Tango Training

Tango Training

- Introduction (1)
- Device and device server (2)
- Writing device server and client (the basic) (3 – 5)
- Events (6)
- Device server level 2 (7)
- Advanced features (8)
- GUIs (9)
- Archiving system (10)
- Miscellaneous (11)

Tango Training:
Part 1 : Introduction

- What isTango?
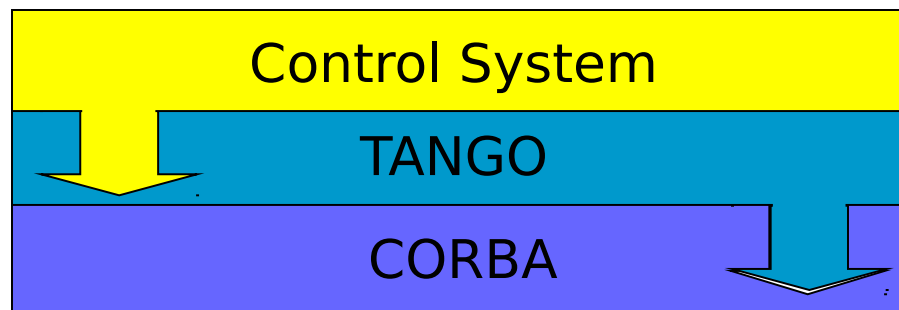- Collaboration
- Languages/OS/compilers
- CORBA

What is Tango?

■ A CORBA framework for doing controls
- A toolbox to implement a control system
- A specialization of CORBA adapted to Control
- Hide the complexity of Corba to the programmer
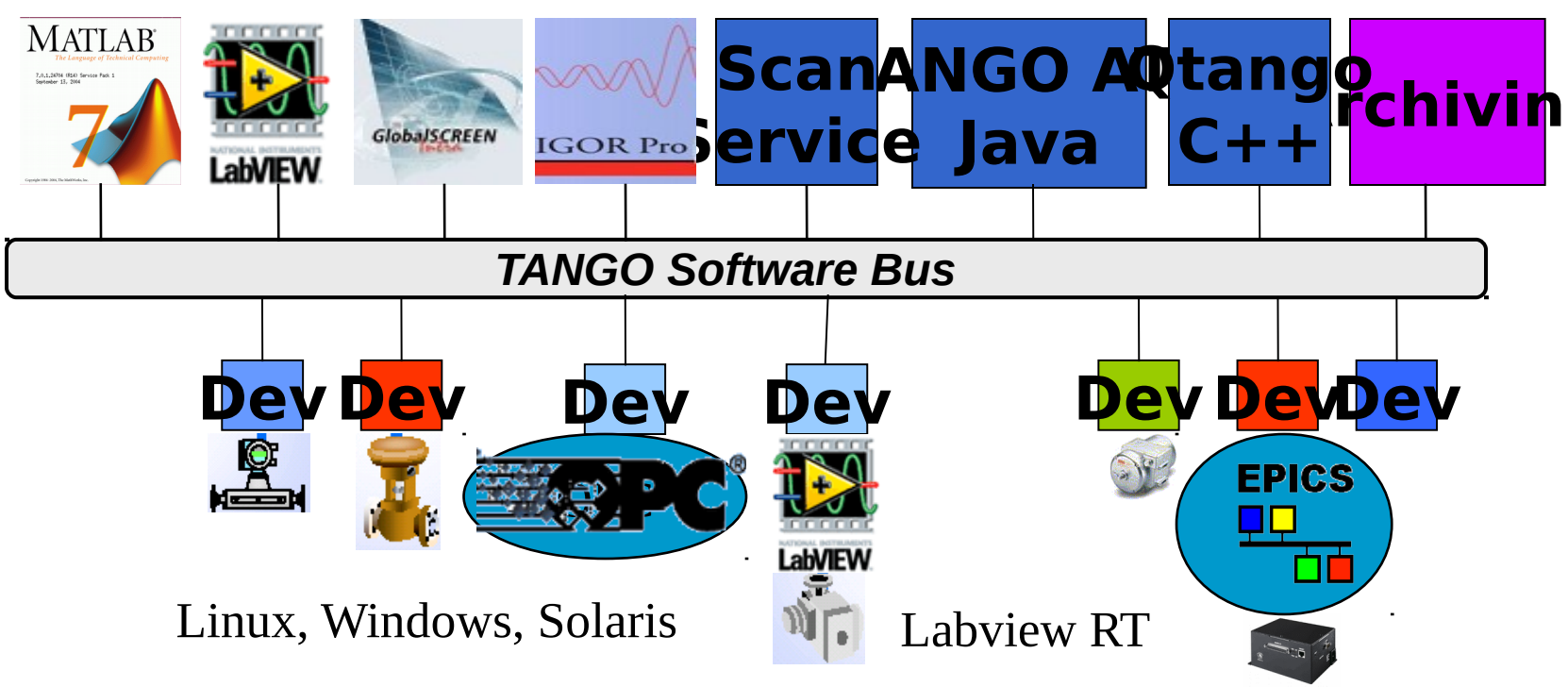- Adds specific contol system features

