class defining the interface and implementing the device control or the implementation of a software algorithm

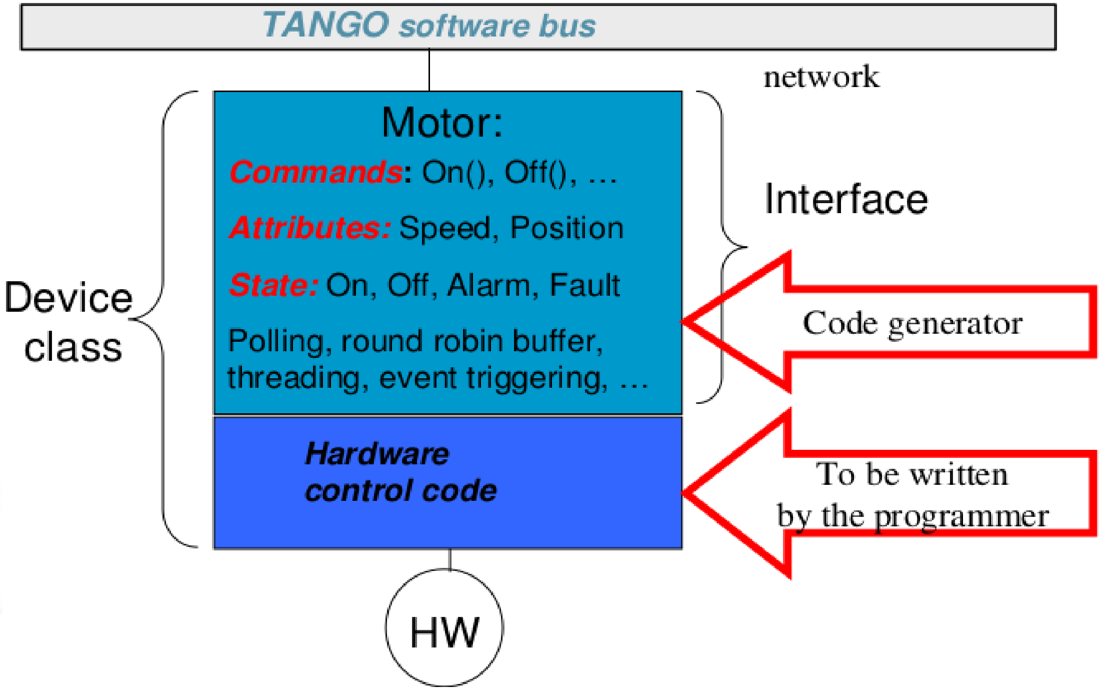
- **TANGO Device**: an instance of a TANGO Class giving access to the services of the class

- **TANGO Device Server**: the process in which one or more TANGO Classes are executed, making thus available one or more Tango Devices

Device Interface

Everything which needs to be controlled is modeled as a Device
Each Device is identified by the Fully Qualified Domain Name (FQDN)
tango://host:port/domain/family/member
Each Device belongs to a TANGO class that inherits from the same root class *Device_XImpl*
Every Device exposes the **same interface**:

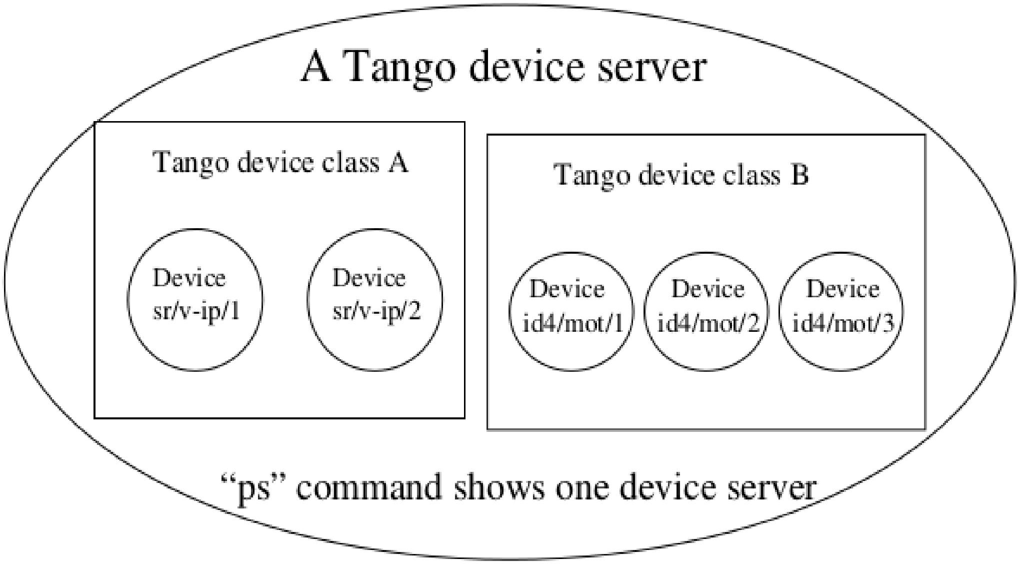
- **Command**(s): act on devices (e.g. power on)
- **Attribute**(s): set/get physical values (e.g. set/get motor position)
  - Can be *memorized*
  - Attribute properties: per-attribute configuration parameters (*)
  - State/Status: TANGO Device finite state machine value (also available as Commands) (*)
- **Properties:** config. Parameters
  - Attribute level
  - Device level
  - Class level
  - Free/Global


TANGO software bus
network
Motor:
Commands: On(), Off(), ...
Attributes: Speed, Position
State: On, Off, Alarm, Fault
Polling, round robin buffer, threading, event triggering, ...
Interface
Code generator
Device class
Hardware control code
To be written by the programmer
HW

(*) more to come on these

Device Server

- The Device Server is the process where the TANGO class(es) run
- Device Server configuration is stored into the TANGO database (MySQL)
- Device number and names for a TANGO class are defined within the database, **not in the code**
- Which TANGO class(es) are part of a DS process is defined in the database **but also in the code**
- The Device Server **can** host several TANGO classes, each class **can** be instantiated several times *...but be careful with code or DLLs not thread safe*



Device Server

Startup sequence

Device server

1 - the TANGO device server contacts the TANGO database to know which devices it has to create and manage based on the **instance** specified

2 - the TANGO device server registers device(s) IOR

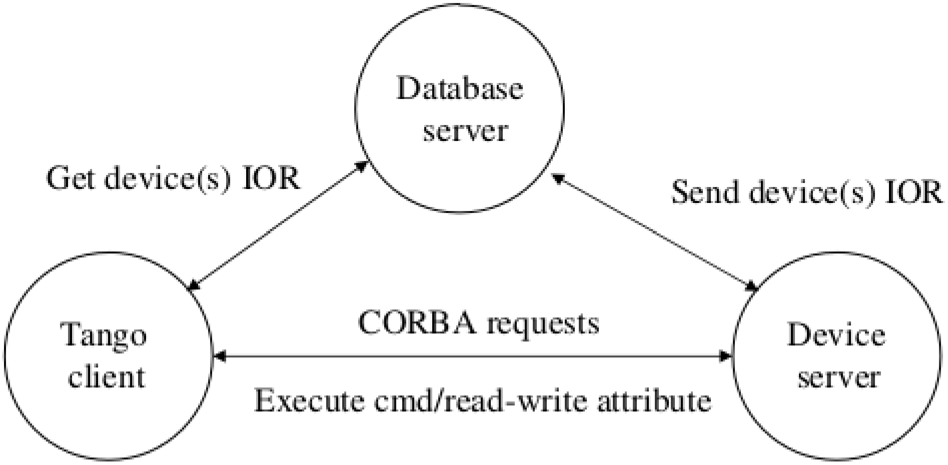
Client

1 - the client asks the TANGO database for device IOR

2 - the client connects to the device server

The TANGO database is involved, and necessary, only during the connection phase (*)

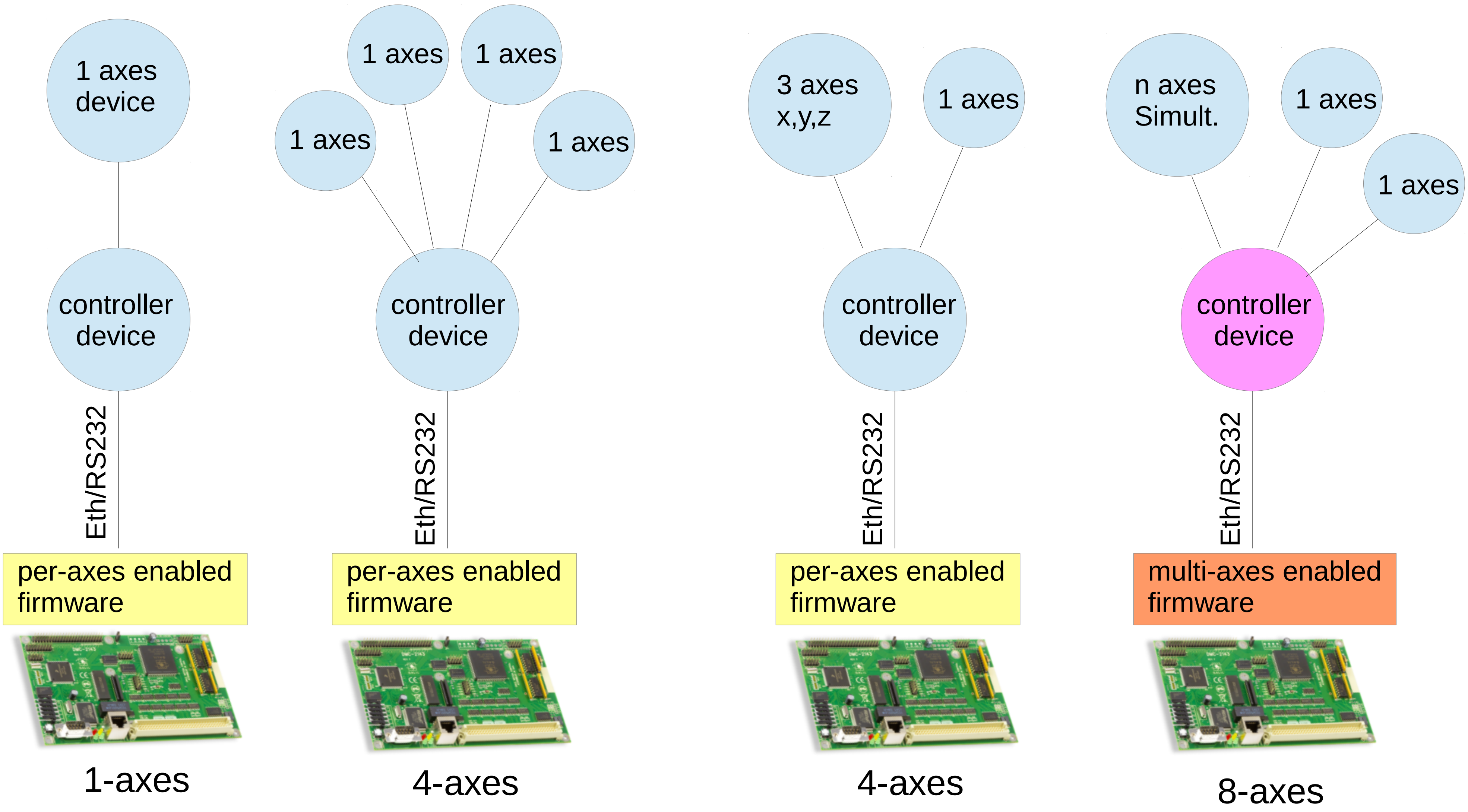
(*) exception: memorized attributes



client-server

Device patterns

Stepper motor controller example
Ethernet/RS232 1-8 axes, single control interface



1 axes
device

controller
device

Eth/RS232

per-axes enabled
firmware

1-axes

1 axes

1 axes

1 axes

1 axes

controller
device

Eth/RS232

per-axes enabled
firmware

4-axes

3 axes
x,y,z

1 axes

controller
device

Eth/RS232

per-axes enabled
firmware

4-axes

n axes
Simult.

1 axes

1 axes

controller
device

Eth/RS232

multi-axes enabled
firmware

8-axes

Naming

Tango uses a well defined naming scheme, each device has a unique *name* within the control system. The *name* is the key to get access to the device

- the device *name* is a character string(a..z,0..9), composed by three fields separated by /
- the three fields are known as **domain** , **family** and **member**

domain/family/member

Hierarchical view , tree structure

Operations on device names are case insensitive within Tango:

LH/PSQ/1 **lh/psq/1** **Lh/PsQ/1**

are equivalent.

Within Tango decices names are sorted alphabtically.
pay attention to numbers: 10 is sorted before 2!

Naming

Tango names can be any valid string : **sasackjn/kljd122334/hdhsah**

The **naming convention** is part of the desig of a Tango control system

names must be meaningful to all people involved

names must be compatible with names used by other departments / installations,
for example technical drawings, plant schematics.

what and ***where*** should be indicated in names

what: the type of device, its function , device Class, ...
where: the location of the device, geographical or logical

A good naming convention allows one to refer to groups of devices using *widcards*

example:
sr/power_supply/* all power supplys of Elettra magnets
sr/power_supply/psq1_* all power_supplys of Elettra type 1 quadrupole magnets

see Tango manual , appendix C, for more details about naming

TANGO Database

- Centralized storage for control system (TANGO device) configuration parameters and for persistent data.
- Based on MySQL database engine
- Centralized service for establishing connections (name resolution)
- it is a special Tango Device

A minimum TANGO system - to run a TANGO control system you need:
- a running MySQL database
- the TANGO Database device server listening on a fixed port
- the TANGO_HOST environment variable is used by clients/servers to know on which host and port the Database server is running:

```
TANGO_HOST=tango://hostname.full.domain.name:port
```

short form

```
TANGO_HOST=hostname:port
```

note: you can run a small control system without a database using static configurations stored in files.

Device Command(s)

Commands may have **zero** or **one input** and **zero** or **one output** argument

Supported argument data types are:

- void
- boolean, short, long, long64, float, double, string, unsigned short, unsigned long, unsigned long64
- homogeneous array of the former data types
- state
- encoded (structure with 2 fields: a string and an array of unsigned char)

Commands are typically used for starting actions on devices or change their operating state
example: ON(), OFF(), ENABLE(),...

Device Attribute(s)

Twelve data types:

- boolean, unsigned char, short, unsigned short, long, long64, unsigned long, unsigned long64, float, double, string
- array of the former
- array of strings and values
- state/status
- encoded (TANGO >=8): images encode in jpg, 8/16 bit gray, 24 bit RGB

Three “access modes”:

- read, write, read-write

Three data formats:

- scalar (single value)
- spectrum (one dimensional array)
- image (bi-dimensional array)

When you **read** an attribute you receive also some metadata:

- the attribute data (value, and also w_value for r/w attributes)
- the attribute quality factor (VALID, INVALID, CHANGING, WARNING, ALARM)
- the attribute timestamp
- the name
- the dimension

When you **write** an attribute you send:

- the desired attribute data (value)
- the attribute name

Device Attribute Properties

Each Attribute configuration is defined by its **Properties**; five type available:

hard-coded

name, data_type, data_format, writable, max_dim_x, max_dim_y, writable_attr_name, display_level

GUI parameters

Description, Label, Unit, Standard_unit, Display_unit, Format (C++ or printf)

Range (for writable attributes)

min_value, max_value

Alarm parameters (*)

min_alarm, max_alarm, min_warning, max_warning, delta_t, delta_val

Event parameters (*)

change event: absolute, relative
archive event: absolute, relative, period
periodic event: period

Network calls `get_attribute_config/set_attribute_config` allow clients to access configuration

(*) More to come on Alarm, Event and attribute configuration

Device State/Status

TANGO defines a couple of special Commands/Attributes named **State** and **Status**

A set of **14 device State** (enum) is available:

ON, OFF, CLOSE, OPEN, INSERT, EXTRACT, MOVING, STANDBY, FAULT, INIT, RUNNING, ALARM, DISABLE, UNKNOWN

it is synthetic information about the of the device. Accessibility of device attributes and commands may be forbidden in some of the States (State Machine). Machine readable.

Status string info describing the State; managed by the programmer. Its main use is to provide human readable messages.

Device State is not easily extensible/customizable in TANGO 8 (nor in TANGO 9)
If you want to add additional values to the enum you need to modify the IDL; this implies a new IDL release and a new Device implementation class.

Properties

Properties can be thought as device configuration parameters
Stored into the TANGO Database
You can define properties at

- object level (free properties)
- class level
- device level

Types for scalar property

boolean, short, unisgned short, long, unsigned long, float, double, string

Types for array property

short, long, float, double, string

Algorithm to assign default property value:

```
/IF/ class property has a default value
    property = class property default value
/ENDIF/
/IF/ class property is defined in db
    property = class property as found in db
/ENDIF/
/IF/ device property has a default value
    property = device property default value
/ENDIF/
/IF/ device property is defined in db
    property = device property as found in db
/ENDIF/
```

Device Hierarchy

A TANGO control system ~~can~~ **must** be hierarchically (logically) organized

Devices associated with hardware equipments usually live at lower level

Higher level devices aim to:

- abstract functionalities from mechanisms
- group similar devices
- group devices into subsystems
- implement “abstract” features (e.g. processing)
- implement services based on many low level devices (e.g. alarms)

Higher level devices are
clients of lower level devices!

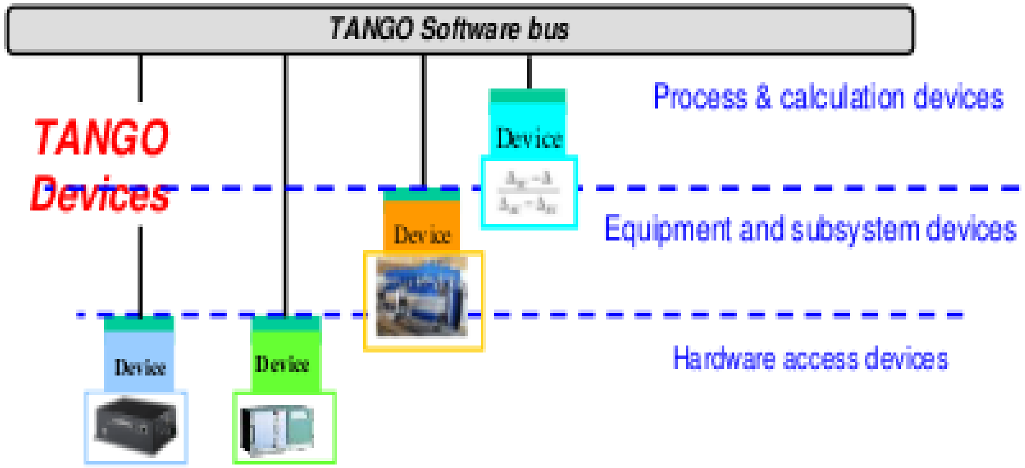
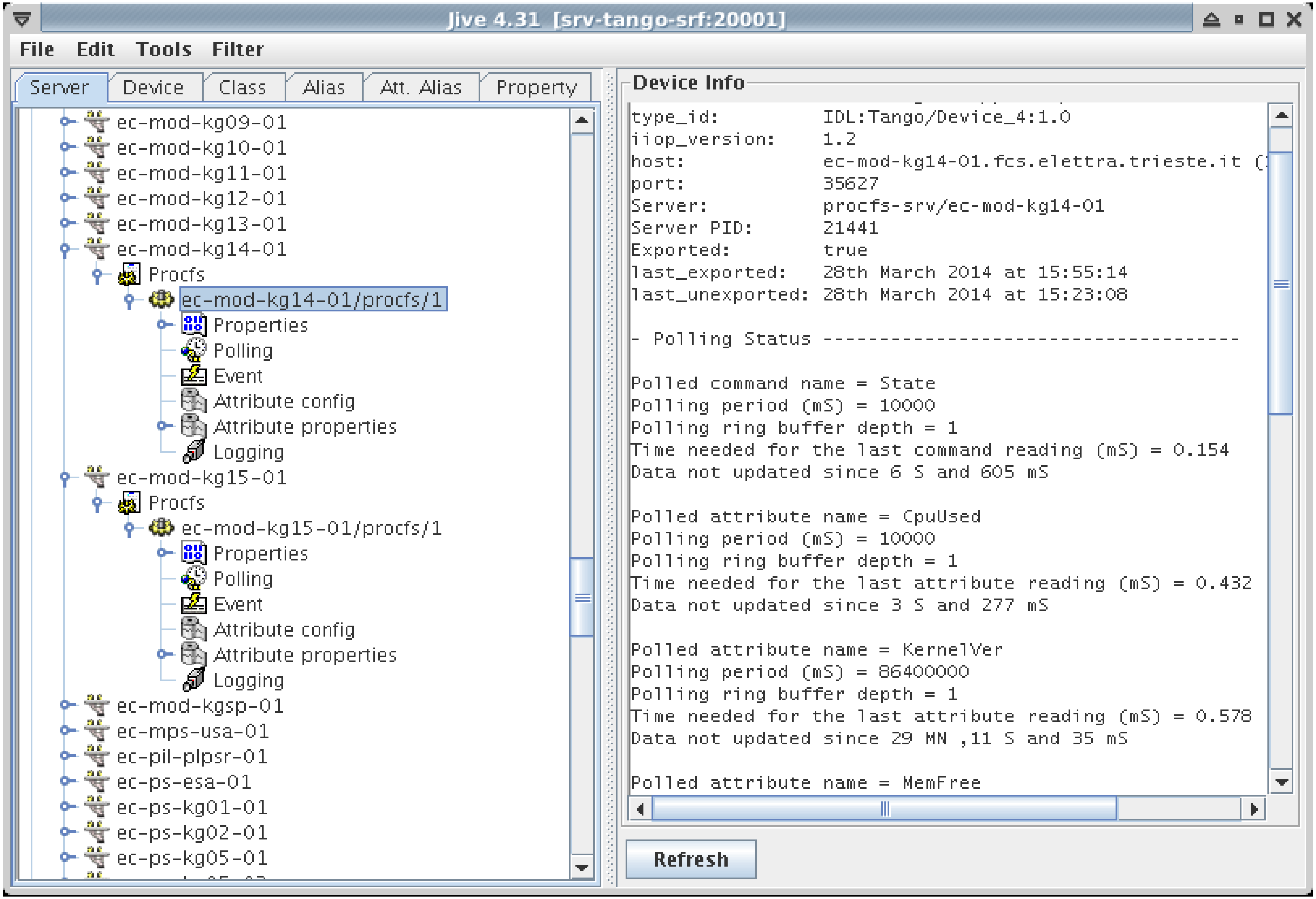