Control System
TANGO
CORBA

04/28/10

4

What is Tango?

■ A software bus for distributed objects

Java, C++,Python   Linux, Windows, Solaris

| MATLAB | LabVIEW | GlobalSCREEN | IGOR Pro | Scan Service | ANGO AQ Java | Qtango C++ | rchiving |

TANGO Software Bus

Dev **Dev**   **Dev**   **Dev**   **Dev** **Dev** **Dev**

EPICS

Linux, Windows, Solaris   Labview RT

What is Tango?

- Provides a unified interface to all equipments, hiding how they are connected to a computer (serial line, USB, sockets….)

- Hide the network

- Location transparency

- Tango is one of the Control System available today but other exist (EPICS…)

04/28/10

The Tango Collaboration

- Tango collaboration history
  - Started in 2000 at ESRF
  - In 2002, Soleil joins ESRF to develop Tango
  - End 2003, Elettra joins the club
  - End 2004, Alba also joins
  - 2006: Hasilab, GKSS will use Tango for Petra 3 beamlines
  - 2009: MAX-lab will use it for Max 4
  - 2009: LMJ uses it for target diagnostics
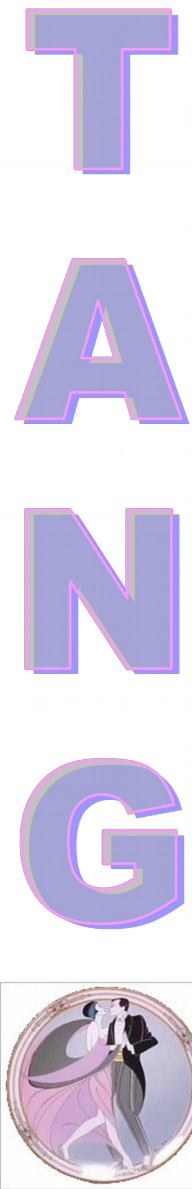 - 2010: FRM II moves from Taco to Tango

The Tango Collaboration

■ How it works:

– Two collaboration meetings per year

– A mailing list (tango@esrf.fr)

– One Tango coordinator per site

– WEB site to download code, get documentation, search the mailing list history, read collaboration meeting minutes…

http://www.tango-controls.org

– Collaborative development using SourceForge

Language/OS/compilers

- Tango is now (June 2010) at release 7.1
  - The training is based on the features of this release.
- Languages/Commercial tools

	C++	Java	Python	Matlab	LabView	IgorPro
Client	OK	OK	OK	OK	OK	OK
Server	OK	OK ***	OK			