Figure 1 : The software bus view of devices

Administration: Jive

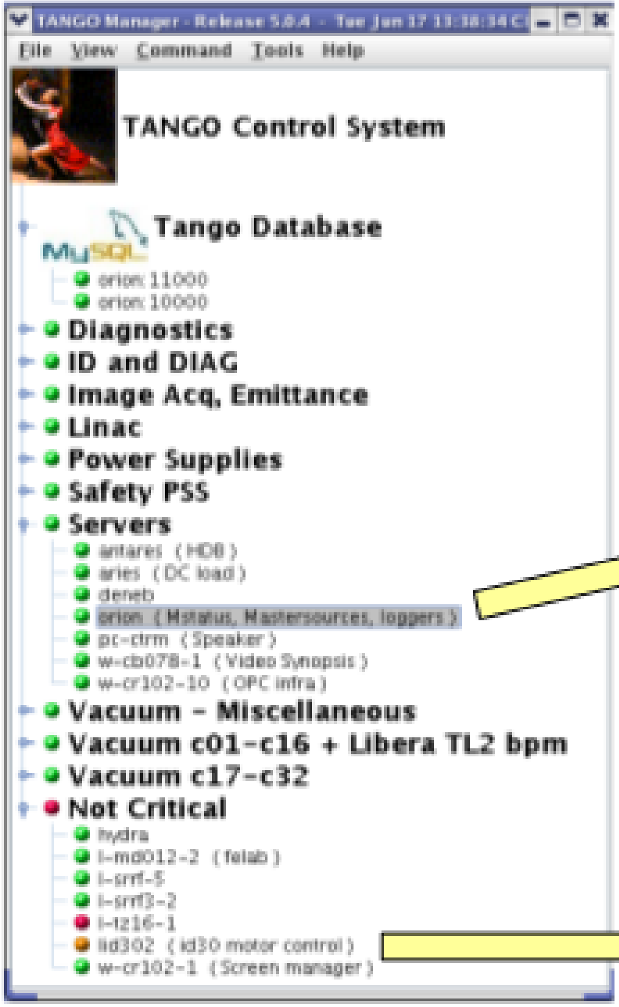
TANGO database browser and device configuration/administration/testing tool



```
type_id:          IDL:Tango/Device_4:1.0
iiop_version:     1.2
host:             ec-mod-kg14-01.fcs.elettra.trieste.it (
port:             35627
Server:           procfs-srv/ec-mod-kg14-01
Server PID:       21441
Exported:         true
last_exported:    28th March 2014 at 15:55:14
last_unexported:  28th March 2014 at 15:23:08

- Polling Status -------------------------------------

Polled command name = State
Polling period (mS) = 10000
Polling ring buffer depth = 1
Time needed for the last command reading (mS) = 0.154
Data not updated since 6 S and 605 mS

Polled attribute name = CpuUsed
Polling period (mS) = 10000
Polling ring buffer depth = 1
Time needed for the last attribute reading (mS) = 0.432
Data not updated since 3 S and 277 mS

Polled attribute name = KernelVer
Polling period (mS) = 86400000
Polling ring buffer depth = 1
Time needed for the last attribute reading (mS) = 0.578
Data not updated since 29 MN ,11 S and 35 mS

Polled attribute name = MemFree
```

Administration: Starter/Astor

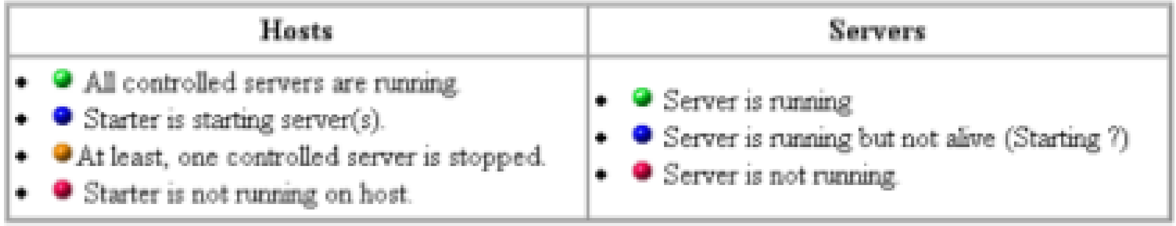
Starter: TANGO Device Server to manage device servers on hosts
Astor: control system manager GUI



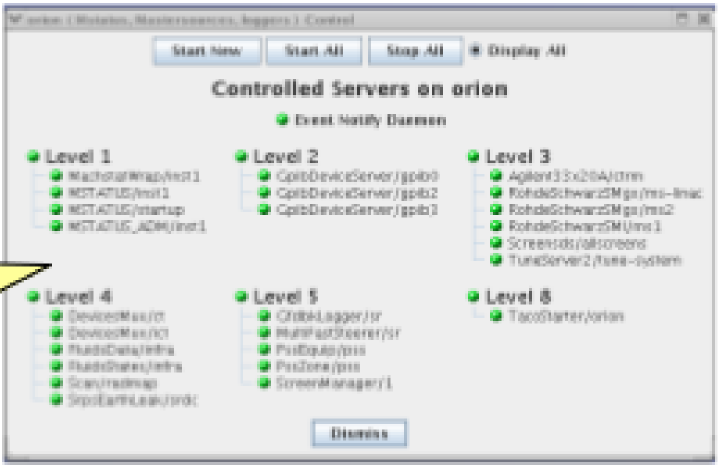
TANGO Manager - Release 5.0.4 - Tue Jan 17 11:38:34 C

TANGO Control System

- Tango Database
  - orion: 11000
  - orion: 10000
- Diagnostics
- ID and DIAG
- Image Acq, Emittance
- Linac
- Power Supplies
- Safety PSS
- Servers
  - antares (HDB)
  - aries (DC load)
  - deneb
  - orion (Mstatus, Mastersources, loggers)
  - pc-ctrm (Speaker)
  - w-cb078-1 (Video Synopsis)
  - w-cr102-10 (OPC infra)
- Vacuum - Miscellaneous
- Vacuum c01-c16 + Libera TL2 bpm
- Vacuum c17-c32
- Not Critical
  - hydra
  - l-md012-2 (felab)
  - l-srrf-5
  - l-srrf3-2
  - l-tz16-1
  - lid302 (id30 motor control)
  - w-cr102-1 (Screen manager)



Hosts	Servers
• 🟢 All controlled servers are running. • 🔵 Starter is starting server(s). • 🟠 At least, one controlled server is stopped. • 🔴 Starter is not running on host.	• 🟢 Server is running • 🔵 Server is running but not alive (Starting ?) • 🔴 Server is not running

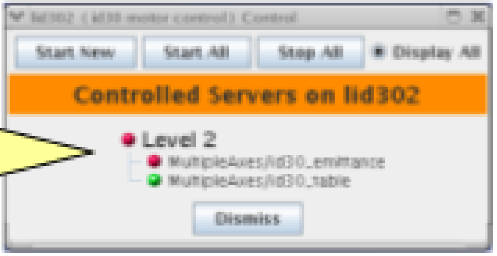


orion (Mstatus, Mastersources, loggers) Control

Controlled Servers on orion

Event Notify Daemon

- Level 1
  - MachstatWrap/inst1
  - MSTATUS/inst1
  - MSTATUS/startup
  - MSTATUS_ADM/inst1
- Level 2
  - GpibDeviceServer/gpib0
  - GpibDeviceServer/gpib2
  - GpibDeviceServer/gpib3
- Level 3
  - Agilent33x20A/ctrm
  - RohdeSchwarzSMgx/ms-linac
  - RohdeSchwarzSMgx/ms2
  - RohdeSchwarzSMU/ms1
  - Screensds/allscreens
  - TuneServer2/tune-system
- Level 4
  - DevicesMux/ct
  - DevicesMux/ct
  - FluidsData/infra
  - FluidsStates/infra
  - Scan/radmap
  - SrpEarthLeak/srdc
- Level 5
  - GlobalLogger/sr
  - MultiFastSteerer/sr
  - PssEquip/pss
  - PssZone/pss
  - ScreenManager/1
- Level 8
  - TacoStarter/orion



lid302 (id30 motor control) Control

Controlled Servers on lid302

- Level 2
  - MultipleAxes/id30_emittance
  - MultipleAxes/id30_table

TANGO ACL

Two kind of users (identified by system login name):

- users defined in the ACL
- users not defined in the ACL → rights fall below “All users”

Two kind of rights, at host **and** device level:

- Read (+ optional **per-class** allowed commands)
- Write

taurel

- write to sr/d-ct/01 and fe/*/* only from pcantares
- read all other devices only from pcantares

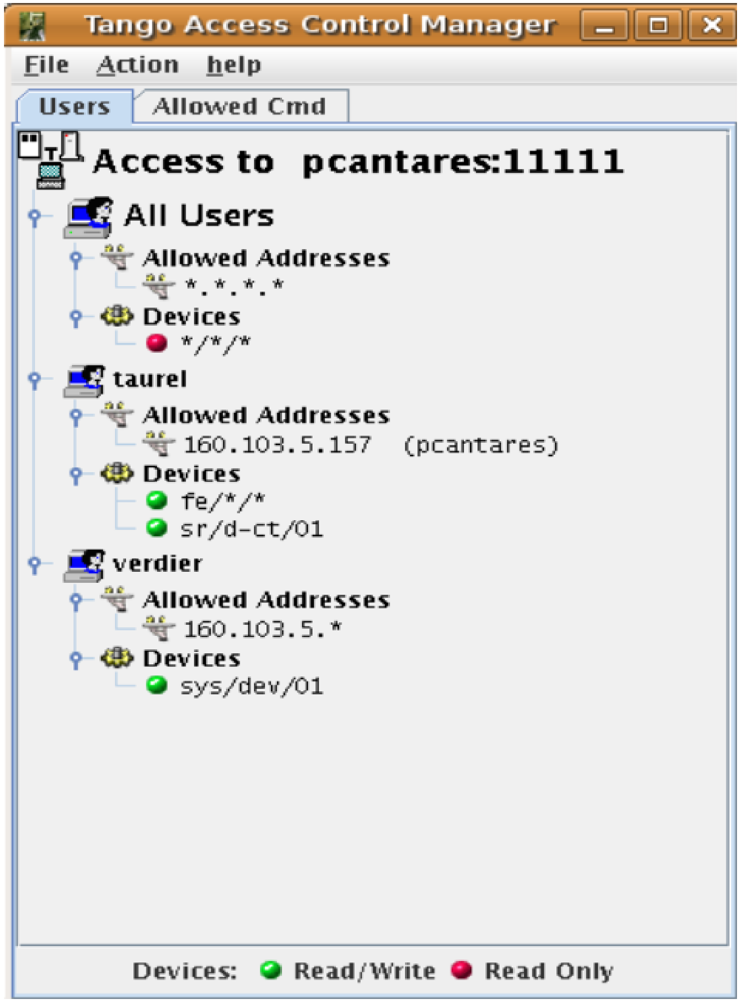
verdier

- write to sys/dev/01 from any host on 160.103.5.0/24 subnet
- read all other devices from the same subnet

all users

- read-only access from any host

Advice: TANGO ACL provides **basic** access control and can be bypassed; it's basically meant to avoid mistakes



Access to pcantares:11111

- All Users
  - Allowed Addresses
    - *.*.*.*
  - Devices
    - 🔴 */*/*
- taurel
  - Allowed Addresses
    - 160.103.5.157 (pcantares)
  - Devices
    - 🟢 fe/*/*
    - 🟢 sr/d-ct/01
- verdier
  - Allowed Addresses
    - 160.103.5.*
  - Devices
    - 🟢 sys/dev/01

Devices: 🟢 Read/Write 🔴 Read Only

Writing a TANGO device class

GOAL: model in Tango a Skilift

IDEAS:

possible states
- working properly
- switched off
- in error condition

action needed
- switch on
- switch off
- recover from error condition

physical quantities
- speed of the skilift, should be possible to be changed
- wind speed, cannot control it, just read
- current position of each seat, just read

Writing a TANGO device class

SkiLift: the Tango Device Server Model

3 states

ON, **OFF**, **FAULT**

3 commands (without arguments)

On – to switch device ON

Off – to switch device OFF

Reset – to reset the device in case of FAULT

3 attributes

Speed – current speed

WindSpeed – current wind speed

SeatPos – seats position

Writing a TANGO device class

SkiLift: the Tango Device Server Model

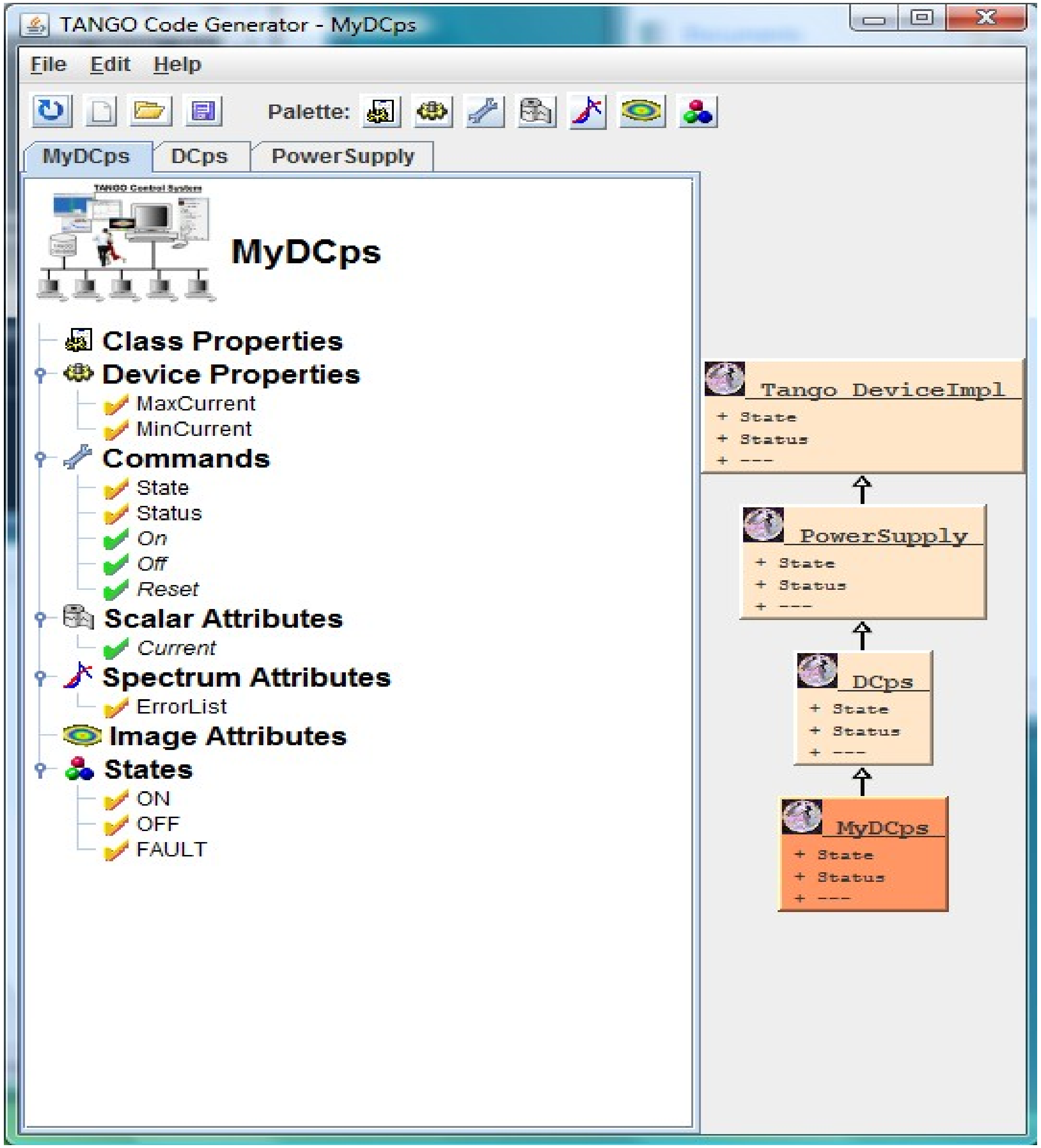
Tango Commands vs Attributes

Commands are actions

Attributes are physical quantities
- can have labels, units, conversion factors
- can have range of validity, alarm
- can have user defined properties
- can have a quality (VALID, INVALID, ...)
- can be memorized
- can generate events
- can be archived

For example the Tango Device Server of a motor
- should have a R/W Position Attribute
- and should not have Get_Position / GoTo_Position commands
- but could have Forward / Backward commands

Writing a TANGO device class: Pogo



Pogo is a TANGO class generator

Generates C++, Java and Python Source code and html documentation

The class skeleton is saved in a .xmi file

Well defined areas for programmer's code

Writing a TANGO device class

Use **POGO** to design a SkiLift class with the following functionalities:

3 states

ON, OFF, FAULT

3 commands (without arguments)

On – to switch device ON

allowed only when switched OFF

Off – to switch device OFF

allowed only when switched ON

Reset – to reset the device in case of FAULT

allowed only when in FAULT

3 attributes

Speed – current speed

scalar, double, read-write, min = 0.0, max = 5.0, alarm >= 4.0

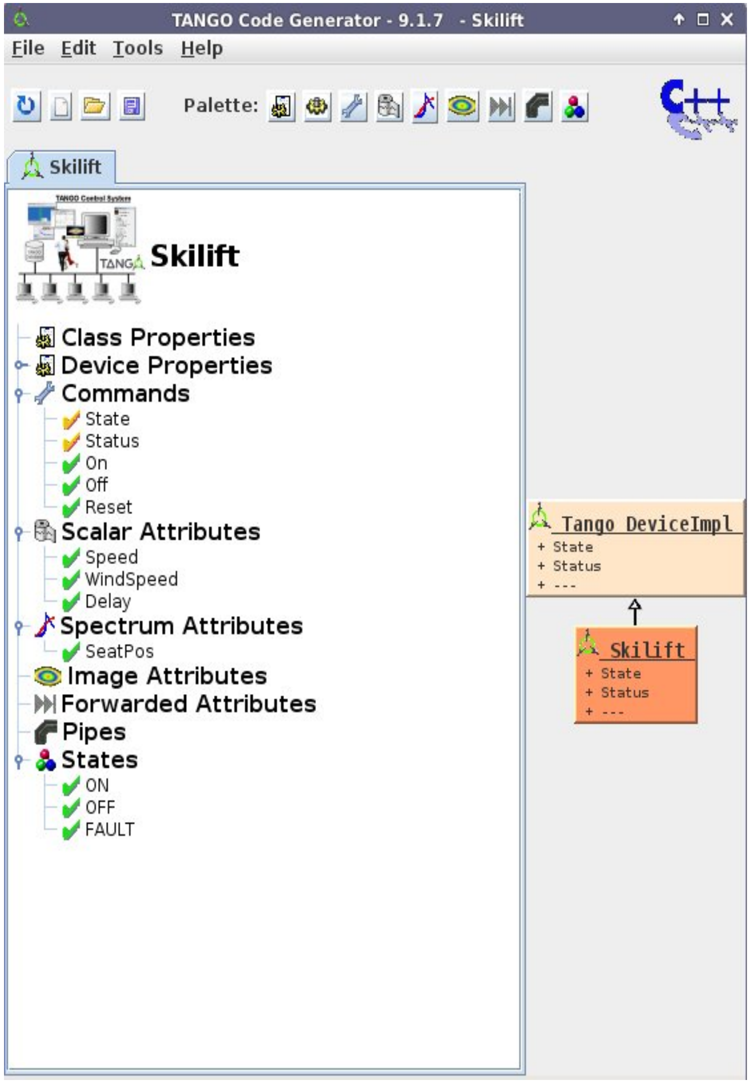
WindSpeed – current wind speed

scalar, double, read

SeatPos – seats position

spectrum, long, read

Generate the documentation



TANGO Code Generator - 9.1.7 - Skilift

File Edit Tools Help

Palette:

Skilift

Class Properties
Device Properties
Commands
- State
- Status
- On
- Off
- Reset

Scalar Attributes
- Speed
- WindSpeed
- Delay

Spectrum Attributes
- SeatPos

Image Attributes
Forwarded Attributes
Pipes
States
- ON
- OFF
- FAULT

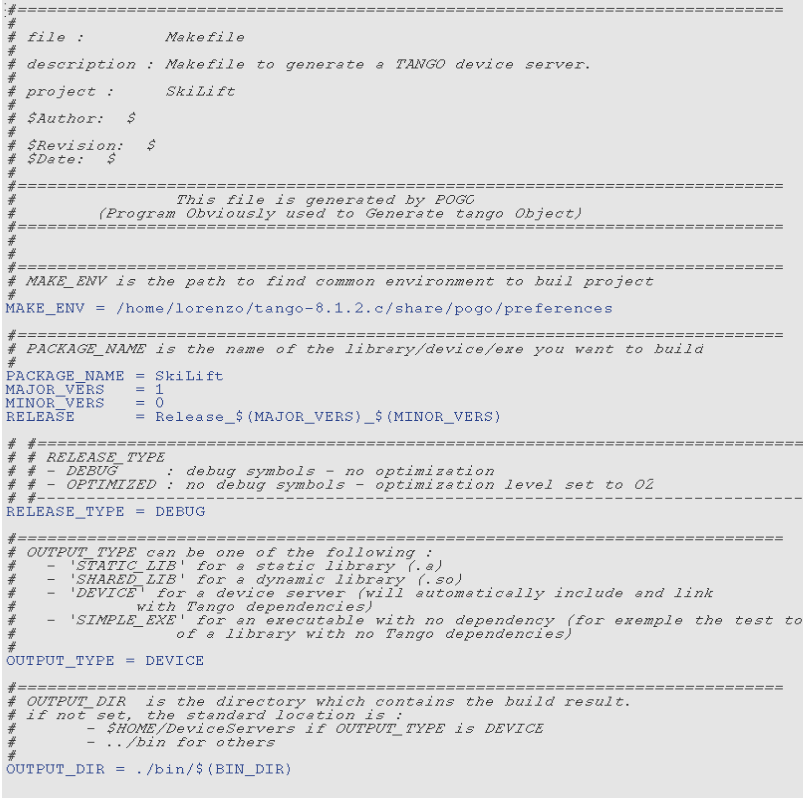
Writing a TANGO device class

POGO generates:

- C++, Python and Java device class source code
- Makefile
- TANGO device class documentation (HTML)