Language/OS/Compilers
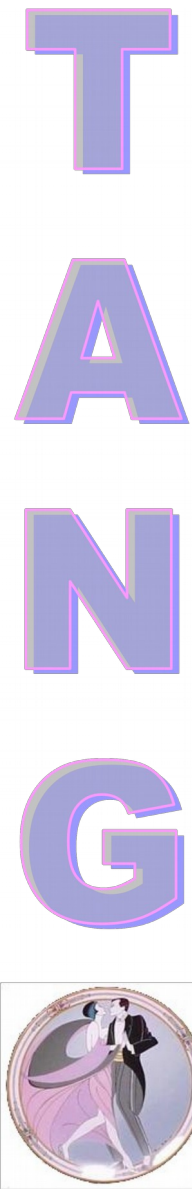
- Linux (32 / 64 bits)
  - Redhat E4.0 / E5.0, Ubuntu 9.04 and 9.10 (Suse at Alba)
  - gcc
- Solaris
  - Solaris 9 + CC
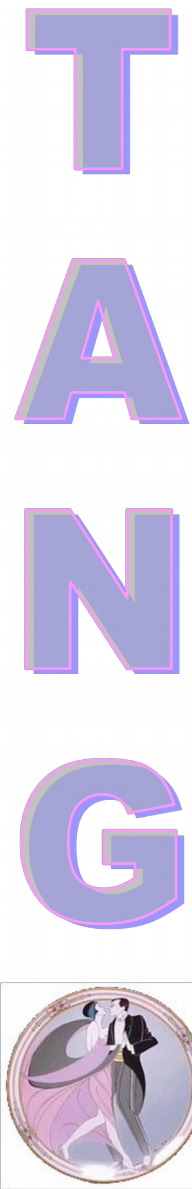 - Solaris 9 + gcc
- Windows
  - Windows XP / Vista with VC8 / VC9

CORBA

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
  - Promoted by OMG
  - It's just paper, not software
- CORBA defines the ORB: a way to call an object "method" wherever the object is
  - In the same process
  - In another process
  - In a process running somewhere on the network
- CORBA also defines services available for all objects (event, naming, notification)

CORBA

- CORBA allows mixing languages: a client is not necessarily written in the same language as server

- CORBA uses an Interface Definition Language (IDL)

- CORBA defines bindings between IDL and computing languages (C++, Java, Python, Ada….)

- It uses IOR (Interoperable Object Reference) to locate an object
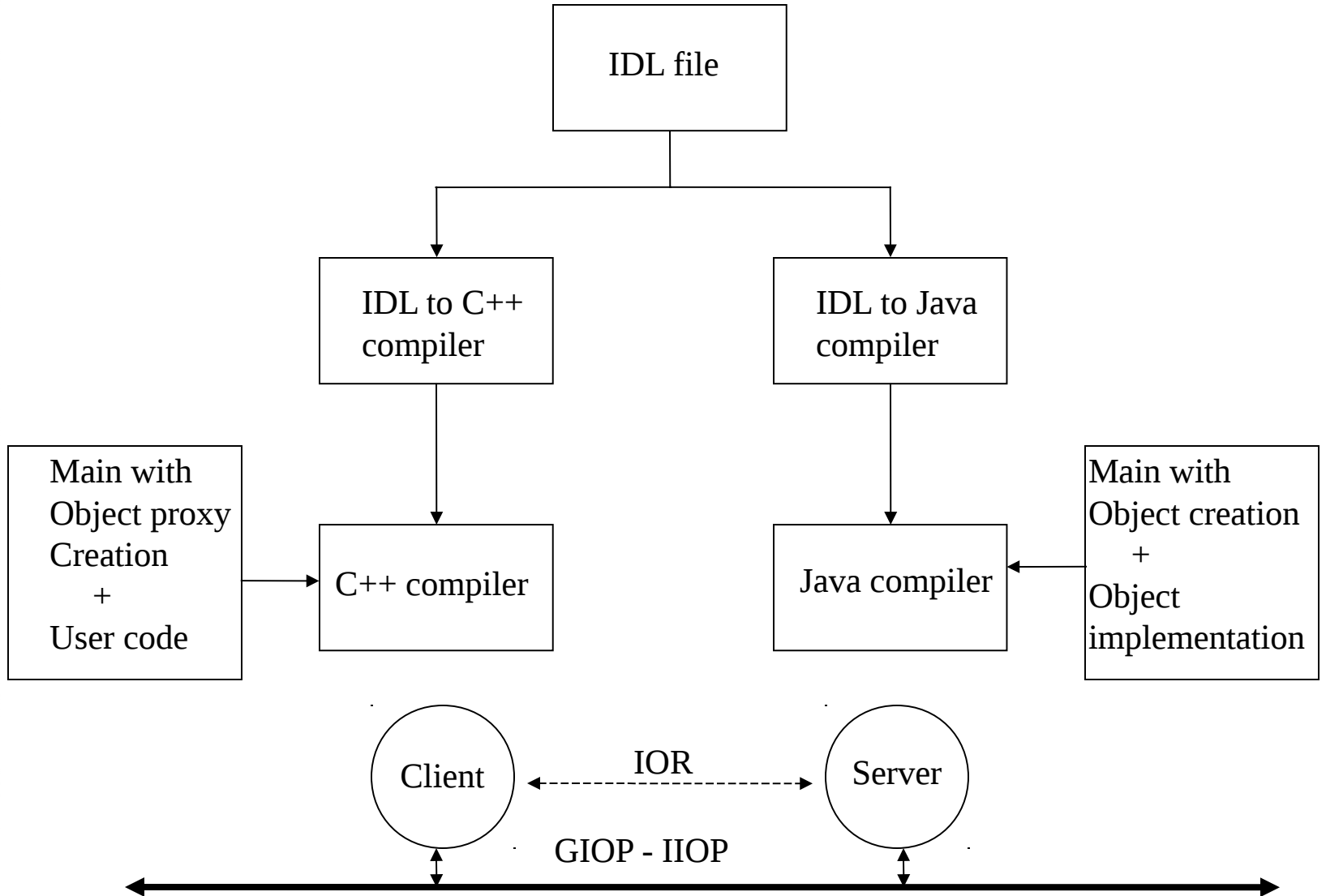
CORBA