Compiling/Linking a TANGO device server

- two include directories
  - $(TANGO_ROOT)/include
  - $(OMNI_ROOT)/include
- two library directories
  - $(TANGO_ROOT)/lib
  - $(OMNI_ROOT)/lib
- Libraries needed (UNIX like OS)
  - 2 TANGO libs: libtango.so, liblog4tango.so
  - 4 CORBA libs: libomniORB.so, libCOS.so, libomniDynamic4.so, libomnithread.so
- OS libs
  - libpthread.so, libzmq.so



```
#==============================================================================
#
# file :        Makefile
#
# description : Makefile to generate a TANGO device server.
#
# project :     SkiLift
#
# $Author:  $
#
# $Revision:  $
# $Date:  $
#
#==============================================================================
#                   This file is generated by POGO
#           (Program Obviously used to Generate tango Object)
#==============================================================================
#
#
#==============================================================================
# MAKE_ENV is the path to find common environment to buil project
#
MAKE_ENV = /home/lorenzo/tango-8.1.2.c/share/pogo/preferences

#==============================================================================
# PACKAGE_NAME is the name of the library/device/exe you want to build
#
PACKAGE_NAME = SkiLift
MAJOR_VERS   = 1
MINOR_VERS   = 0
RELEASE      = Release_$(MAJOR_VERS)_$(MINOR_VERS)

# #===============================================================================
# # RELEASE_TYPE
# # - DEBUG     : debug symbols - no optimization
# # - OPTIMIZED : no debug symbols - optimization level set to O2
# #-------------------------------------------------------------------------------
RELEASE_TYPE = DEBUG

#==============================================================================
# OUTPUT_TYPE can be one of the following :
#   - 'STATIC_LIB' for a static library (.a)
#   - 'SHARED_LIB' for a dynamic library (.so)
#   - 'DEVICE' for a device server (will automatically include and link
#            with Tango dependencies)
#   - 'SIMPLE_EXE' for an executable with no dependency (for exemple the test to
#                of a library with no Tango dependencies)
#
OUTPUT_TYPE = DEVICE

#==============================================================================
# OUTPUT_DIR  is the directory which contains the build result.
# if not set, the standard location is :
#       - $HOME/DeviceServers if OUTPUT_TYPE is DEVICE
#       - ../bin for others
#
OUTPUT_DIR = ./bin/$(BIN_DIR)
```

Writing a TANGO device class

For the SkiLift class POGO created: 7 source code files, 1 configuration file, and the Makefile.
2 of the source code files are reserved for the device server process:

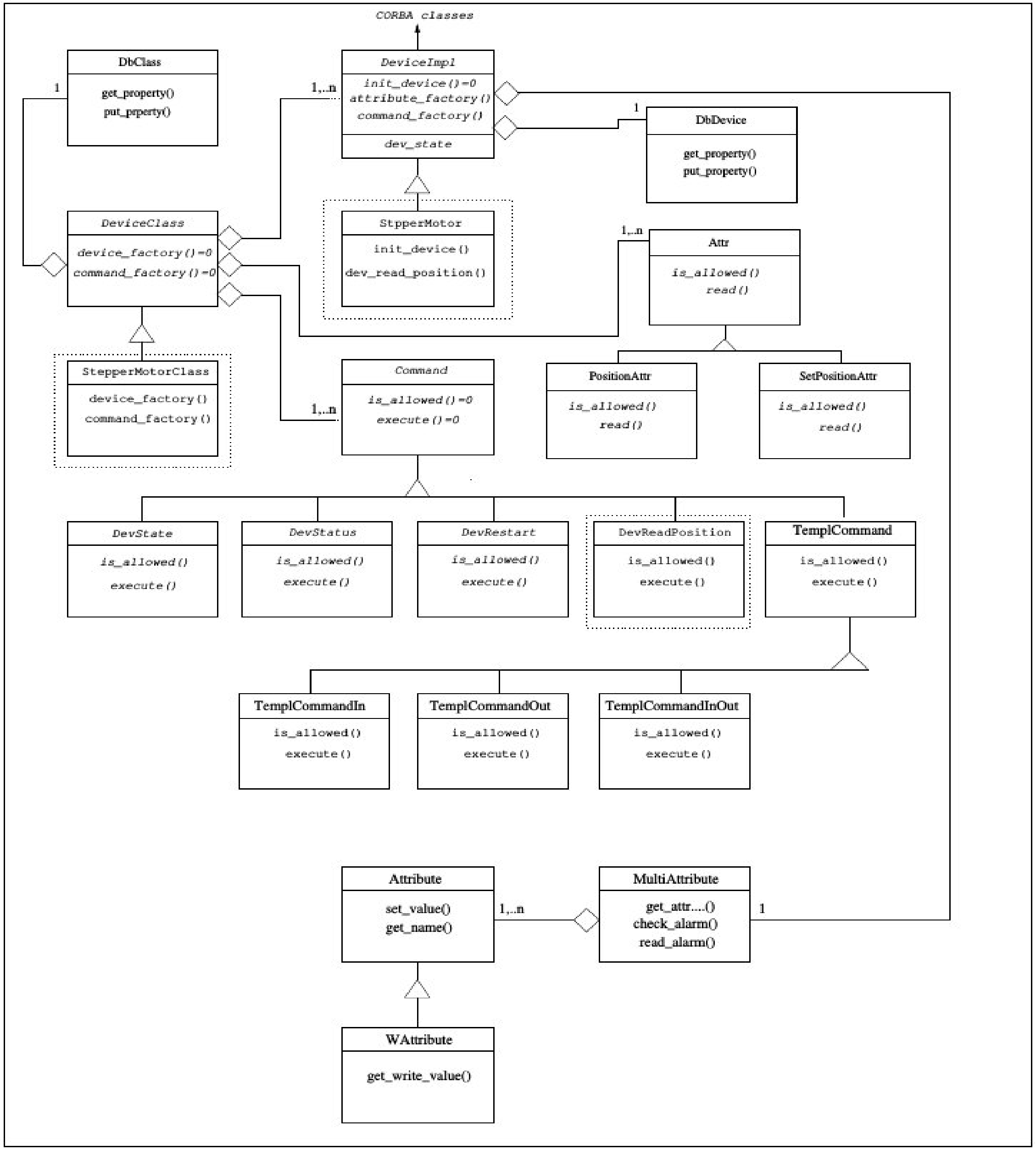
- SkiLift.h, SkiLift.cpp
- SkiLiftClass.h, SkiLiftClass.cpp
- SkiLiftStateMachine.cpp
- class_factory.cpp, main.cpp
- SkiLift.xmi
- Makefile

Most of the time only SkiLift.h and SkiLift.cpp files have to be modified

Which methods are available within a TANGO class?
- SkiLift class inherits from Device_<X>Impl class → all methods from this class
- methods that receive Attribute or Wattribute objects → all methods of these classes

See http://www.tango-controls.org "Tango Kernel" and "Tango device server classes"

Writing a TANGO device class



Device startup/shutdown

Besides class constructor and destructor methods, TANGO provides some additional methods for initialize and destroy the device

Initialization method

```
    void SkiLift::init_device()
```

Shutdown method

```
    void SkiLift::delete_device()
```

Advice: all memory allocated in init_device() **must** be deleted in delete_device()

Suppose hardware returns Speed ans WindSpeed as scalars and seat position as an array. The programmer can choose whether to let POGO allocate the memory for the required data structures or do it by herself.

In SkiLift.h the programmer has to deal with the variables/structures possibly used for hardware access

Device startup/shutdown

```
void SkiLift::init_device()
{
    DEBUG_STREAM << "SkiLift::init_device() create device " << device_name << endl;
    /*----- PROTECTED REGION ID(SkiLift::init_device_before) ENABLED START -----*/

    //  Initialization before get_device_property() call

    /*----- PROTECTED REGION END -----*/  //  SkiLift::init_device_before

    //  No device property to be read from database

    attr_Speed_read = new Tango::DevDouble[1];
    attr_WindSpeed_read = new Tango::DevDouble[1];
    attr_SeatPos_read = new Tango::DevLong[120];

    /*----- PROTECTED REGION ID(SkiLift::init_device) ENABLED START -----*/

    //  Initialize device
    *attr_Speed_read = 0.0;
    *attr_WindSpeed_read = 0.0;
    for (int i = 0; i < 120; i++)
        attr_SeatPos_read[i] = 0;

    set_state(Tango::OFF);
    set_status("SkiLift is OFF");

    /*----- PROTECTED REGION END -----*/
}
```

Device startup/shutdown

```
void SkiLift::delete_device()
{
    DEBUG_STREAM << "SkiLift::delete_device() " << device_name << endl;
    /*----- PROTECTED REGION ID(SkiLift::delete_device) ENABLED START -----*/

    //  Delete device allocated objects

    /*----- PROTECTED REGION END -----*/   //  SkiLift::delete_device
    delete[] attr_Speed_read;
    delete[] attr_WindSpeed_read;
    delete[] attr_SeatPos_read;
}
```

Command implementation

Reset command implementation

TANGO provides one *always_executed_hook()* method for all commands

```
    void SkiLift::always_executed_hook()
```

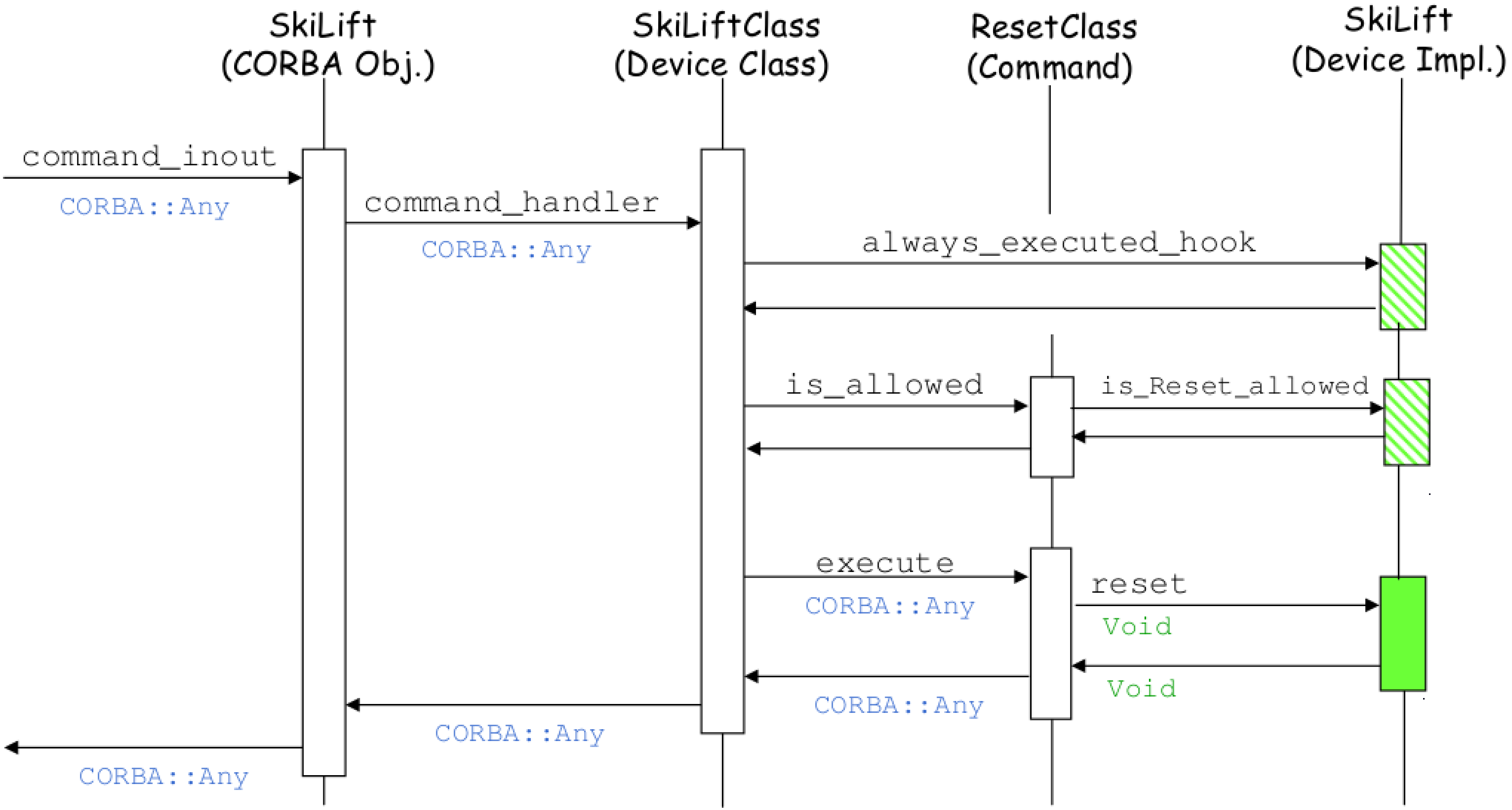
If State management is required POGO generates one `is_<xxx>_allowed()` method in SkiLiftStateMachine.cpp file

```
    bool SkiLift::is_Reset_allowed(const CORBA::Any &)
```

One method per command in SkiLift.cpp

```
    void SkiLift::Reset()
```

Command sequencing



Command implementation

`SkiLift::is_Reset_allowed()` method code, in SkiLiftStateMachine.cpp

```
bool SkiLift::is_Reset_allowed(TANGO_UNUSED(const CORBA::Any &any))
{
    //  Compare device state with not allowed states.
    if (get_state()==Tango::ON ||
        get_state()==Tango::OFF)
    {
    /*----- PROTECTED REGION ID(SkiLift::ResetStateAllowed) ENABLED START -----*/

    /*----- PROTECTED REGION END -----*/   //  SkiLift::ResetStateAllowed
        return false;
    }
    return true;
}
```

Command implementation

SkiLift::Reset() method code

```
void SkiLift::reset()
{
    DEBUG_STREAM << "SkiLift::Reset()  - " << device_name << endl;
    /*----- PROTECTED REGION ID(SkiLift::reset) ENABLED START -----*/

    //  Add your own code
    *attr_Speed_read = 0.0
    set_state(Tango::OFF);
    set_status("SkiLift is OFF");

    /*----- PROTECTED REGION END -----*/  //  SkiLift::reset
}
```

Reading Attribute(s)

TANGO provides one method for “hardware access”

```
void SkiLift::read_attr_hardware(vector<long> &)
```

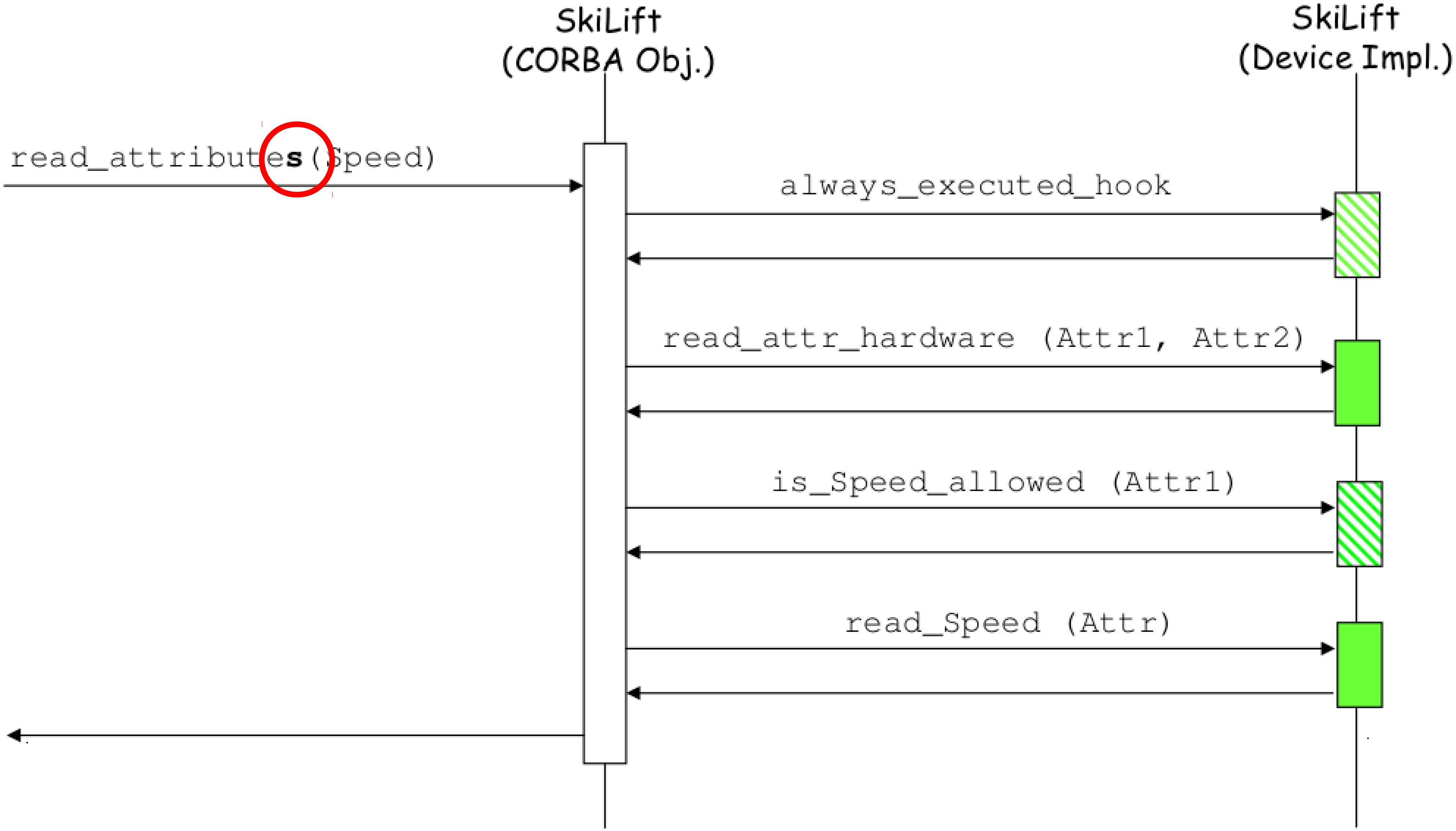
If State management is required POGO generates one `is_<xxx>_allowed()` method in SkiLiftStateMachine.cpp file

```
bool SkiLift::is_Speed_allowed(Tango::AttReqType &)
```

One method per attribute in SkiLift.cpp

```
void SkiLift::read_Speed(Tango::Attribute &)
```

Reading Attribute(s)



Reading Attribute(s)

More generally when the `read_attributes()` method is invoked the following sequencing takes place

```
/CALL/ always_executed_hook()       ← just once
/CALL/ read_attr_hardware()         ← just once
/FOR/ each attribute to be read
   /CALL/ is_<xxx>_allowed()
   /IF/ previous call returns true
      /CALL/ read_<xxx>()
   /ENDIF/
/ENDFOR/
```

This is **not** true if your client calls `read_attribute()` on several attributes; In that case no optimization takes place and the hardware will be accessed several times.

Reading Attribute(s)

read_attr_hardware() method

```
void SkiLift::read_attr_hardware(TANGO_UNUSED(vector<long> &attr_list))
{
    DEBUG_STREAM << "SkiLift::read_attr_hardware(vector<long> &attr_list) entering... "
                 << endl;
    /*----- PROTECTED REGION ID(SkiLift::read_attr_hardware) ENABLED START -----*/

    //  Add your own code
    /*
     * insert code to access you hardware
     */

    /*----- PROTECTED REGION END -----*/   //  SkiLift::read_attr_hardware
}
```

Reading Attribute(s)

`read_Speed()` method

```
void SkiLift::read_Speed(Tango::Attribute &attr)
{
    DEBUG_STREAM << "SkiLift::read_Speed(Tango::Attribute &attr) entering... " << endl;
    /*----- PROTECTED REGION ID(SkiLift::read_Speed) ENABLED START -----*/
    //   Set the attribute value
    attr.set_value(attr_Speed_read);

    /*----- PROTECTED REGION END -----*/   //   SkiLift::read_Speed
}
```

associates the method argument attr and the variable which represents it (attr_Speed_read)

Writing Attribute(s)

If State management is required, one is_<xxx>_allowed() method in SkiLiftStateMachine.cpp

```
bool SkiLift::is_Speed_allowed(Tango::AttReqType &)
```

Then, one method per write attribute

```
void SkiLift::write_Speed(Tango::Wattribute &)
```

TANGO provides one method for “hardware access”, similarly to the `read_attr_hardware()` metohod available for reading attributes

```
virtual void SkiLift::write_attr_hardware(vector<long> &)
```

The TANGO kernel provides a default implementation doing nothing

Writing Attribute(s)

More generally when the `write_attributes()` method is invoked the following sequencing takes place (Device_4Impl)

```
/CALL/ always_executed_hook()              ← just once
/FOR/ each attribute to be written
    /CALL/ is_<xxx>_allowed()
    /IF/ previous call returns true
        /CALL/ write_<xxx>()
    /ENDIF/
/ENDFOR/
/CALL/ write_attr_hardware()               ← just once
```

This is **not** true if your client calls `write_attribute()` on several attributes; In that case no optimization takes place and the hardware will be accessed several times.

Writing Attribute(s)

write_Speed() method

```
void SkiLift::write_Speed(Tango::WAttribute &attr)
{
    DEBUG_STREAM << "SkiLift::write_Speed(Tango::WAttribute &attr) entering... " << endl;
    //  Retrieve write value
    Tango::DevDouble w_val;
    attr.get_write_value(w_val);
    /*----- PROTECTED REGION ID(SkiLift::write_Speed) ENABLED START -----*/
    // insert your write Speed code here

    /*
     * trick to get some reading back
     */
    *attr_Speed_read = w_val;

    /*----- PROTECTED REGION END -----*/   //  SkiLift::write_Speed
}
```

Reporting errors

Error reporting is made using exceptions (C++ or Java)
TANGO provides the `Tango::DevFailed` class
`Tango::DevFailed` is an array of `Tango::DevError` data type
`Tango::DevError` data type has 4 elements:

- reason (string)
    the exception summary
- desc (string)
    the full error description
- origin (string)
    the method throwing the exception
- severity (enum)
    error type

TANGO provides a static method to help throwing exceptions and another method to re-trow an exception and add one element in the error stack

```
Tango::Except::throw_exception((const char *)“SkiLift::NoCable",
                               (const char *)”Cable has fall down!”,
                               (const char *)“SkiLift::init_device()");

Tango::Except::re_throw_exception(Tango::DevFailed &ex,
                                  string &reason,
                                  string &desc,
                                  string &origin);
```

Memorized Attributes

Whenever an attribute is **marked as memorized**, every change to the attribute set point is saved into the TANGO database as attribute property __value

Available only for **writable scalar attributes**

Memorized attributes initialization options (POGO)

```
Attr::set_memorized() : marks attribute as memorised

Attr::set_memorized_init(bool write_on_init)

    write_on_init = True:     calls the attribute write method during the server
                              startup
    write_on_init = False:    only initializes the attribute set point to the
                              memorized value
```

One time code

Some code to be executed only one time?