- IDL for a remote controlled car

```
interface remote_car
{
    void go_forward(void);
    void go_backward(void);
    void stop(void);
    void turn(float angle);
};
```

CORBA


```
                        ┌─────────────────┐
                        │                 │
                        │    IDL file     │
                        │                 │
                        └────────┬────────┘
                                 │
                ┌────────────────┴────────────────┐
                ▼                                  ▼
      ┌──────────────────┐              ┌──────────────────┐
      │  IDL to C++      │              │  IDL to Java     │
      │  compiler        │              │  compiler        │
      └────────┬─────────┘              └────────┬─────────┘
               │                                 │
               ▼                                 ▼
```

| Main with Object proxy Creation + User code | → C++ compiler | Java compiler ← | Main with Object creation + Object implementation |

Client ◄------ IOR ------► Server

GIOP - IIOP

CORBA

- Many CORBA ORB and services available
- Tango uses
  - omniORB for C++ ORB (http://omniorb.sourceforge.net)
  - JacORB for Java ORB (http://www.jacorb.org)
  - omniNotify for CORBA notification service (http://omninotify.sourceforge.net)
  - Boost python for PyTango (1.41)

Tango Training:
Part 2 : Device and Device Server

- The Tango device
- The Tango device server
- A minimum Tango System

The Tango Device

- The fundamental brick of Tango is the device!
- Everything which needs to be controlled is a "device" from a very simple equipment to a very sophisticated one
- Every device has a three field name "domain/family/member"
  - sr/v-ip/c18-1, sr/v-ip/c18-2
  - sr/d-ct/1
  - id10/motor/10

Some device(s)