Each TANGO class has a own class (SkiLiftClass) with only one instance

Put code to be executed once in its constructor

Put data common to all devices in its data members

This class instance is constructed **before** any devices

TANGO-added cmds/attrs

TANGO automatically adds **3 commands**

State – *In = void, Out = DevState*
Check for device alarms and return the state
Status – *In = void, Out = DevString*
Return the device status
Init – *In = void, Out = void*
Reintialize the device (delete_device() + init_device())

TANGO automatically adds **2 attributes**

State and **Status**
These behave the same way as the corresponding commands

Elettra
Sincrotrone
Trieste

Remaining network calls

The TANGO core makes available some additional network calls:

- **ping** – just ping the device to see if it' s available on the network
- **command_list_query** – return the list of device supported commands with description
- **command_query** – return the command description for specific command
- **info** – return general info in the device (class, server, host...)
- **get_attribute_config** – return the attribute configuration for x (or all) attributes
- **set_attribute config** – set attribute configuration for x atributes
- **blackbox** – return n entries of the device blackbox (*)

(*) each device has a round robin buffer, with configurable depth, called blackbox
Where each network call is registered with its date and calling host

The administration device

For each device server the TANGO core provides an administration device identified by a conventional name:

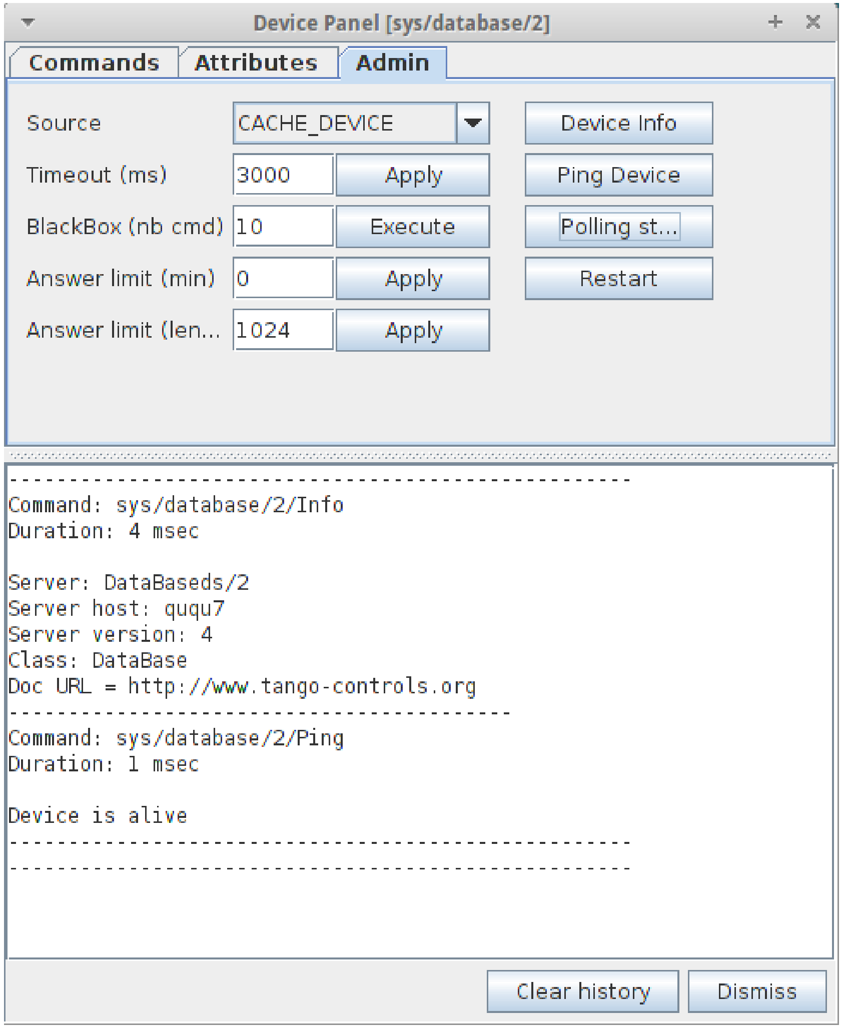
dserver/<exec-name>/<instance-name>

This device supports 20 (23) commands and 0 (2) attributes

- 8 miscellaneous commands
- 7 commands for the logging system
- 1 command for the event system
- 7 commands for the polling system

Miscellaneous commands

- DevRestart : destroy and recreate a device. Clients need to reconnect
- RestartServer : restart a complete device server instance
- QueryClass : get the list of available classes
- QueryDevice – get the list of available devices
- Kill : kill the device server process
- State, Status, Init : the ubiquitous commands



Source: CACHE_DEVICE
Timeout (ms): 3000
BlackBox (nb cmd): 10
Answer limit (min): 0
Answer limit (len...: 1024

```
----------------------------------------------------
Command: sys/database/2/Info
Duration: 4 msec

Server: DataBaseds/2
Server host: ququ7
Server version: 4
Class: DataBase
Doc URL = http://www.tango-controls.org
------------------------------------------
Command: sys/database/2/Ping
Duration: 1 msec

Device is alive
----------------------------------------------------
----------------------------------------------------
```

Logging system

The TANGO logging system allows a device server to send messages to:

- The console
- A file
- An application called LogViewer (GUI)
- A file on a remote host via specialized TANGO device server exposing the appropriate API

Six ordered logging levels: DEBUG < INFO < WARN < ERROR < FATAL < OFF

Each logging request with a level lower than the device loggin level is ignored

Device default logging level is WARN

Five macros to send logging messages

- C++ streams like: <level>_STREAM
- C printf like: LOG_<level>

Usage:

```
DEBUG_STREAM << “This is a test” << endl;
LOG_DEBUG(“Same test as before, for the %dnd time\n”, times);
```

Logging system

- Logging on the console

  send messages to the console the device server has been started

- File logging

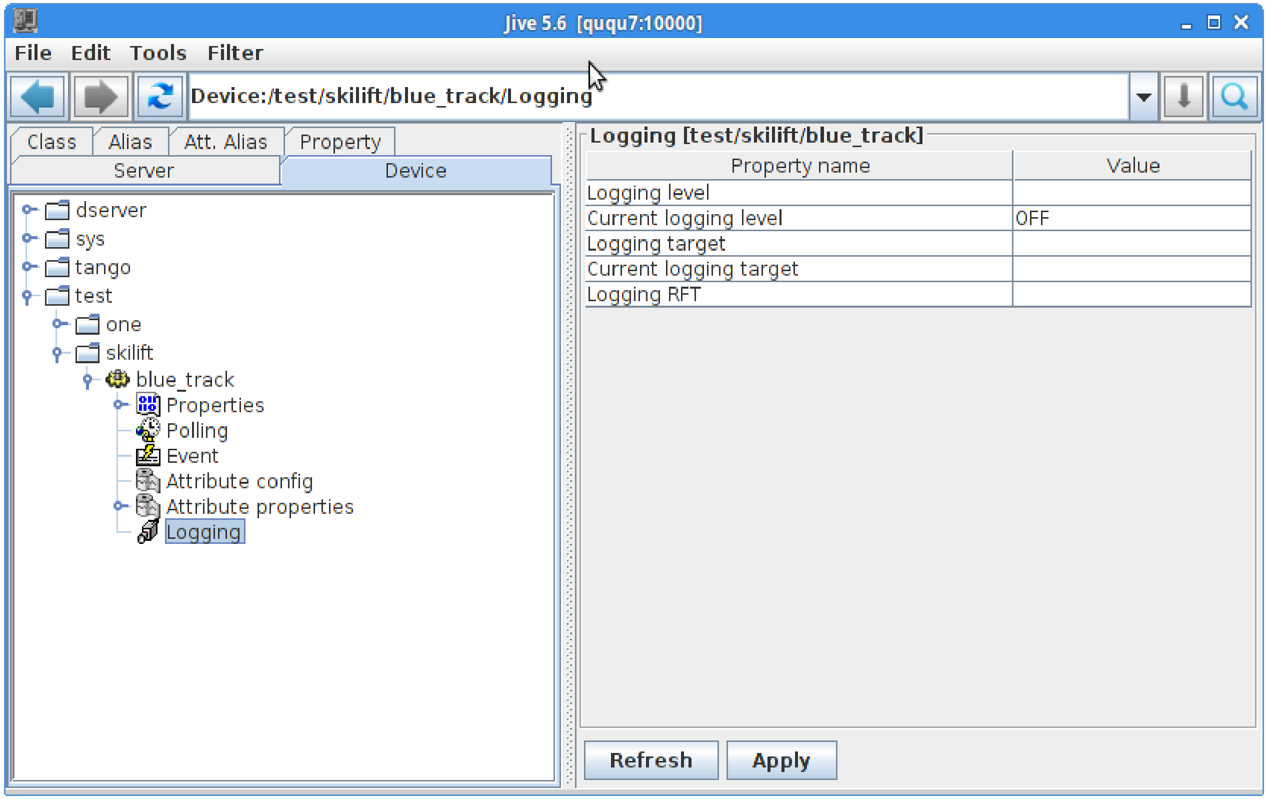
  Messages stored in a XML file
  Files rotated when size grater than predefined threshold (property, default 2MB)
  Open log files with LogViewer application

Administration device logging commands:

- AddLoggingTarget
- RemoveLoggingTarget
- GetLoggingTarget
- GetLoggingLevel
- SetLoggingLevel
- StopLogging
- StartLogging

Logging configuration with Jive

- Current logging level : not saved
- Logging level : memorized in db
- Current logging target : not saved
- Logging target : memorized in db
- Logging RFT : rolling file threshold



Device:/test/skilift/blue_track/Logging

Logging [test/skilift/blue_track]

Property name	Value
Logging level	
Current logging level	OFF
Logging target	
Current logging target	
Logging RFT	

Logging system

Device server “-v” command line option

-v1 and -v2

  Level = INFO
  Target = console::cout

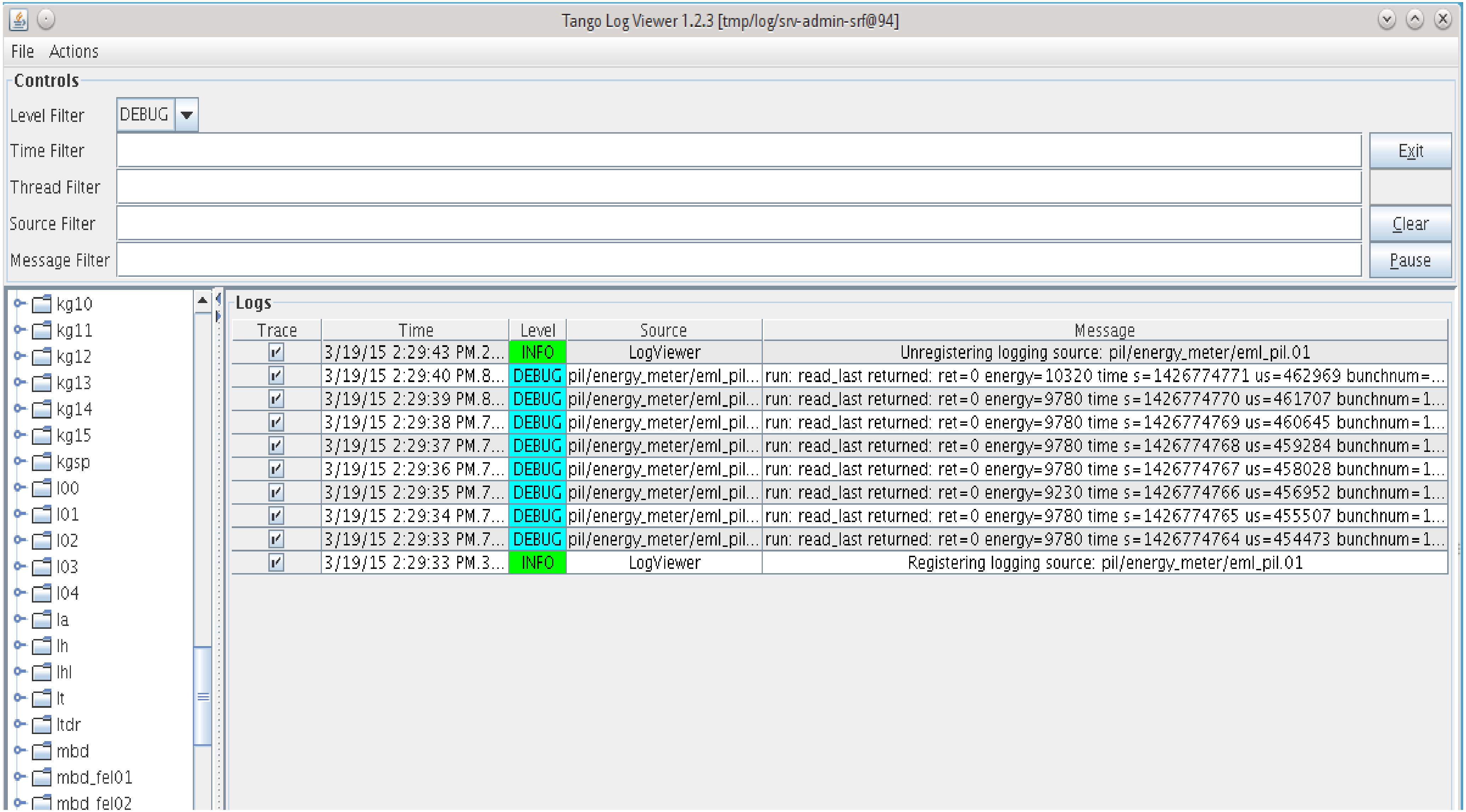
-v3 and -v4

  Level = DEBUG
  Target = console::cout

-v5

  Same as -v4 plus TANGO library messages (lots of!)
  Target = console::cout

Logging system



Tango Log Viewer 1.2.3 [tmp/log/srv-admin-srf@94]

Controls

Level Filter: DEBUG

Time Filter:

Thread Filter:

Source Filter:

Message Filter:

Logs

Trace	Time	Level	Source	Message
☑	3/19/15 2:29:43 PM.2...	INFO	LogViewer	Unregistering logging source: pil/energy_meter/eml_pil.01
☑	3/19/15 2:29:40 PM.8...	DEBUG	pil/energy_meter/eml_pil...	run: read_last returned: ret=0 energy=10320 time s=1426774771 us=462969 bunchnum=...
☑	3/19/15 2:29:39 PM.8...	DEBUG	pil/energy_meter/eml_pil...	run: read_last returned: ret=0 energy=9780 time s=1426774770 us=461707 bunchnum=1...
☑	3/19/15 2:29:38 PM.7...	DEBUG	pil/energy_meter/eml_pil...	run: read_last returned: ret=0 energy=9780 time s=1426774769 us=460645 bunchnum=1...
☑	3/19/15 2:29:37 PM.7...	DEBUG	pil/energy_meter/eml_pil...	run: read_last returned: ret=0 energy=9780 time s=1426774768 us=459284 bunchnum=1...
☑	3/19/15 2:29:36 PM.7...	DEBUG	pil/energy_meter/eml_pil...	run: read_last returned: ret=0 energy=9780 time s=1426774767 us=458028 bunchnum=1...
☑	3/19/15 2:29:35 PM.7...	DEBUG	pil/energy_meter/eml_pil...	run: read_last returned: ret=0 energy=9230 time s=1426774766 us=456952 bunchnum=1...
☑	3/19/15 2:29:34 PM.7...	DEBUG	pil/energy_meter/eml_pil...	run: read_last returned: ret=0 energy=9780 time s=1426774765 us=455507 bunchnum=1...
☑	3/19/15 2:29:33 PM.7...	DEBUG	pil/energy_meter/eml_pil...	run: read_last returned: ret=0 energy=9780 time s=1426774764 us=454473 bunchnum=1...
☑	3/19/15 2:29:33 PM.3...	INFO	LogViewer	Registering logging source: pil/energy_meter/eml_pil.01

Client side

C++, Java and Python API is provided

- easy connection between clients and devices (servers)
- manage re-connections
- hide IDL details
- hide some memory management issues

On client side the TANGO device is an instance of a **DeviceProxy** class
The instance is created from the device name

C++

```
Tango::DeviceProxy dev(“test/device/one”);
```

Python

```
dev = PyTango::DeviceProxy(“test/device/one”);
```

Client side

Command

The DeviceProxy `command_inout()` method is used to send commands to a device

```
    DeviceData DeviceProxy::command_inout(const char *, DeviceData &)
```

The DeviceData is the data type to send/receive data from the command

Read attribute

The DeviceProxy `read_attribute[s]()` method is used to read attribute from a device

```
    DeviceAttribute DeviceProxy::read_attribute[s](string &)
```

The DeviceData is the data type received from the attribute

Write attribute

The DeviceProxy `write_attribute[s]()` method is used to write attribute to a device

```
    void DeviceProxy::write_attribute[s](DeviceAttribute &)
```

The DeviceAttribute is the data type sent to the attribute

Many methods available in the DeviceProxy class

ping, info, state, status, set_timeout_millis, get_timeout_millis, attribute_query, get_attribute_config, set_attribute_config...

Use **AttributeProxy** class if you're interested only in attributes (no commands)

Communication models

Two communication models available

Client/server: the client inquires the server

- The **client** sends the request to the server; the reply can be synchronous or asynchronous

Publish/subscribe: the communication is event-driven

The device **server informs** the client that something has happened

Additionally, as a special case, **multicast** is also available through ZMQ, that uses the OpenPGM implementation of PGM protocol (RFC 3208 – reliable multicasting Protocol). Has to be configured, defining the global property CtrlSystem->MulticastEvent containing the following fields:

multicast address,	*226.20.21.22*
port number,	*2222*
[rate in Mbit/s]	*20*
[ivl in s]	*10*
event name	*device/with/multicast/state.change*

Client/Server

Synchronous call

- The **client** sends the request to the server and **blocks** waiting for the answer

Asynchronous call

- The **client** sends the request to the server and **does not block** waiting for the answer
- The device server informs the client process that the request has ended

Both mechanisms are available and do not request any change on the server side

Supported for:

- command_inout method
- read_attribute[s] method
- write_attribute[s] method

Client/Server

Asynchronous call

TANGO supports two models for clients to get the requested answer

The **polling** model
- the client decides when to check for requested answer
    - with a blocking call
    - with a non blocking call

The **callback** model
- The device server reply triggers a callback method; this can occur in one of the following sub-models:
    - when the client requested it with a synchronization method: **pull model**
    - as soon as the reply arrives in a dedicated thread: **push model**

Client/Server

Asynchronous call – polling mode

For polling mode, use

`DeviceProxy::command_inout_asynch()` method to send commands

`DeviceProxy::command_inout_reply()` method to get command replies (blocking or not blocking)

```
Tango::DeviceProxy dev(....);
long asyn_id;
asyn_id = dev.command_inout_asynch(“MyCmd”);
...
Tango::DeviceData dd;
dd = command_inout_reply(asyn_id);
```

Client/Server

Asynchronous call – callback mode

For callback mode, write a class inheriting from Tango::CallBack and write:

- `cmd_ended()` method for command execution
- `attr_read()` method for tribute reading
- `attr_written()` method for attribute writing

By default the client uses the pull model. Use `ApiUtil::set_asynch_cb_model()` to chenge

```
using namespace Tango;
class MyCb:CallBack
{
    public;
        MyCb(double d): data(d) {};
        void cmd_ended(CmdDoneEvent *);
    private:
        double data;
}

Void MyCb::cmd_ended(CmdDoneEvent *cmd)
{
    if (cmd->err == true) {
        Tango::Except::print_error_stack(cmd->errors);
    } else {
        short cmd_result;
        cmd->argout >> cmd_result;
        cout << “Cmd=“ << cmd_result << “data=“ << data << endl;
    }
}
```

```
DeviceProxy dev(...);
double my_data = 3.2;

MyCb cb(my_data);
dev.command_inout_aynch(“MyCmd”,cb);
....
dev.get_asynch_replies(150);
```

TANGO groups

TANGO groups provide the user with a **single control point for a collection of devices**. For instance, the TANGO Group API supplies a *command_inout()* method to execute the same command on all the elements of a group.
Tango Group is also a **hierarchical object**: in other words, it is possible to build a group of both groups and individual devices.

On a groups of devices you can:

Execute a command
- without arguments
- with the same input argument to all group devices
- with different input arguments for group members

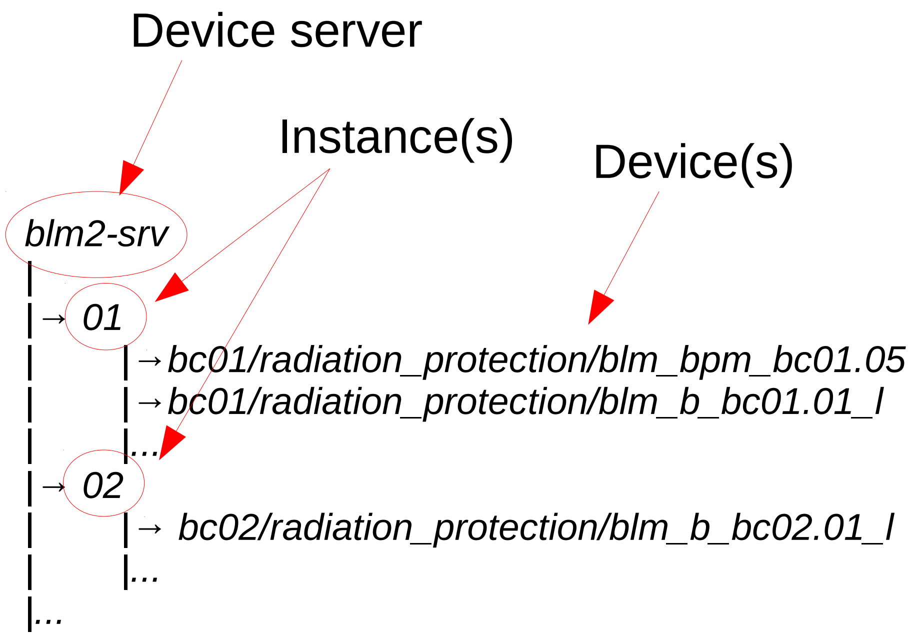
Read one attribute

Write one attribute
- with same input value for all group members
- with different inut values for group members

Simple and effective way to create logical views of the control system.

TANGO groups

Example: Beam Loss Monitors



Device server

Instance(s)

Device(s)

```
blm2-srv
|
|→ 01
|       |→bc01/radiation_protection/blm_bpm_bc01.05
|       |→bc01/radiation_protection/blm_b_bc01.01_l
|       |...
|→ 02
|       |→ bc02/radiation_protection/blm_b_bc02.01_l
|       |...
|...
```

193 total device number

```
blm = Group('radiation_protection')
blm.add('*/radiation_protection/*')
if blm->ping() == True:
        print "all devices alive"
else
        print "at least one device dead"
```

Polling

The Polling mechanism allows the Tango device to **decouple** the real device from the client(s) request(s)

Each Tango device server may have **one or more polling thread**(s) (tuning)

Polling allows to continuously monitor the “health” of the equipment

Attributes and/or Commands can be polled

The polling result is stored in a **buffer with configurable depth**, just limited by available Memory

Each device has its own polling buffer

A client is able to read data from:

- The real device (DEVICE)
- The last record in the polling buffer (CACHE)
- The polling buffer with fall-back to the real device (CACHE_DEVICE)

The complete buffer history is also available to the client → large buffers mean “automatic” shared memory mechanism available

Advice: the frequency of real hardware access has to be tuned on the equipment (e.g. accessing that old reliable 9600 baud serial line...)