One device

One device
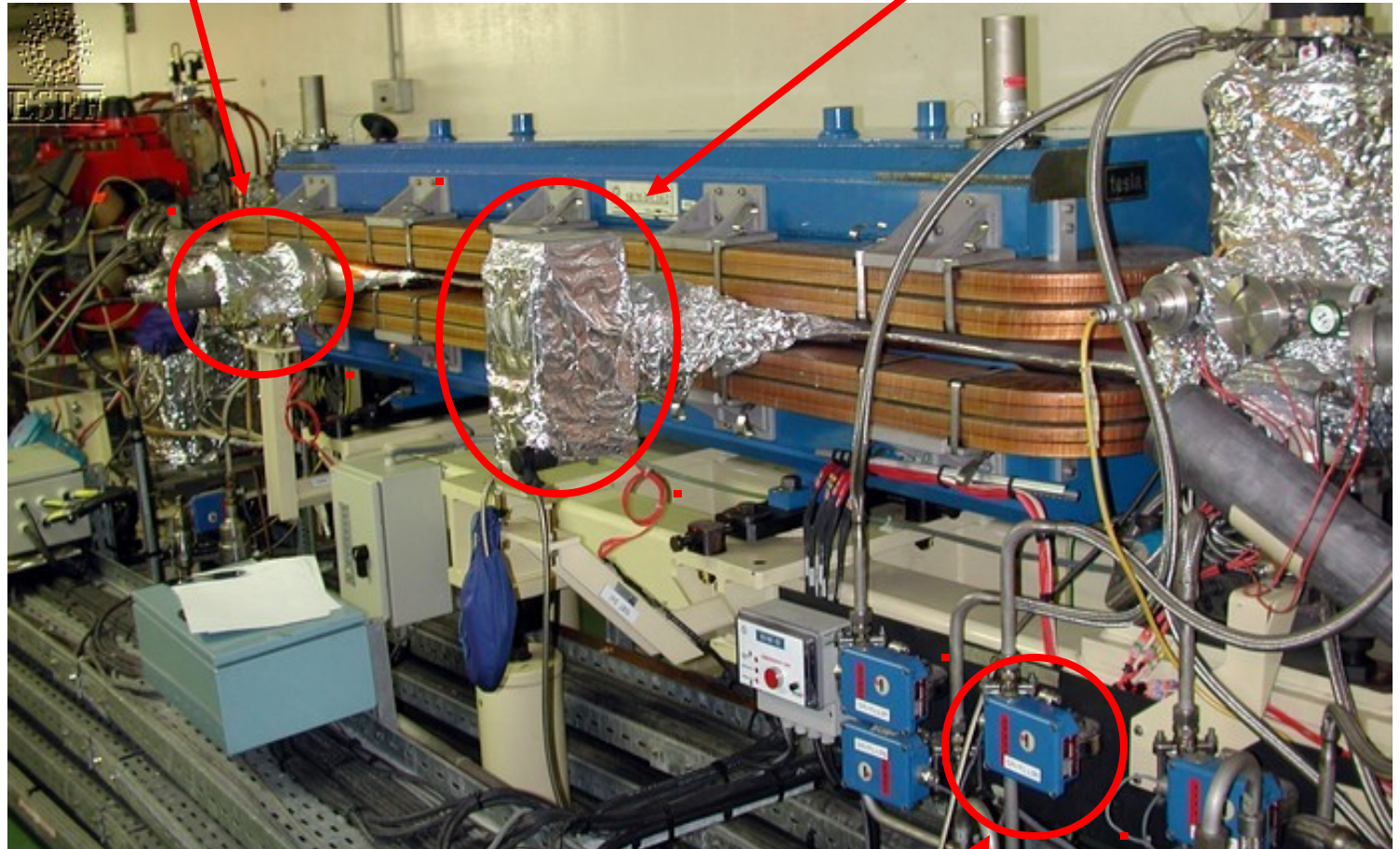
One device

A sophisticated device (RF cavity)
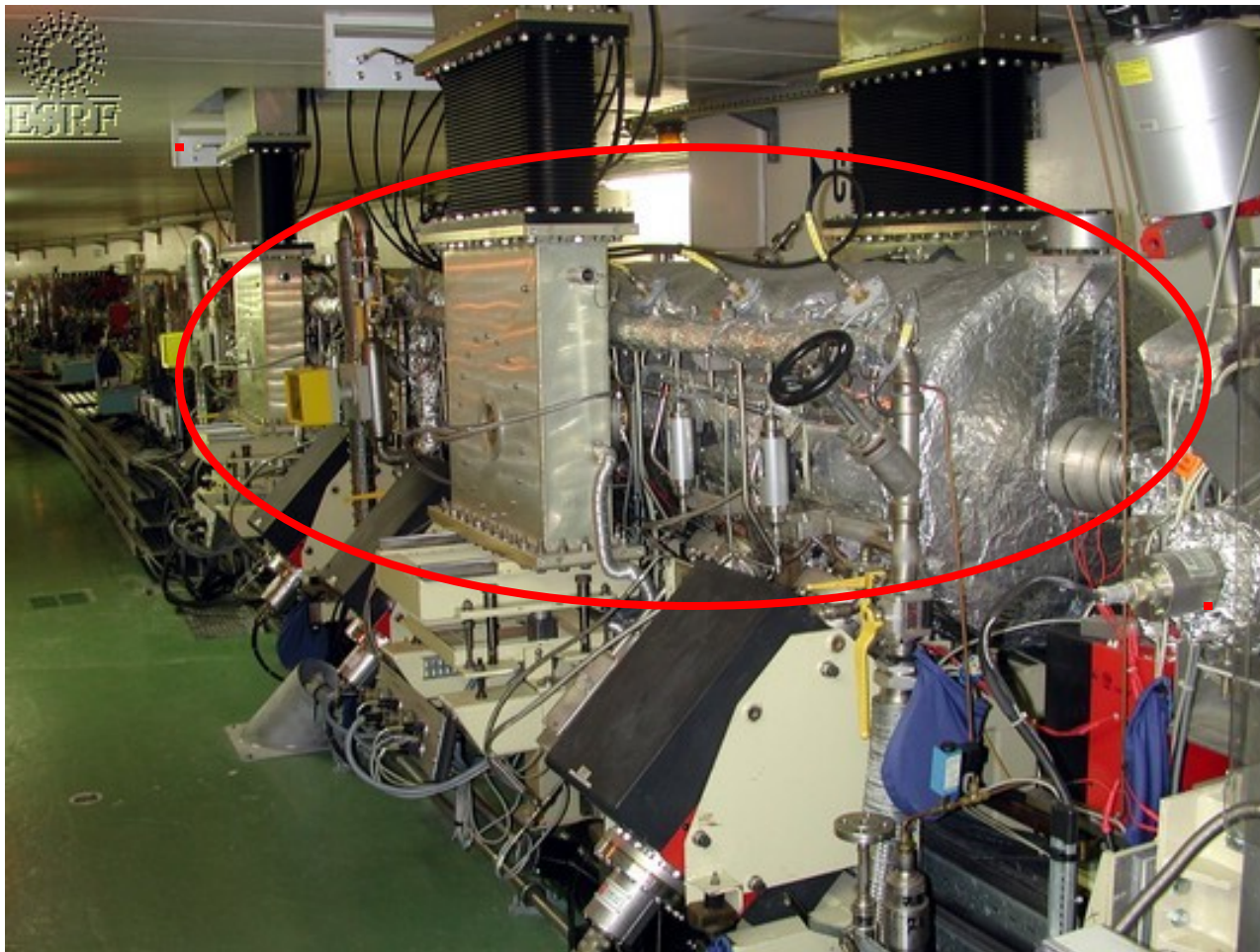
another
device

The Tango Class

- Every device belongs to a Tango class (not a computing language class)

- Every device inherits from the same root class (DeviceImpl class)

- A Tango class implements the necessary features to control one kind of equipment
  - Example : The Agilent 4395a spectrum analyzer controlled by its GPIB interface

The Tango Device Server

■ A Tango device server is the process where the Tango class(es) are running.