Advice: the polling thread uses *read_attribute()* on each polled attribute as per TANGO 8; TANGO 9 uses *read_attributes()* if attribute have the same polling period

Elettra Sincrotrone Trieste

Polling

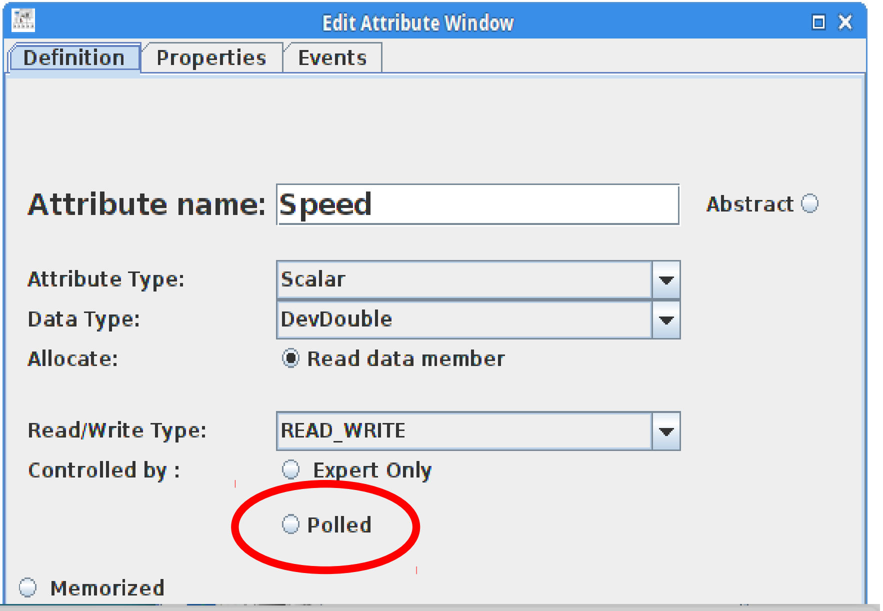
How to setup polling?

During the design phase with POGO, using the available check-buttons

At runtime, configuring the TANGO Database with Jive

Programmatically, using, for instance, Python with the client API

Programmatically in the device server itself



Attribute name: Speed

Abstract ○

Attribute Type: Scalar

Data Type: DevDouble

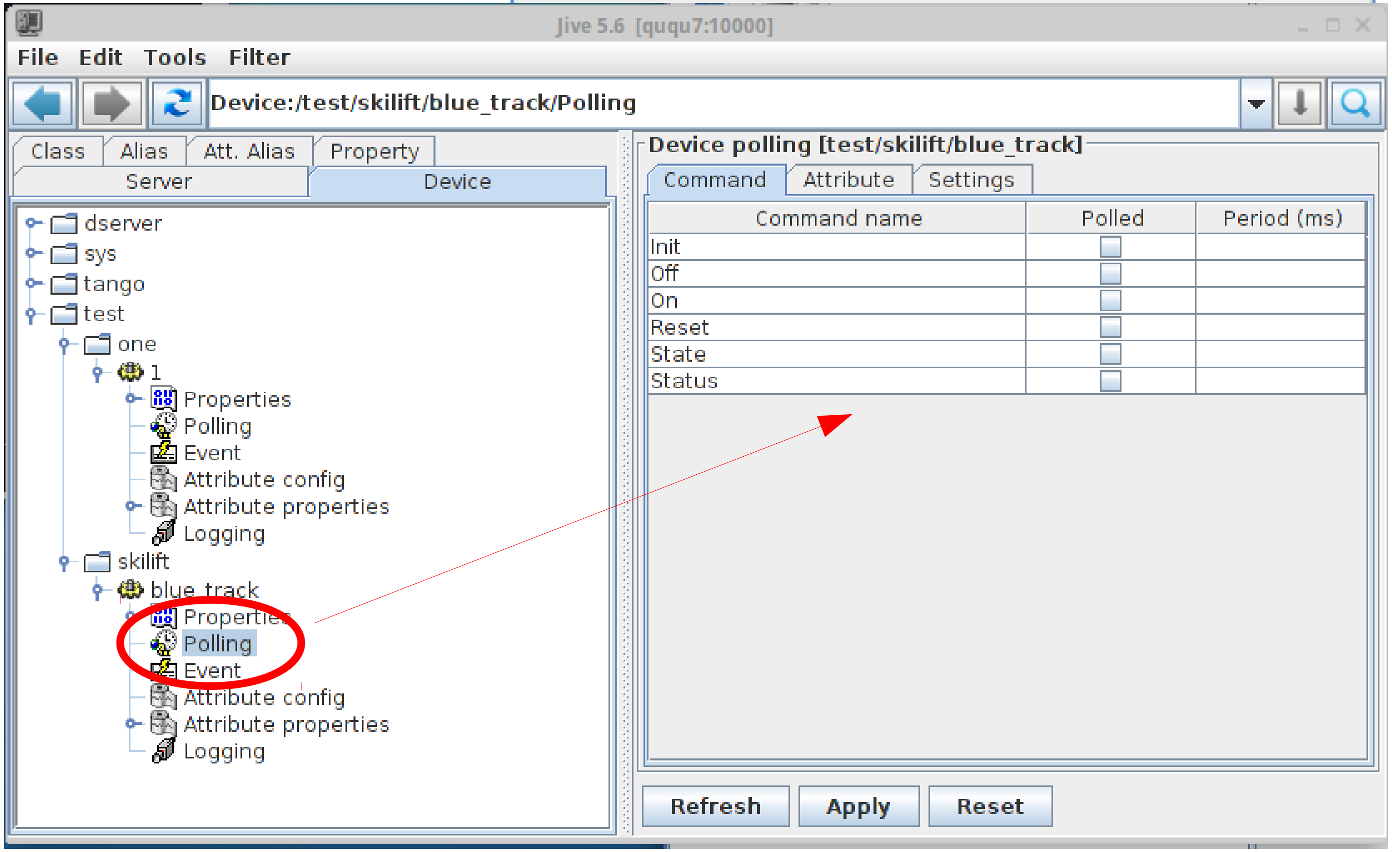
Allocate: ● Read data member

Read/Write Type: READ_WRITE

Controlled by : ○ Expert Only

○ Polled

○ Memorized



Command name	Polled	Period (ms)
Init	□	
Off	□	
On	□	
Reset	□	
State	□	
Status	□	

Test with SkiLift TANGO device

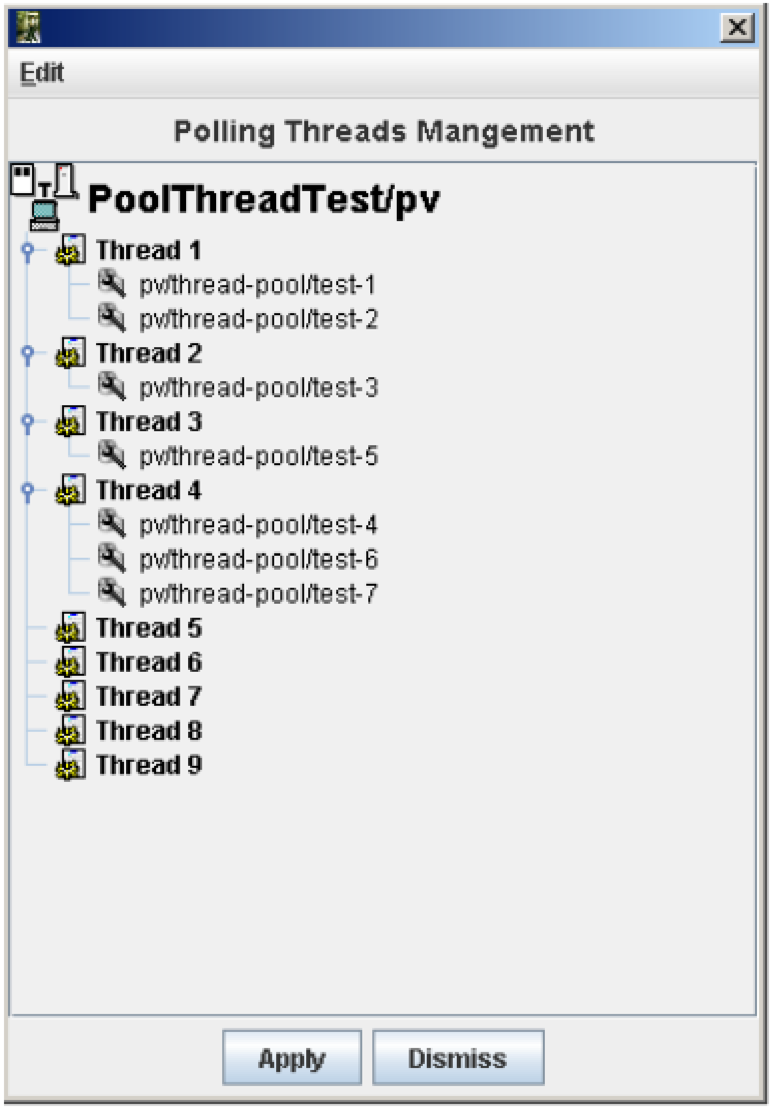
Polling

Polling thread(s) pool

Starting with Tango release 7, a Tango device server process may have several polling threads managed as a pool.

This could be useful in case of devices within the same device server process but accessing different hardware channels when one of the channel is not responding (Thus generating long timeout and de-synchronising the polling thread)

The polling thread pool can be managed
- with a GUI, available in the administration Tools
- acting on the TANGO administration device



Polling Threads Mangement

PoolThreadTest/pv
- Thread 1
  - pv/thread-pool/test-1
  - pv/thread-pool/test-2
- Thread 2
  - pv/thread-pool/test-3
- Thread 3
  - pv/thread-pool/test-5
- Thread 4
  - pv/thread-pool/test-4
  - pv/thread-pool/test-6
  - pv/thread-pool/test-7
- Thread 5
- Thread 6
- Thread 7
- Thread 8
- Thread 9

Events

Implement the publish/subscribe pattern; **based on ZeroMQ since Tango 8** (no more notification service)

Available on **attributes**

The client registers her interest **once** in an event (value)

The server informs the client every time an event has occurred

Default based on device server polling: needs configuration but does not require changes in the device server code

Additionally the event generation can be managed by the developer: **events pushed by code**

Client callback executed when an event is received

Six types of events available:

- **Change**: absolute change, relative change
- **Periodic**: period
- **Archive**: absolute change, relative change, period
- **Attribute configuration**: no parameters
- **Data ready**: managed by the developer
- **User**: managed by the developer
- **Device interface change ***: managed by the kernel
- **Pipe ***: managed by the developer

(*) Tango 9

Events

When are events pushed?

Change event

- at event subscription
- a change is detected in attribute data
- a change is detected in attribute size (spectrum/image)
- the attribute quality factor changes
- exception in the polling thread

Periodic event

- at event subscription
- on a periodic basis

Archive event

- a mix of periodic and change

Attribute configuration event

- at event subscription
- the attribute configuration is modified

User defined event

- when the user decides

Device interface change (Tango 9)

- when the device interface changes

Pipe (Tango 9)

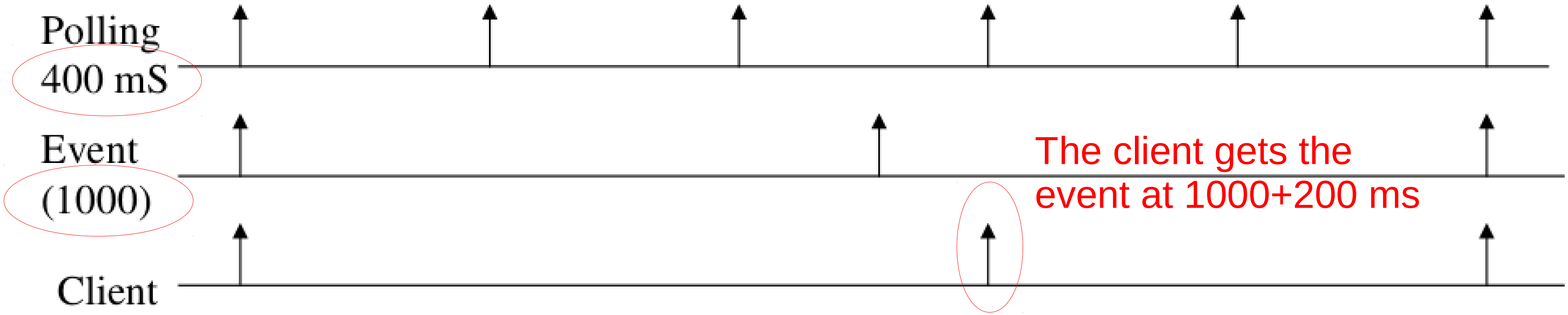
- when is executed the user code *DeviceImpl::push_pipe_event()*

Events

Periodic event configuration and behavior

event_period [ms]
- default value is 1000 ms
- cannot be faster than the polling period

Advice: whenever **event_period != polling period**
- the event system does **not** change the attribute polling period
- the event is sent when polling occurs



The client gets the event at 1000+200 ms

Push events by code to squeeze the best performance from the event system
Drawback: you need to write some code...

Events

Change event configuration

Test with SkiLift TANGO device

- Checked at the polling period

- Two thresholds: **rel_change** and **abs_change**
  Up to 2 values per threshold (positive and negative delta)
  If both set, rel_change is checked first
  If none set → **no change event**

Archive event configuration

- Checked at the polling period

- Two thresholds: **archive_rel_change**, **archive_abs_change**
  Up to 2 values per threshold (positive and negative delta)
  If both set, rel_change is checked first
  If none set → **no archive event on change**
- **archive_period [ms]**
  Default *None* → **no periodic archive event**

Events

Heartbeat

- To check that the device server is alive

  Every 10 seconds a special heartbeat event is sent to all clients on the event channel

- To inform the server that no more clients are interested in events

  A re-subscription command is sent by the client every 200 seconds.

  The device server stops sending events as soon as the last subscription command is older than 600 seconds

A dedicated client thread (keepalive thread) wakes up every 10 seconds to check the server's 10 seconds heartbeat and to send the subscription command periodically

Alarms

Device alarms

- Warning and alarm **thresholds available** as **per-attribute** configuration
- TANGO changes the State of the Device and the Quality factor of the attribute depending on attribute value and thresholds

TANGO alarms

Specialized TANGO device servers, useful to handle complex alarm rules based on multiple values/multiple logics

- C++ alarm device server: event based
- Python alarm device server: polling/event (with Taurus)

Parser for arbitrary alarm formula support

*kg01/mod/linkstabilizer_kg01.01/State == ON && kg01/mod/linkstabilizer_kg01.01/Drift1_Threshold && *
abs(kg01/mod/linkstabilizer_kg01.01/Drift1_rate) > kg01/mod/linkstabilizer_kg01.01/Drift1_Threshold

Support for alarm groups and alarm levels (LOG, WARNING, FAULT)
Support for external command execution on TANGO device server

Scalability: any number of TANGO alarm servers can be deployed, based on requirements, architectural constraints, performance required...

Device Alarms

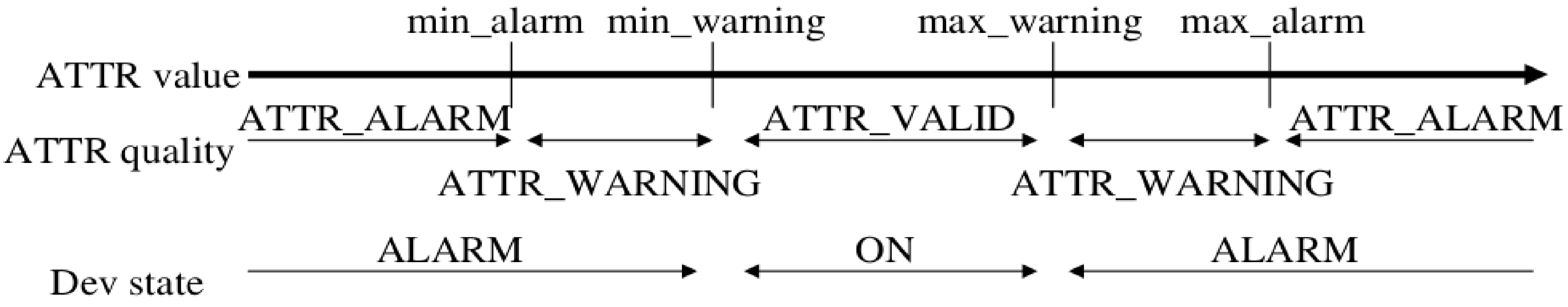
Device alarms

Test with SkiLift TANGO device

Two types of alarms can be configured on **Attributes:**

- **on value**
  - two thresholds: **WARNING** and **ALARM** with min and max parameters
- **on read different than set** (for read-write Attributes)
  - two parameters
    - the authorized **delta value**
    - the **delta time** between last attribute setting (write)and the attribute value check

TANGO manages automatically the quality factor associated to the attribute and the device State



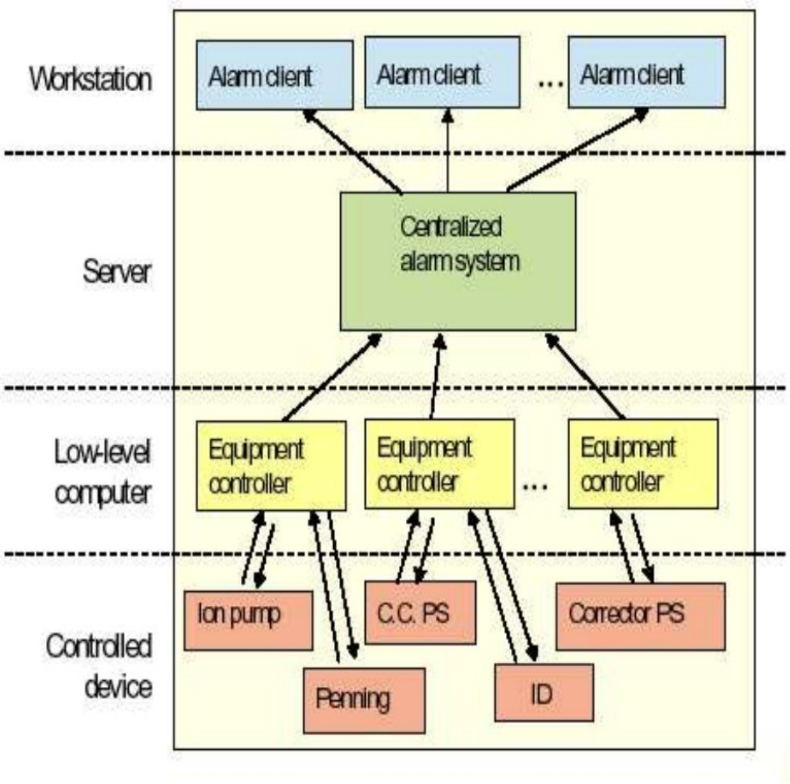
Alarm device server

Is a Tango device server based on a double client/server architecture:

as a client gathers input values from Tango devices

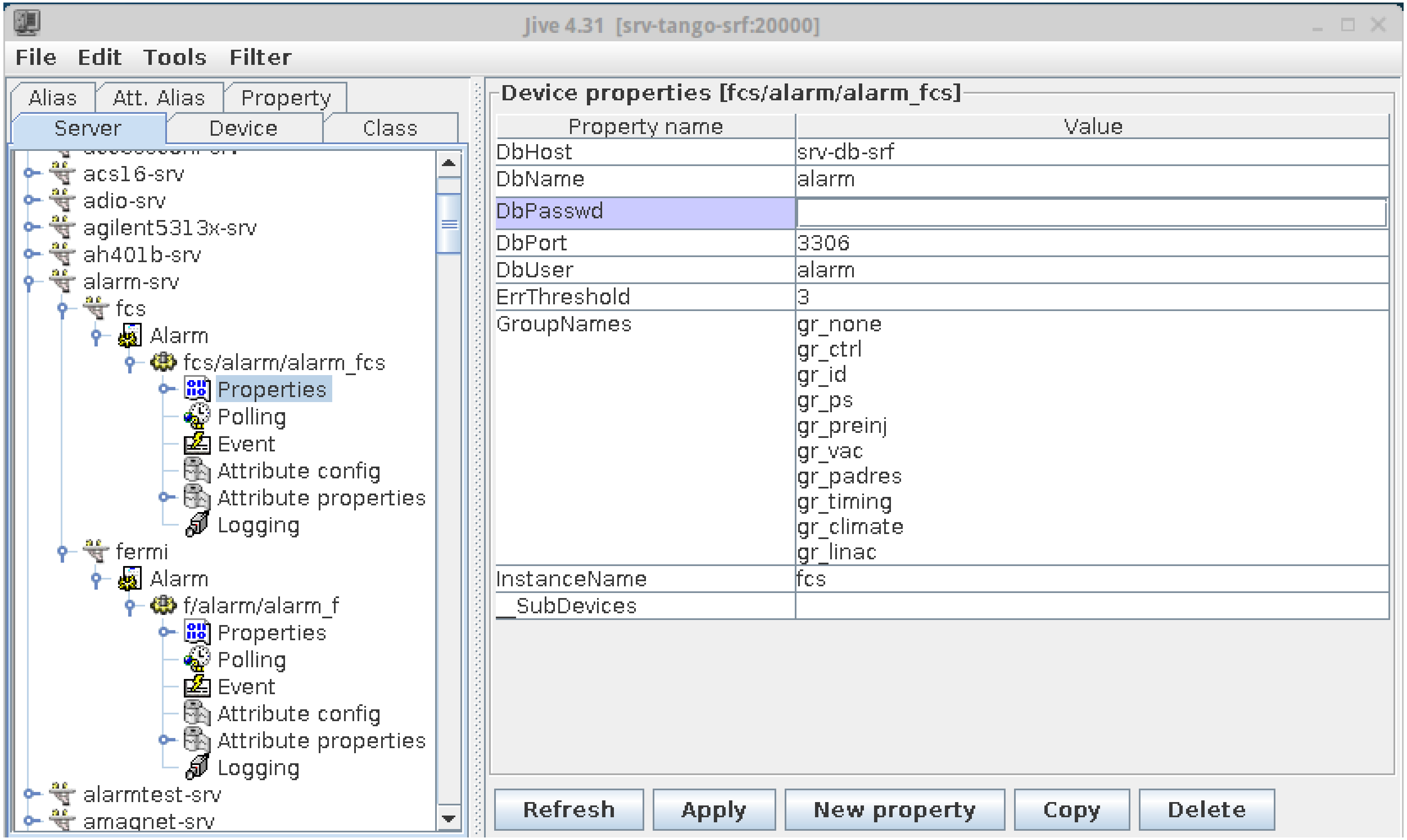
as a server provide alarm notifications

Relies on the Tango event system to collect input values as well as to provide alarm notifications



Alarm device server

Based on the BOOST library to parse and evaluate the alarm rules
Dedicated MySQL database schema to store the alarms and alarm history
Dedicated database user



Device properties [fcs/alarm/alarm_fcs]

Property name	Value
DbHost	srv-db-srf
DbName	alarm
DbPasswd	
DbPort	3306
DbUser	alarm
ErrThreshold	3
GroupNames	gr_none gr_ctrl gr_id gr_ps gr_preinj gr_vac gr_padres gr_timing gr_climate gr_linac
InstanceName	fcs
__SubDevices	

Alarm device server

Alarm GUI screenshot

Active alarms tab

Alarm name

Alarm status
ALARM, NORMAL

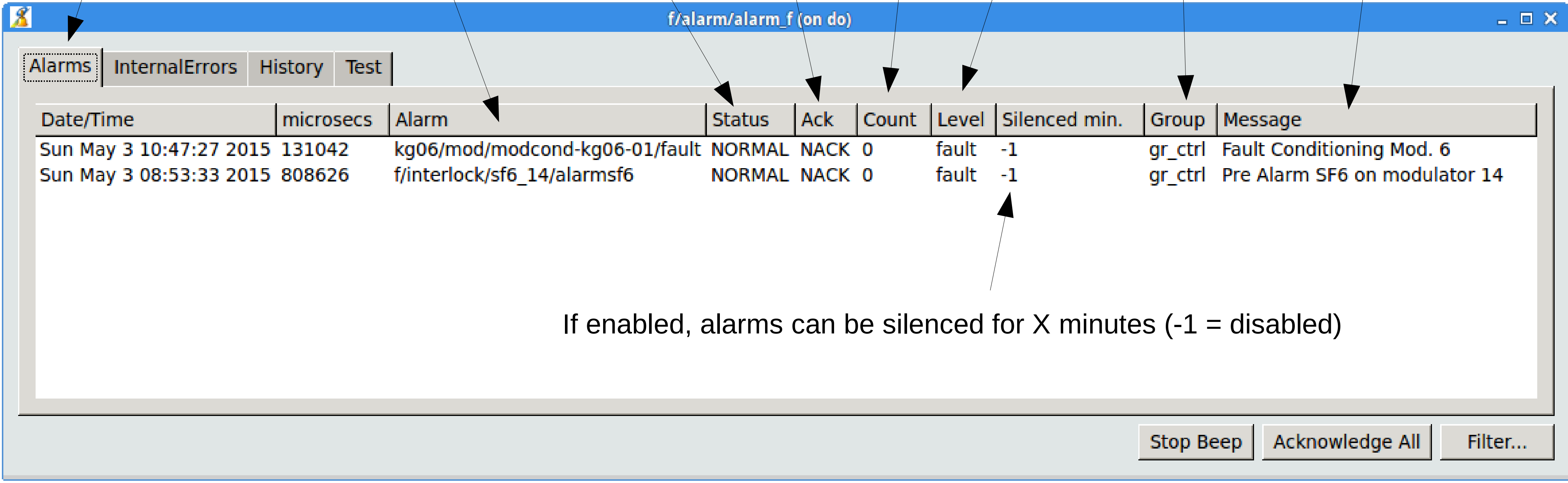
Acknowledge status
NACK, ACK

Alarm count

Alarm level
LOG, WARNING, FAULT

Alarm group

Alarm message



Date/Time	microsecs	Alarm	Status	Ack	Count	Level	Silenced min.	Group	Message
Sun May 3 10:47:27 2015	131042	kg06/mod/modcond-kg06-01/fault	NORMAL	NACK	0	fault	-1	gr_ctrl	Fault Conditioning Mod. 6
Sun May 3 08:53:33 2015	808626	f/interlock/sf6_14/alarmsf6	NORMAL	NACK	0	fault	-1	gr_ctrl	Pre Alarm SF6 on modulator 14

If enabled, alarms can be silenced for X minutes (-1 = disabled)

All the configuration is kept in the Alarm device server Properties or in the alarm database
All the logic is maintained by the alarm device server, no logic in the GUI

Historical Database

HDB (Java) - Set of three databases

- HDB: permanent, up to 0.1 Hz (1 Hz) archiving rate
- TDB: temporary, up to 1 Hz (10 Hz) archiving rate
- Snap: context save/restore
- Support for Oracle and MySQL RDBMS
- 4(+3)+3 Device servers
- **Polling** based
- GUI: Mambo, Bensikin

HDB++ (C++)

- One database for slow and fast archiving (up to 1 Khz)
- Support for existing HDB schema on MySQL
- Support for **hdb++ new schema** with improved features (µs timestamp)
- Support for **noSQL** backend (Apache Cassandra)
- 2 Device servers (EventSubscriber, ConfigurationManager)
- **Event** based
- Fast data extraction library
- GUI: HdbConfigurator, qhdbextractor (plotting)
- **Scalability**: same as TANGO, deploy as many DS as you need

TimeMachine

- System restoring tool based on context, HDB++ archived data and extraction library

HDB++ archiving system

HDB++ Archiver TANGO device server (HdbEventSubscriber)

- event based (receive archive events, generate archive and change events)
- all the **configuration** stored in the TANGO device
- storing through an external library
- defined interface for the external library
- implementations for the external library for different backends, schema

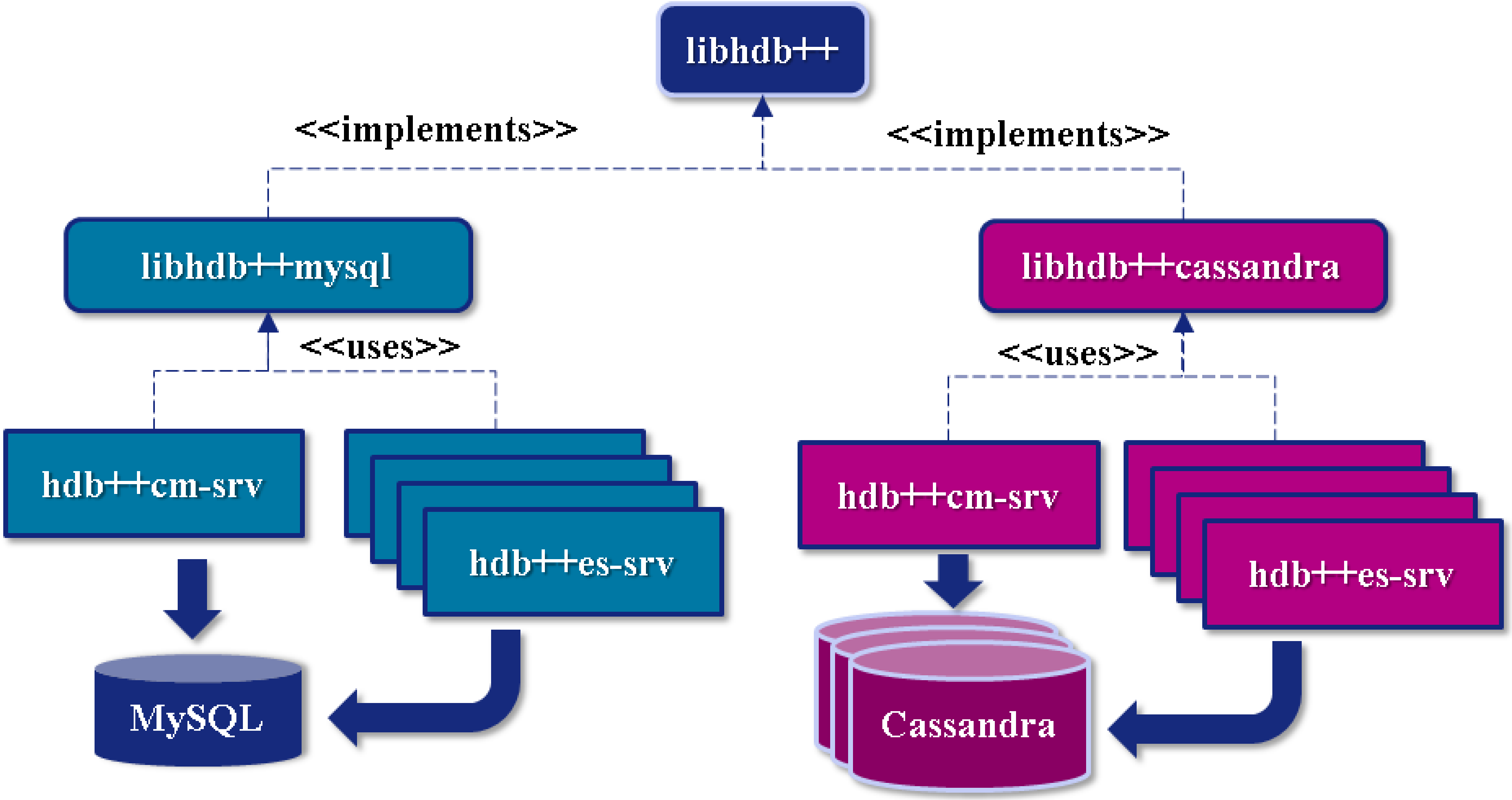
HDB++ Configurator TANGO device server (HdbConfigurationManager)

- collect information on status, performances from many archivers
- send configuration to many archivers

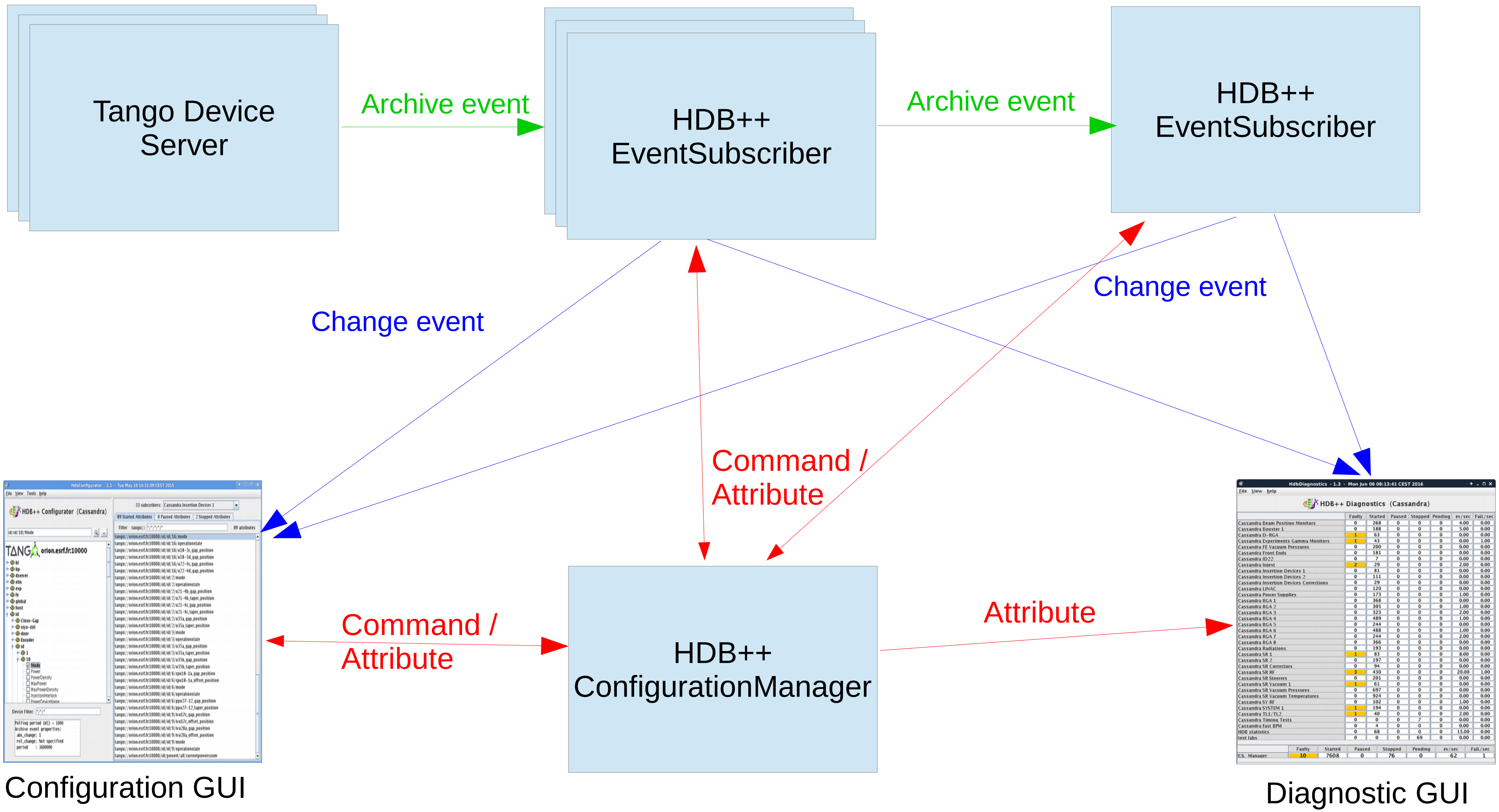
HDB++ Extraction Tools

- defined interface for an extraction library
- implementation of the extraction library for different backends, schema
- implementation of the extraction library with different languages (C++, Java)
- GUIs implemented in different languages (C++, Java, Python)

HDB++ archiving system



HDB++ archiving system



```
tango://orion.esrf.fr:10000/id/id/16/mode
tango://orion.esrf.fr:10000/id/id/16/operationstate
tango://orion.esrf.fr:10000/id/id/16/u18-3c_gap_position
tango://orion.esrf.fr:10000/id/id/16/u18-3d_gap_position
tango://orion.esrf.fr:10000/id/id/16/u22-4c_gap_position
tango://orion.esrf.fr:10000/id/id/16/u22-4d_gap_position
tango://orion.esrf.fr:10000/id/id/2/mode
tango://orion.esrf.fr:10000/id/id/2/operationstate
tango://orion.esrf.fr:10000/id/id/2/u21-4b_gap_position
tango://orion.esrf.fr:10000/id/id/2/u21-4b_taper_position
tango://orion.esrf.fr:10000/id/id/2/u21-4c_gap_position
tango://orion.esrf.fr:10000/id/id/2/u21-4c_taper_position
tango://orion.esrf.fr:10000/id/id/2/u35a_gap_position
tango://orion.esrf.fr:10000/id/id/2/u35a_taper_position
tango://orion.esrf.fr:10000/id/id/3/mode
tango://orion.esrf.fr:10000/id/id/3/operationstate
tango://orion.esrf.fr:10000/id/id/3/u35a_gap_position
tango://orion.esrf.fr:10000/id/id/3/u35a_taper_position
tango://orion.esrf.fr:10000/id/id/3/u35b_gap_position
tango://orion.esrf.fr:10000/id/id/3/u35b_taper_position
tango://orion.esrf.fr:10000/id/id/6/cpu18-1a_gap_position
tango://orion.esrf.fr:10000/id/id/6/cpu18-1a_offset_position
tango://orion.esrf.fr:10000/id/id/6/mode
tango://orion.esrf.fr:10000/id/id/6/operationstate
tango://orion.esrf.fr:10000/id/id/6/ppu27-12_gap_position
tango://orion.esrf.fr:10000/id/id/6/ppu27-12_taper_position
tango://orion.esrf.fr:10000/id/id/9/ivu17c_gap_position
tango://orion.esrf.fr:10000/id/id/9/ivu17c_offset_position
tango://orion.esrf.fr:10000/id/id/9/ivu20a_gap_position
tango://orion.esrf.fr:10000/id/id/9/ivu20a_offset_position
tango://orion.esrf.fr:10000/id/id/9/mode
tango://orion.esrf.fr:10000/id/id/9/operationstate
tango://orion.esrf.fr:10000/id/powert/all/currentpowerssum
```

```
Polling period (mS) = 1000
Archive event properties:
 abs_change: 1
 rel_change: Not specified
 period    : 3600000
```

	Faulty	Started	Paused	Stopped	Pending	ev/sec	Fail./sec
Cassandra Beam Position Monitors	0	268	0	0	0	4.00	0.00
Cassandra Booster 1	0	188	0	0	0	5.00	0.00
Cassandra D-RGA	1	63	0	0	0	0.00	0.00
Cassandra Experiments Gamma Monitors	1	43	0	0	0	0.00	1.00
Cassandra FE Vacuum Pressures	0	200	0	0	0	0.00	0.00
Cassandra Front Ends	0	181	0	0	0	0.00	0.00
Cassandra ID22	0	7	0	0	0	0.00	0.00
Cassandra Injext	2	29	0	0	0	2.00	0.00
Cassandra Insertion Devices 1	0	81	0	0	0	0.00	0.00
Cassandra Insertion Devices 2	0	111	0	0	0	0.00	0.00
Cassandra Insertion Devices Corrections	0	29	0	0	0	0.00	0.00
Cassandra LINAC	0	120	0	0	0	0.00	0.00
Cassandra Power Supplies	0	173	0	0	0	1.00	0.00
Cassandra RGA 1	0	368	0	0	0	0.00	0.00
Cassandra RGA 2	0	305	0	0	0	1.00	0.00
Cassandra RGA 3	0	323	0	0	0	2.00	0.00
Cassandra RGA 4	0	489	0	0	0	1.00	0.00
Cassandra RGA 5	0	244	0	0	0	0.00	0.00
Cassandra RGA 6	0	488	0	0	0	1.00	0.00
Cassandra RGA 7	0	244	0	0	0	2.00	0.00
Cassandra RGA 8	0	366	0	0	0	0.00	0.00
Cassandra Radiations	0	193	0	0	0	0.00	0.00
Cassandra SR 1	1	83	0	0	0	8.00	0.00
Cassandra SR 2	0	197	0	0	0	0.00	0.00
Cassandra SR Correctors	0	94	0	0	0	0.00	0.00
Cassandra SR RF	3	430	0	0	0	20.00	1.00
Cassandra SR Steerers	0	201	0	0	0	0.00	0.00
Cassandra SR Vacuum 1	1	61	0	0	0	0.00	0.00
Cassandra SR Vacuum Pressures	0	697	0	0	0	0.00	0.00
Cassandra SR Vacuum Temperatures	0	924	0	0	0	0.00	0.00
Cassandra SY RF	0	102	0	0	0	1.00	0.00
Cassandra SYSTEM 1	1	194	0	0	0	0.00	0.00
Cassandra TL1/TL2	1	40	0	0	0	2.00	0.00
Cassandra Timing Tests	0	0	0	7	0	0.00	0.00
Cassandra fast BPM	0	4	0	0	0	0.00	0.00
HDB statistics	0	68	0	0	0	15.00	0.00
test labs	0	0	0	69	0	0.00	0.00

	Faulty	Started	Paused	Stopped	Pending	ev/sec	Fail./sec
E.S. Manager	10	7608	0	76	0	62	1

Configuration GUI

Diagnostic GUI

HDB++ design guidelines

Data extraction

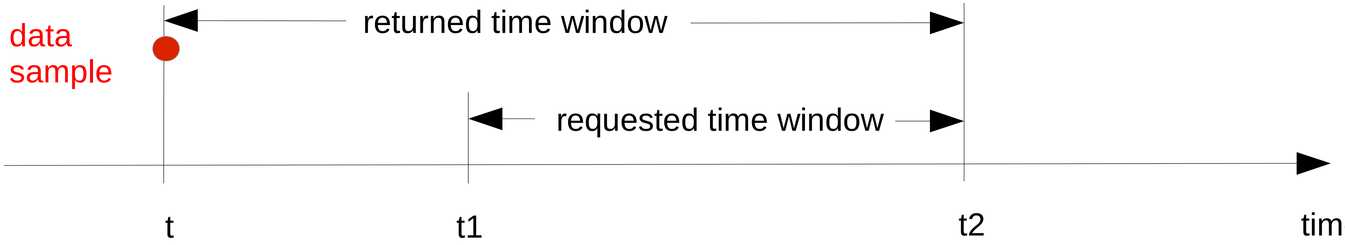
Work in progress
Not exhaustive

- C++ and Java native libraries

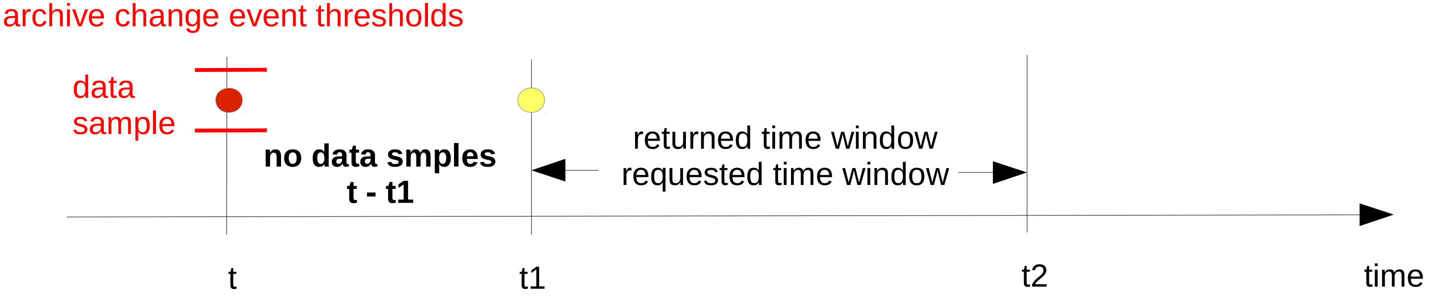
- The data extraction library shall be able to **deal with event based archiving;** the possible lack of data in the requested time window shall be properly managed:

  - returning some no-data-available error: in this case the reply contains no data

  - enlarging the time window to include some archived data; no fake samples have to be introduced

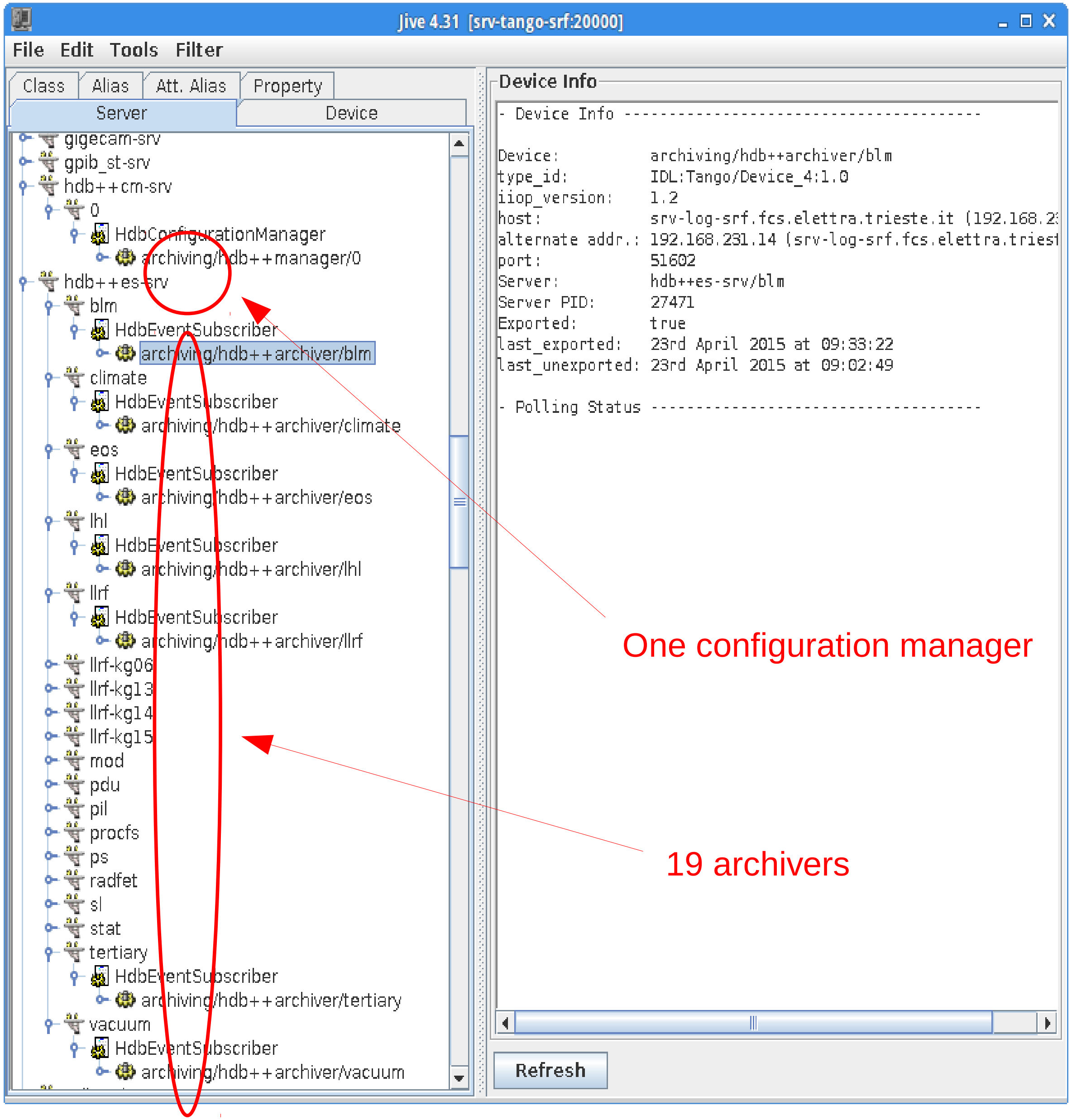


  - returning the value of the last archived data anyhow; the requested time interval is kept and the last available data sample returned; the validity of the data is guaranteed when **archive change event** is used, care must be taken in case of **archive periodic event**



Historical Database

Example: FERMI setup
- 1 host
- 1 configuration manager
- 19 archivers
- functional partitioning: one archiver per subsystem
- 5356 attributes total
- from 1 to 1467 attributes per archiver



```
gigecam-srv
gpib_st-srv
hdb++cm-srv
  0
    HdbConfigurationManager
      archiving/hdb++manager/0
hdb++es-srv
  blm
    HdbEventSubscriber
      archiving/hdb++archiver/blm
  climate
    HdbEventSubscriber
      archiving/hdb++archiver/climate
  eos
    HdbEventSubscriber
      archiving/hdb++archiver/eos
  lhl
    HdbEventSubscriber
      archiving/hdb++archiver/lhl
  llrf
    HdbEventSubscriber
      archiving/hdb++archiver/llrf
  llrf-kg06
  llrf-kg13
  llrf-kg14
  llrf-kg15
  mod
  pdu
  pil
  procfs
  ps
  radfet
  sl
  stat
  tertiary
    HdbEventSubscriber
      archiving/hdb++archiver/tertiary
  vacuum
    HdbEventSubscriber
      archiving/hdb++archiver/vacuum
```

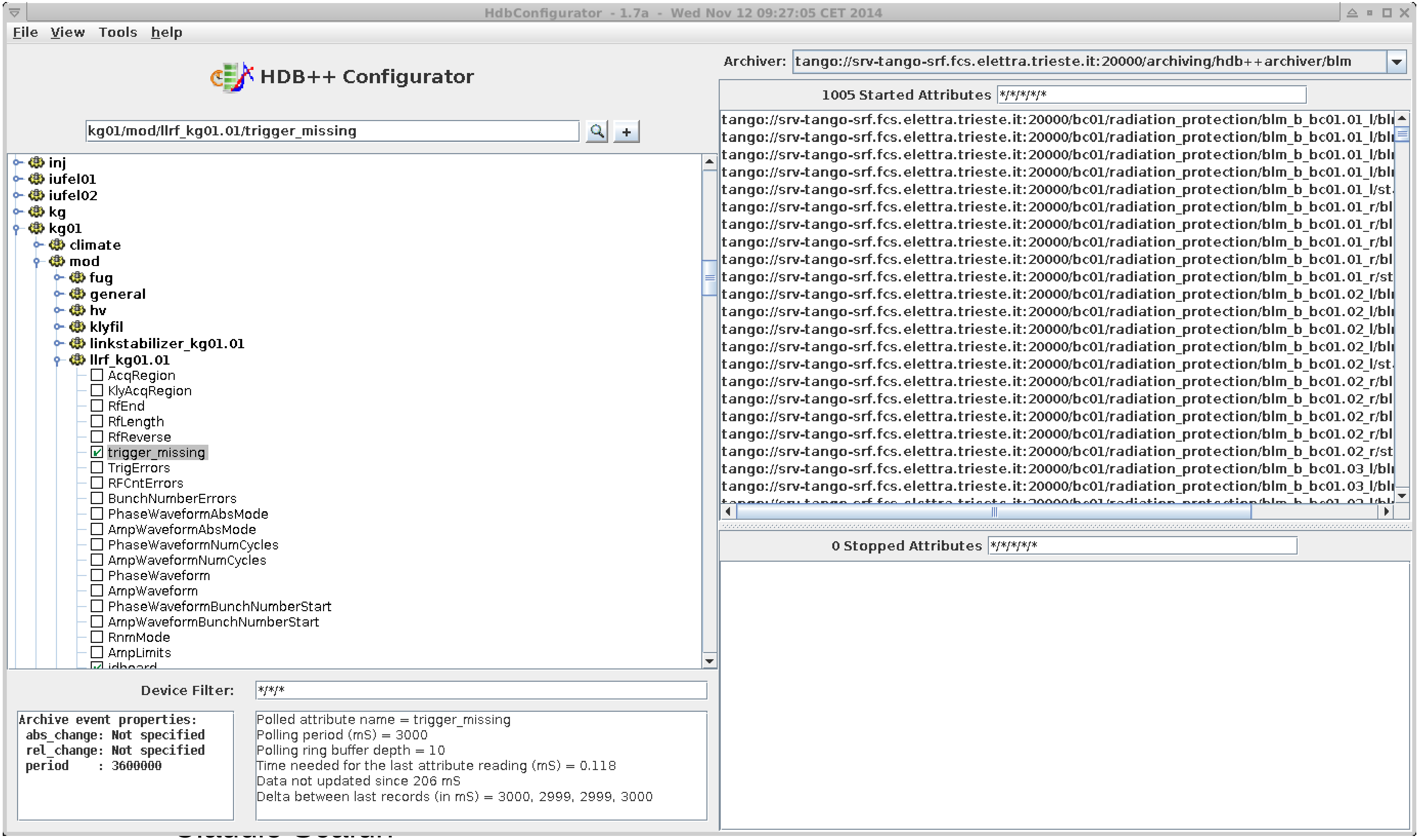
```
- Device Info ------------------------------------

Device:           archiving/hdb++archiver/blm
type_id:          IDL:Tango/Device_4:1.0
iiop_version:     1.2
host:             srv-log-srf.fcs.elettra.trieste.it (192.168.2
alternate addr.:  192.168.231.14 (srv-log-srf.fcs.elettra.triest
port:             51602
Server:           hdb++es-srv/blm
Server PID:       27471
Exported:         true
last_exported:    23rd April 2015 at 09:33:22
last_unexported:  23rd April 2015 at 09:02:49

- Polling Status ----------------------------------
```

One configuration manager

19 archivers

Elettra
Sincrotrone
Trieste

Historical Database



HdbConfigurator - 1.7a - Wed Nov 12 09:27:05 CET 2014

File View Tools help

HDB++ Configurator

kg01/mod/llrf_kg01.01/trigger_missing

```
inj
iufel01
iufel02
kg
kg01
  climate
  mod
    fug
    general
    hv
    klyfil
    linkstabilizer_kg01.01
    llrf_kg01.01
      ☐ AcqRegion
      ☐ KlyAcqRegion
      ☐ RfEnd
      ☐ RfLength
      ☐ RfReverse
      ☑ trigger_missing
      ☐ TrigErrors
      ☐ RFCntErrors
      ☐ BunchNumberErrors
      ☐ PhaseWaveformAbsMode
      ☐ AmpWaveformAbsMode
      ☐ PhaseWaveformNumCycles
      ☐ AmpWaveformNumCycles
      ☐ PhaseWaveform
      ☐ AmpWaveform
      ☐ PhaseWaveformBunchNumberStart
      ☐ AmpWaveformBunchNumberStart
      ☐ RnmMode
      ☐ AmpLimits
      ☑ idboard
```

Device Filter: */*/*

```
Archive event properties:
 abs_change: Not specified
 rel_change: Not specified
 period    : 3600000
```

```
Polled attribute name = trigger_missing
Polling period (mS) = 3000
Polling ring buffer depth = 10
Time needed for the last attribute reading (mS) = 0.118
Data not updated since 206 mS
Delta between last records (in mS) = 3000, 2999, 2999, 3000
```

Archiver: tango://srv-tango-srf.fcs.elettra.trieste.it:20000/archiving/hdb++archiver/blm

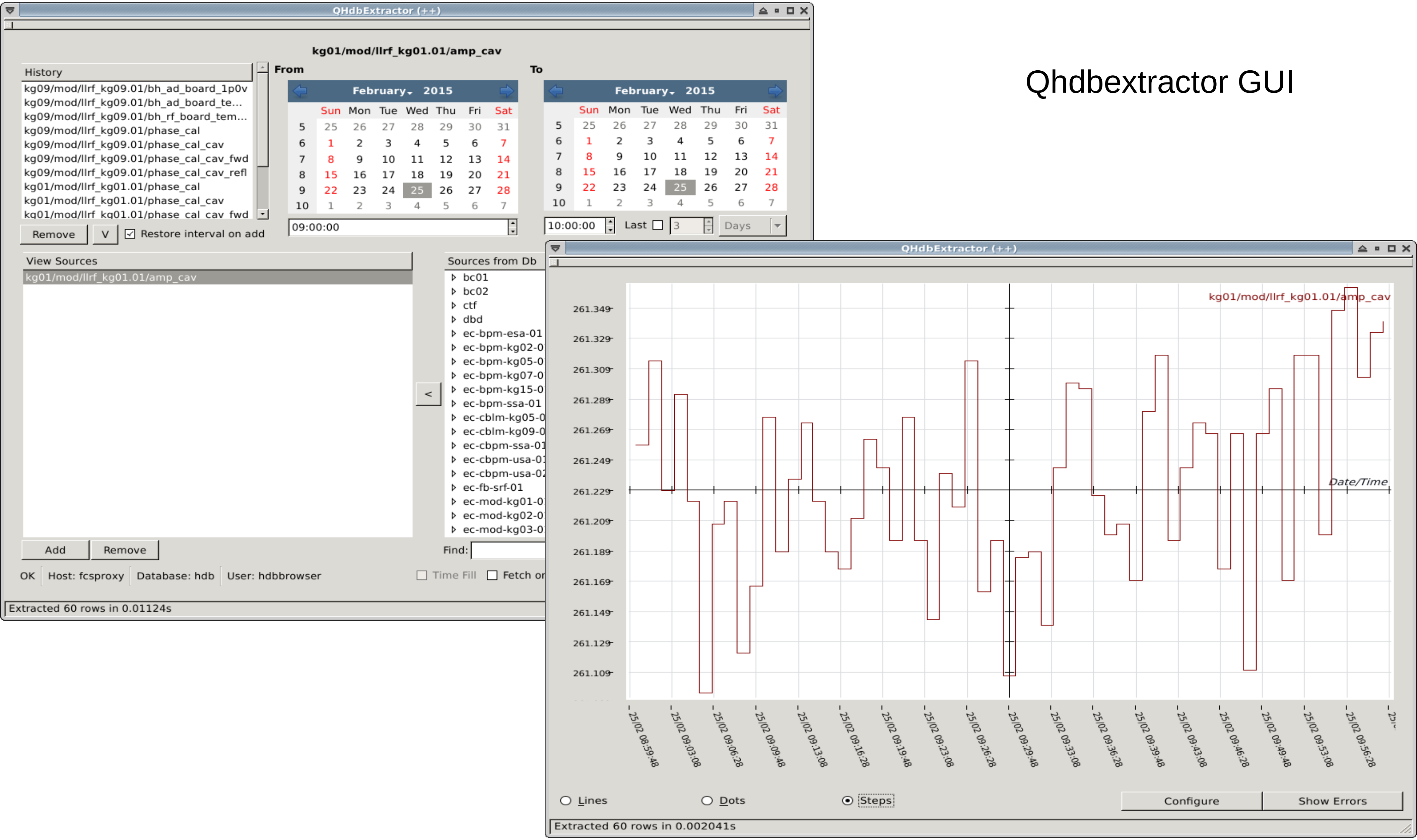
1005 Started Attributes */*/*/*/*

```
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.01_l/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.01_l/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.01_l/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.01_l/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.01_l/st
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.01_r/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.01_r/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.01_r/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.01_r/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.01_r/st
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.02_l/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.02_l/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.02_l/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.02_l/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.02_l/st
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.02_r/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.02_r/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.02_r/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.02_r/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.02_r/st
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.03_l/bl
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/bc01/radiation_protection/blm_b_bc01.03_l/bl
```

0 Stopped Attributes */*/*/*/*

Historical Database

Qhdbextractor GUI



QHdbExtractor (++)

kg01/mod/llrf_kg01.01/amp_cav

History
kg09/mod/llrf_kg09.01/bh_ad_board_1p0v
kg09/mod/llrf_kg09.01/bh_ad_board_te...
kg09/mod/llrf_kg09.01/bh_rf_board_tem...
kg09/mod/llrf_kg09.01/phase_cal
kg09/mod/llrf_kg09.01/phase_cal_cav
kg09/mod/llrf_kg09.01/phase_cal_cav_fwd
kg09/mod/llrf_kg09.01/phase_cal_cav_refl
kg01/mod/llrf_kg01.01/phase_cal
kg01/mod/llrf_kg01.01/phase_cal_cav
kg01/mod/llrf_kg01.01/phase_cal_cav_fwd

From: February 2015, 25 — 09:00:00

To: February 2015, 25 — 10:00:00 — Last □ 3 Days

☑ Restore interval on add

View Sources
kg01/mod/llrf_kg01.01/amp_cav

Sources from Db
bc01
bc02
ctf
dbd
ec-bpm-esa-01
ec-bpm-kg02-0
ec-bpm-kg05-0
ec-bpm-kg07-0
ec-bpm-kg15-0
ec-bpm-ssa-01
ec-cblm-kg05-0
ec-cblm-kg09-0
ec-cbpm-ssa-01
ec-cbpm-usa-01
ec-cbpm-usa-02
ec-fb-srf-01
ec-mod-kg01-0
ec-mod-kg02-0
ec-mod-kg03-0

OK | Host: fcsproxy | Database: hdb | User: hdbbrowser

□ Time Fill □ Fetch or

Extracted 60 rows in 0.01124s

QHdbExtractor (++)

○ Lines ○ Dots ● Steps

Extracted 60 rows in 0.002041s

Elettra Sincrotrone Trieste

GUI: ATK/Jdraw/Synoptic

TANGO

Application ToolKit: provides a framework to speed up the development of TANGO applications

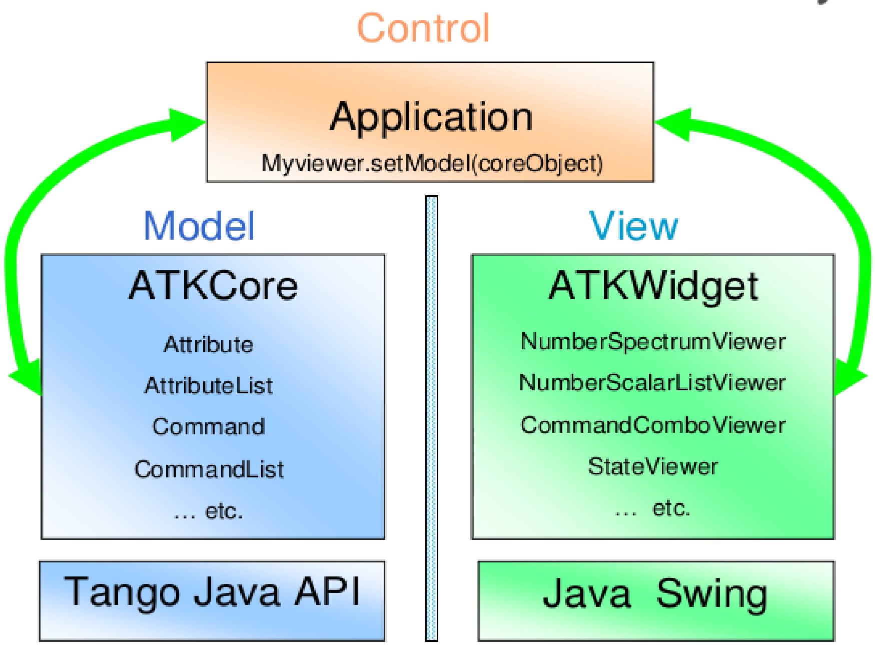
Core of any TANGO Java client

ATKpanel: generic GUI (data introspection)

Use Jdraw to draw the specialized synoptic

Design your own specific ATK application
Using your favorite Java IDE

Final result...



Control

Application
Myviewer.setModel(coreObject)

Model

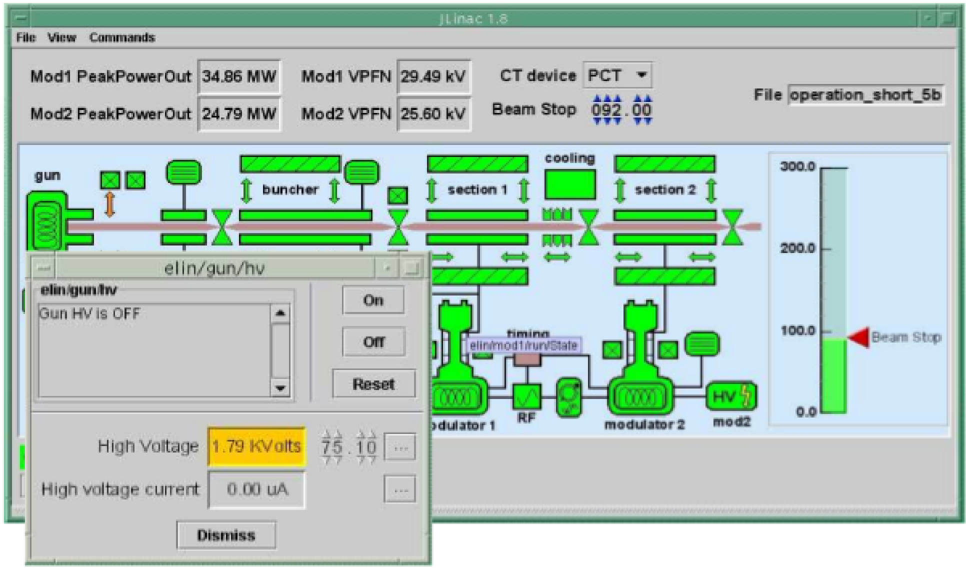
ATKCore
Attribute
AttributeList
Command
CommandList
… etc.

Tango Java API

View

ATKWidget
NumberSpectrumViewer
NumberScalarListViewer
CommandComboViewer
StateViewer
… etc.

Java Swing



Mod1 PeakPowerOut 34.86 MW
Mod2 PeakPowerOut 24.79 MW
Mod1 VPFN 29.49 kV
Mod2 VPFN 25.60 kV
CT device PCT
Beam Stop 092.00
File operation_short_5b

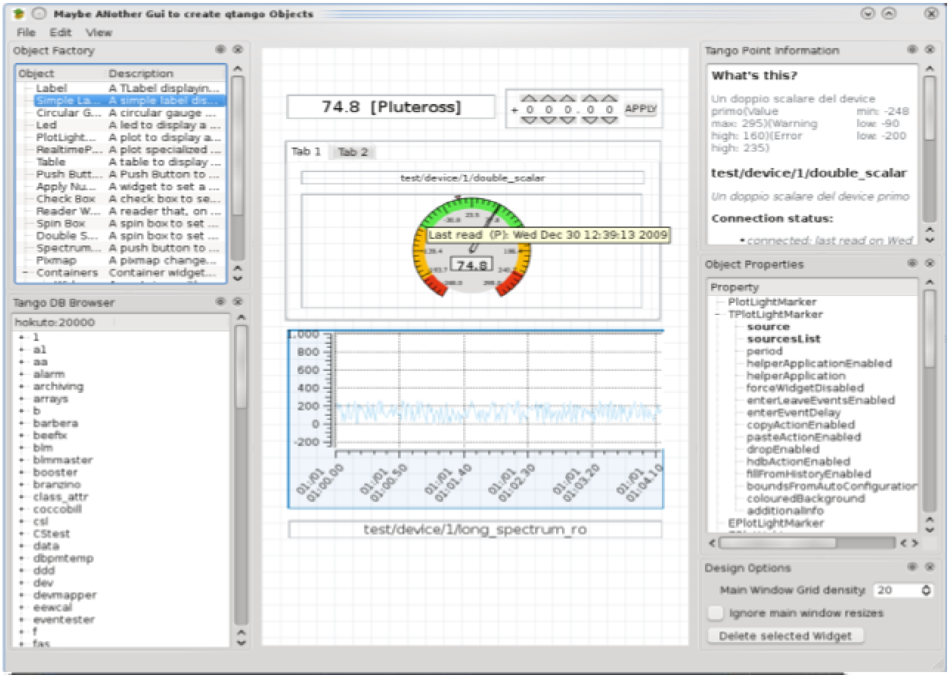
elin/gun/hv
Gun HV is OFF

High Voltage 1.79 KVolts 75.10
High voltage current 0.00 uA

GUI: Qtango/Mango

Qtango
- A multi-threaded framework to develop TANGO applications
- Based on Qt
- API to manage/talk to TANGO devices
- Widgets to draw the GUI
- For programmers

Mango
- An on-line designer to easily create graphical interfaces based on Qtango
- Quick development of simple GUI
- Useful for the device server programmer, the control room operator, the tests, the end-user



Object	Description
Label	A TLabel displayin...
Simple La...	A simple label dis...
Circular G...	A circular gauge ...
Led	A led to display a ...
PlotLight...	A plot to display a...
RealtimeP...	A plot specialized ...
Table	A table to display ...
Push Butt...	A Push Button to ...
Apply Nu...	A widget to set a ...
Check Box	A check box to se...
Reader W...	A reader that, on ...
Spin Box	A spin box to set ...
Double S...	A spin box to set ...
Spectrum...	A push button to ...
Pixmap	A pixmap change...
Containers	Container widget...

hokuto:20000
- 1
- a1
- aa
- alarm
- archiving
- arrays
- b
- barbera
- beefix
- blm
- blmmaster
- booster
- branzino
- class_attr
- coccobill
- csl
- CStest
- data
- dbpmtemp
- ddd
- dev
- devmapper
- eewcal
- eventester
- f
- fas

What's this?

Un doppio scalare del device primo(Value min: -248 max: 295)(Warning low: -90 high: 160)(Error low: -200 high: 235)

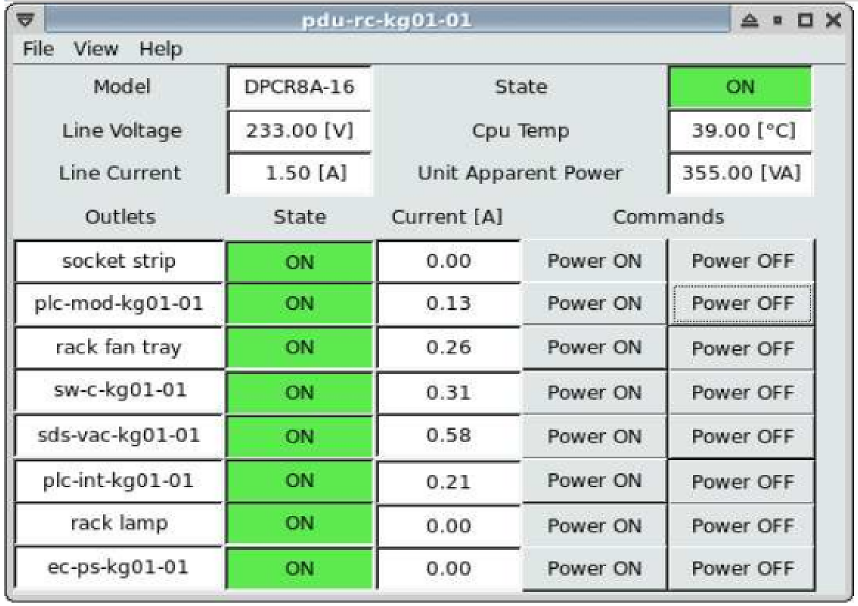
test/device/1/double_scalar

Un doppio scalare del device primo

Connection status:

- *connected: last read on Wed*

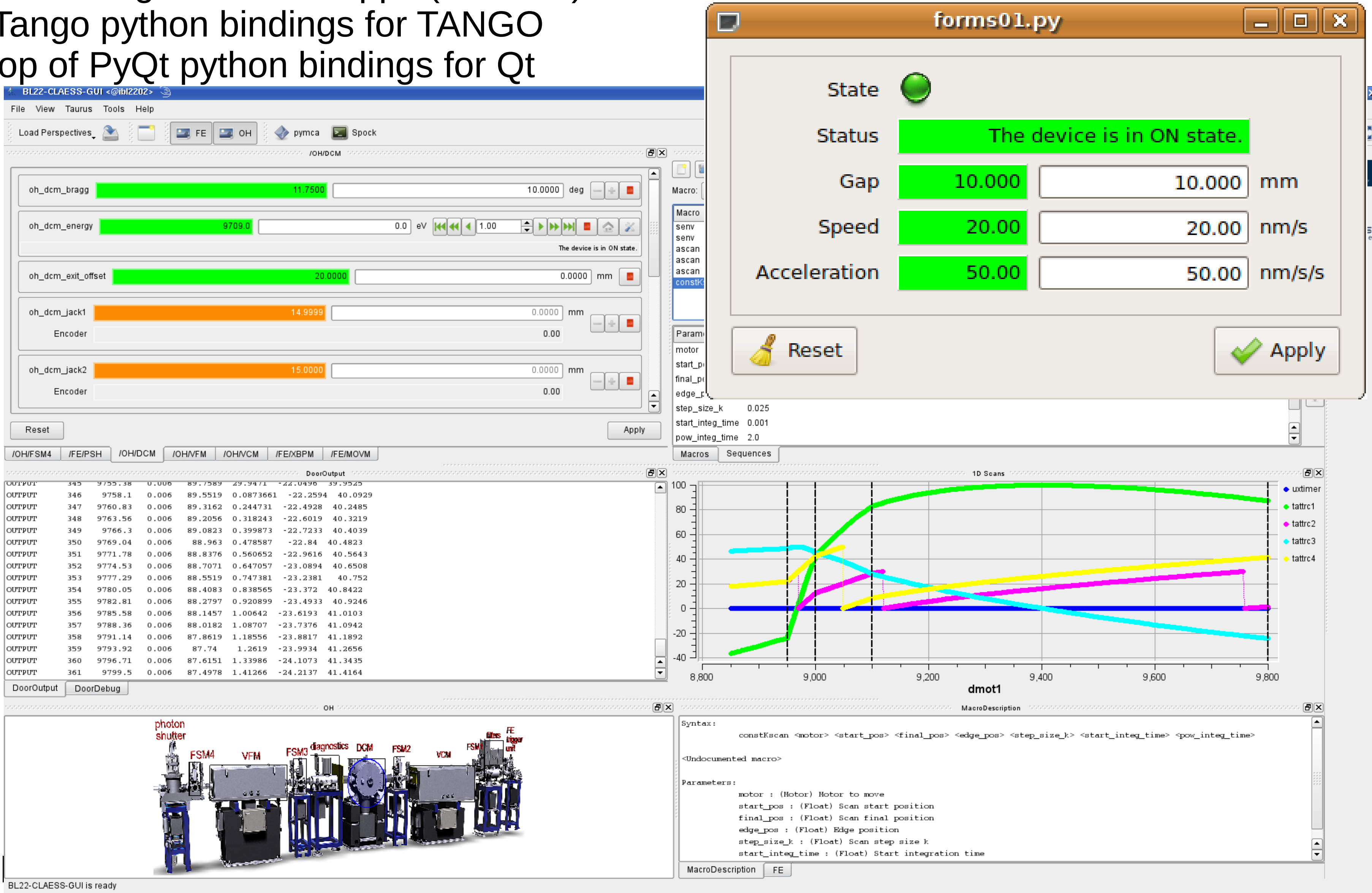
Property
- PlotLightMarker
- TPlotLightMarker
  - **source**
  - **sourcesList**
  - period
  - helperApplicationEnabled
  - helperApplication
  - forceWidgetDisabled
  - enterLeaveEventsEnabled
  - enterEventDelay
  - copyActionEnabled
  - pasteActionEnabled
  - dropEnabled
  - hdbActionEnabled
  - fillFromHistoryEnabled
  - boundsFromAutoConfiguration
  - colouredBackground
  - additionalInfo
- EPlotLightMarker



Model	DPCR8A-16	State	ON	
Line Voltage	233.00 [V]	Cpu Temp	39.00 [°C]	
Line Current	1.50 [A]	Unit Apparent Power	355.00 [VA]	
Outlets	State	Current [A]	Commands	
socket strip	ON	0.00	Power ON	Power OFF
plc-mod-kg01-01	ON	0.13	Power ON	Power OFF
rack fan tray	ON	0.26	Power ON	Power OFF
sw-c-kg01-01	ON	0.31	Power ON	Power OFF
sds-vac-kg01-01	ON	0.58	Power ON	Power OFF
plc-int-kg01-01	ON	0.21	Power ON	Power OFF
rack lamp	ON	0.00	Power ON	Power OFF
ec-ps-kg01-01	ON	0.00	Power ON	Power OFF

GUI: TAURUS

A library for connecting client-side apps (CLI/GUI) to TANGO device servers
Based on PyTango python bindings for TANGO
GUI built on top of PyQt python bindings for Qt



oh_dcm_bragg	11.7500	10.0000	deg	
oh_dcm_energy	9709.0	0.0	eV	1.00
oh_dcm_exit_offset	20.0000	0.0000	mm	
oh_dcm_jack1	14.9999	0.0000	mm	
Encoder	0.00			
oh_dcm_jack2	15.0000	0.0000	mm	
Encoder	0.00			

The device is in ON state.

```
OUTPUT       345    9755.38    0.006    89.7589   29.9471   -22.0496   39.9525
OUTPUT       346     9758.1    0.006    89.5519  0.0873661   -22.2594   40.0929
OUTPUT       347    9760.83    0.006    89.3162  0.244731   -22.4928   40.2485
OUTPUT       348    9763.56    0.006    89.2056  0.318243   -22.6019   40.3219
OUTPUT       349     9766.3    0.006    89.0823  0.399873   -22.7233   40.4039
OUTPUT       350    9769.04    0.006     88.963  0.478587     -22.84   40.4823
OUTPUT       351    9771.78    0.006    88.8376  0.560652   -22.9616   40.5643
OUTPUT       352    9774.53    0.006    88.7071  0.647057   -23.0894   40.6508
OUTPUT       353    9777.29    0.006    88.5519  0.747381   -23.2381    40.752
OUTPUT       354    9780.05    0.006    88.4083  0.838565    -23.372   40.8422
OUTPUT       355    9782.81    0.006    88.2797  0.920899   -23.4933   40.9246
OUTPUT       356    9785.58    0.006    88.1457   1.00642   -23.6193   41.0103
OUTPUT       357    9788.36    0.006    88.0182   1.08707   -23.7376   41.0942
OUTPUT       358    9791.14    0.006    87.8619   1.18556   -23.8817   41.1892
OUTPUT       359    9793.92    0.006      87.74    1.2619   -23.9934   41.2656
OUTPUT       360    9796.71    0.006    87.6151   1.33986   -24.1073   41.3435
OUTPUT       361     9799.5    0.006    87.4978   1.41266   -24.2137   41.4164
```

BL22-CLAESS-GUI is ready

motor	
start_p	
final_p	
edge_p	
step_size_k	0.025
start_integ_time	0.001
pow_integ_time	2.0

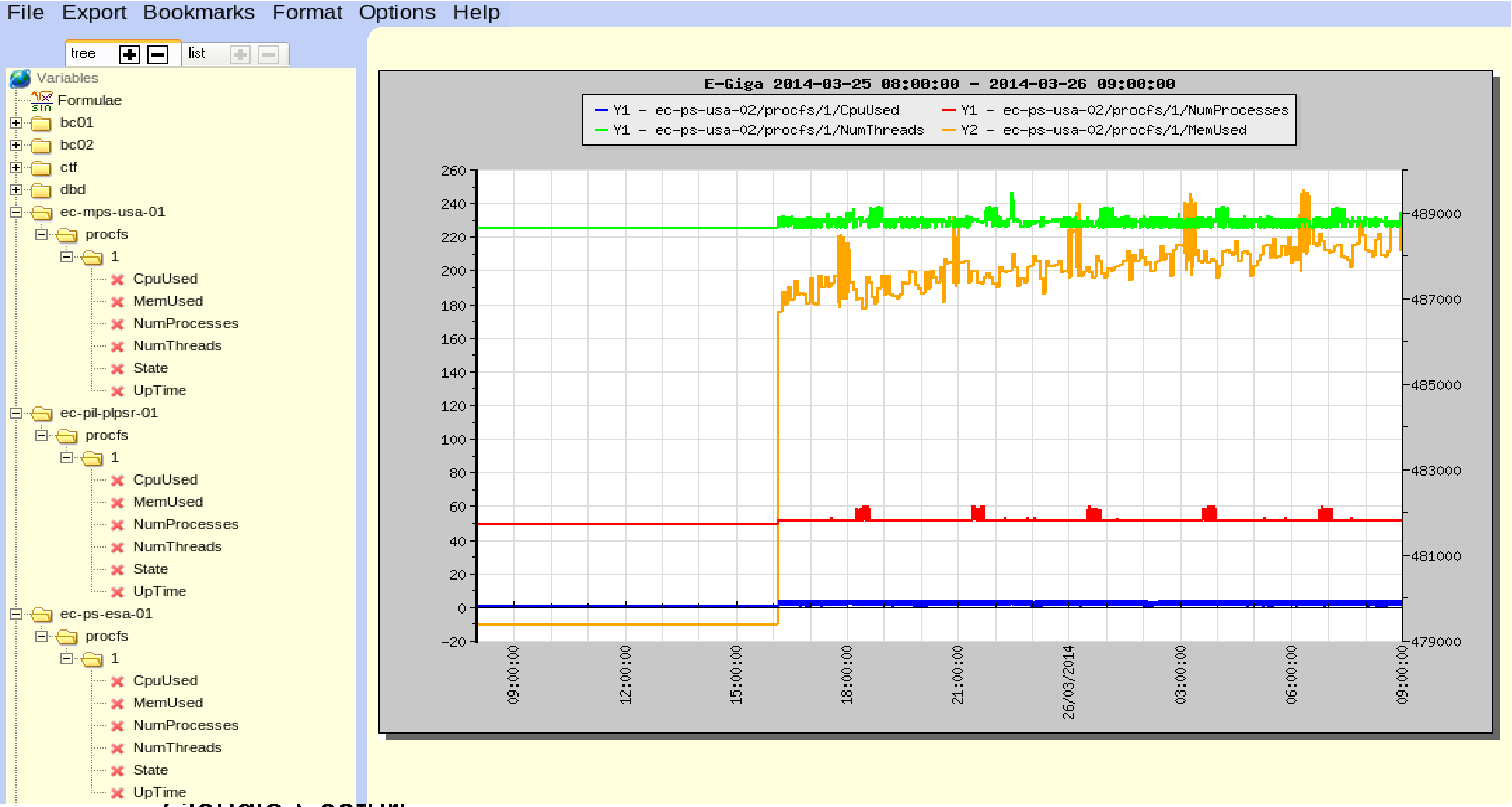
```
Syntax:
        constKscan <motor> <start_pos> <final_pos> <edge_pos> <step_size_k> <start_integ_time> <pow_integ_time>

<Undocumented macro>

Parameters:
        motor : (Motor) Motor to move
        start_pos : (Float) Scan start position
        final_pos : (Float) Scan final position
        edge_pos : (Float) Edge position
        step_size_k : (Float) Scan step size k
        start_integ_time : (Float) Start integration time
```

Status	The device is in ON state.		
Gap	10.000	10.000	mm
Speed	20.00	20.00	nm/s
Acceleration	50.00	50.00	nm/s/s

Cl

E-giga/Canone

E-Giga: a WEB interface to historical archive data
Canone: a tool to develop WEB interfaces to Tango devices



Claudio Scafuri

TANGO bindings

Access TANGO control systems from different high level “programming” environments.

TANGO provides bindings for the following “languages”:

- C language (partial support)
- Matlab (>= R2009b)
  Windows and Linux, 32 and 64 bit
- Octave (>= 3.6.2)
  Windows and Linux, 32 and 64 bit
- LabVIEW 2010 → 2012
  Windows, Linux, MacOSX, 32 and 64 bit
- LabVIEW 2013 (2.0.0 RC2)
  TANGO 8.1.2 with patches; Windows and Linux, 64 bit
- Igor Pro (>= 6.0)
  Windows, Linux, MacOSX, 32 and 64 bit
- Panorama
  Tango 7.2.1, Windows, 32 and 64 bit

TANGO Domains

Each domain is identified by the *TANGO_HOST/port* couple, e.g. by the TANGO Database
An arbitrary number of devices may belong to a domain, limited by

- available memory
- processing power
- network bandwidth

(Operating Database limit ~ $5*10^5$ devices)

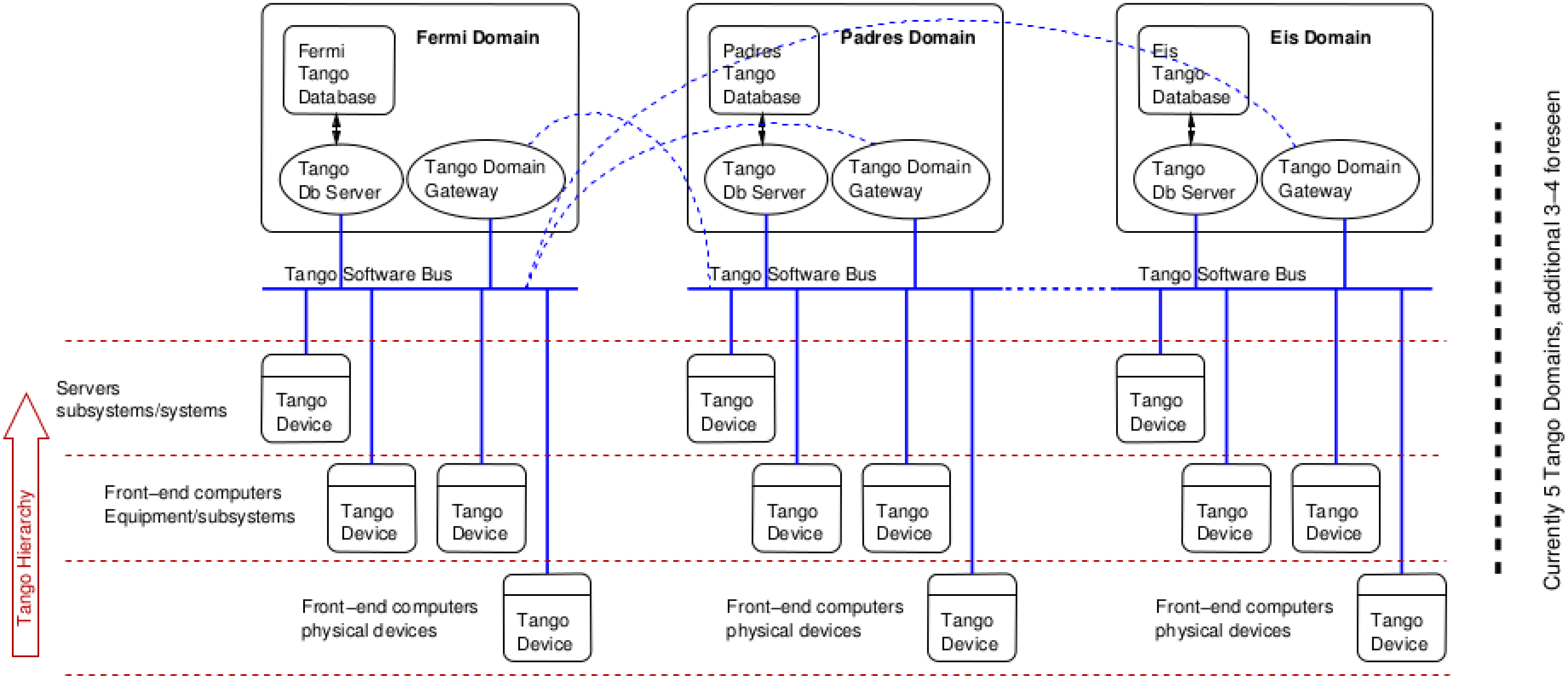
...but...

Multiple domains **can** be configured in a control system

- complex systems ~~can~~ **must** be splitted into different domains
- each Domain ~~can~~ **must** be hierarchically organized

Multiple domains + Device hierarchy + Peer-2-Peer architecture
=
Almost unlimited scalability

TANGO Domains



Hosts:	71
Server types:	193
Server instances:	1092
Classes:	197
Devices:	4437
Control points:	155228

Hosts:	6
Server types:	33
Server instances:	148
Classes:	33
Devices:	661
Control points:	18661

Hosts:	4
Server types:	24
Server instances:	53
Classes:	26
Devices:	127
Control points:	4223

TANGO Domains

Clients can explicitly use host:port for accessing Devices in specified Domains by pre-pending them to the device name:

host:port/domain/family/member

For example:

tom:20000/sr/power_supply/psch_s7.8

Notice :

fermi:20000/sys/database/2

padres:20000/sys/database/2

Same object, the database server, in two different domains!

TANGO 9

TANGO 9

TANGO pipe(s)

- Support for structured data with variable data types
- Variable data type does not fit into the TANGO Attribute model
- TANGO pipes extend the Device interface
- each pipe has:
    - a name, unique for the device
    - a label and a description
    - a description for the input data definition (for the client)
- the pipe transports a **blob** of data
- each blob is a set of data elements
- each element
    - **has** a name
    - **is** a TANGO basic type (or array thereof)
- compared to Command and Attribute pipe(s) have less features:
    - no polling
    - no alarm
    - no quality factor
    - no change/periodic/archive event
    - no TANGO group
- client access to a pipe can be:
    - synchronous: write query and wait for answer
    - event based: register a callback executed when the device writes in the pipe

TANGO 9

TANGO 9

Enumeration as Attribute data type

Many parameters in the hardware have a limited set of values, with a label describing it

Forwarded attribute

- High level TANGO devices often need to “map” Attribute coming fro low level TANGO devices
- A forwarded Attribute is an Attribute which **forwards**
    - its read/write requests
    - its configuration
    - its polling
    - its event subscription

  to another Attribute
- has have the same data type, data format, read/write type of the “root” attribute
- no code is required

Installing TANGO

deb packages – Ubuntu 14.04 LTS

libtango8 - TANGO distributed control system - shared library
liblog4tango5 - logging for TANGO - shared library
libtango-tools - TANGO distributed control system - common executable files
tango-db - TANGO distributed control system - database server
tango-starter - TANGO distributed control system - starter server
tango-common - TANGO distributed control system - common files

tango-accesscontrol - TANGO distributed control system - accesscontrol server
python-pytango - API for the TANGO control system (Python 2)
python-sardana - sardana control system
python-taurus - framework for Tango Control System CLI and GUI applications

from source (tarball)

omniORB-4.1.7.tar.bz2
zeromq-3.2.3.tar.gz
tango-8.1.2c.tar.gz

Documentation

TANGO Controls System Handbook

http://ftp.esrf.fr/pub/cs/tango/tango_81.pdf

TANGO Device Server Guidelines

http://www-controle.synchrotron-soleil.fr:8001/docs/TangoGuidelines/TangoDesignGuidelines-GB4-3.pdf

TANGO Java Device Server User Guide

http://www2.synchrotron-soleil.fr/controle/maven2/soleil/org/tango/JTangoServer

C++ API classes reference guide

http://www.esrf.eu/computing/cs/tango/tango_doc/kernel_doc/cpp_doc/index.html
http://www.esrf.eu/computing/cs/tango/tango_doc/kernel_doc/ds_prog/node7.html

Java API classes reference guide

http://www.esrf.eu/computing/cs/tango/tango_doc/kernel_doc/tango_java_api/index.html
http://www2.synchrotron-soleil.fr/controle/maven2/soleil/org/tango/JTangoServer

Python classes reference guide

http://www.esrf.fr/computing/cs/tango/tango_doc/kernel_doc/pytango

TANGO IDL file documentation

http://www.esrf.fr/computing/cs/tango/tango_idl/idl_html/index.html

Source code repository

TANGO Controls - https://sourceforge.net/projects/tango-cs/?source=directory
TANGO device servers - https://sourceforge.net/projects/tango-ds/?source=directory

Many additional resources on the TANGO site
http://www.tango-controls.org/