A Tango device server

Tango device class A

Tango device class B

Device sr/v-ip/1

Device sr/v-ip/2

Device id4/mot/1

Device id4/mot/2

Device id4/mot/3

"ps" command shows one device server
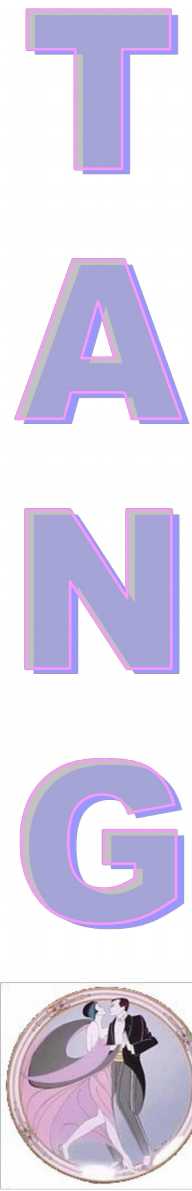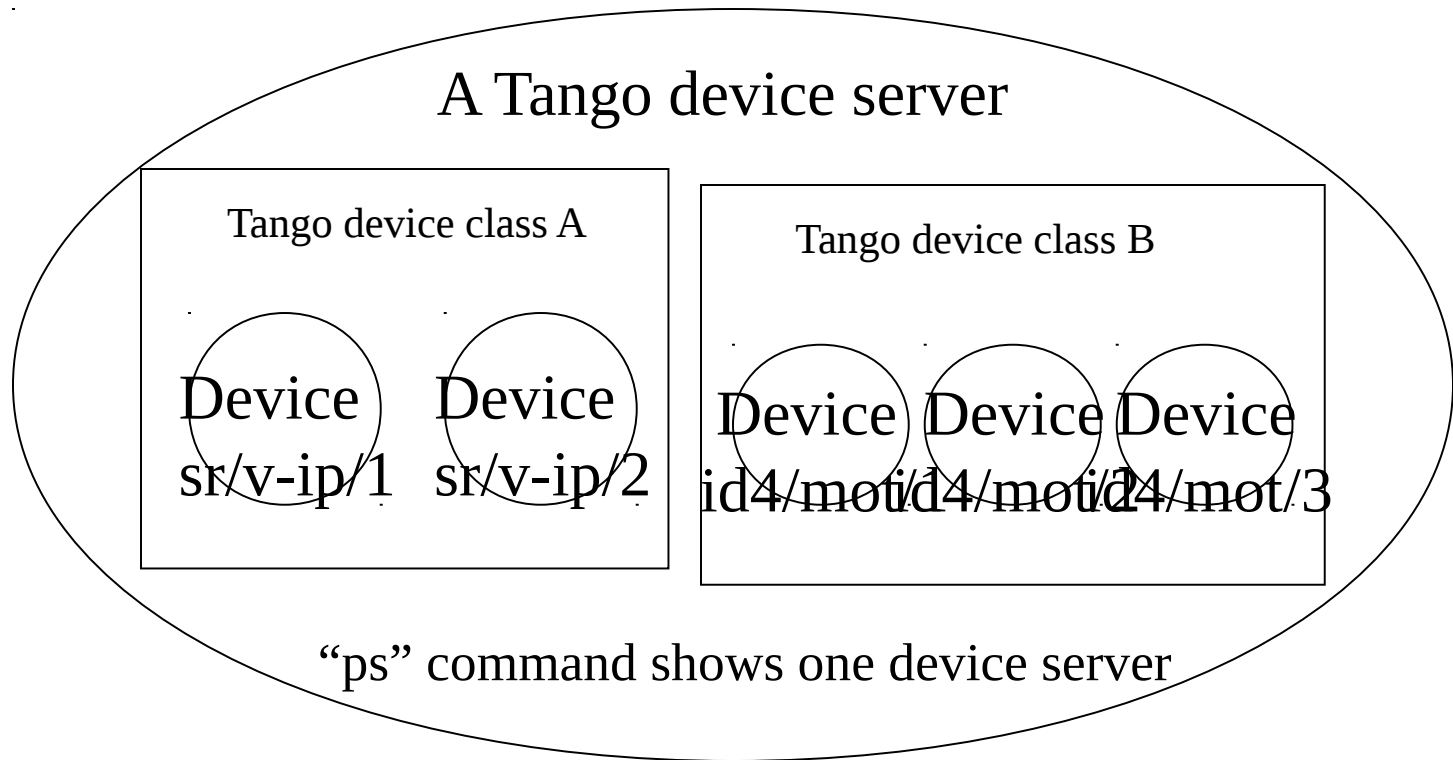
The Tango Device Server

- Tango uses a database to configure a device server process

- 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 device server process is defined in the database but also in the code (training part 6)

The Tango Device Server

- Each device server is defined by the couple "executable name / instance name"

sr/v-ip/c8-1 to sr/v-ip/c8-5

■ One vacuum pump

sr/v-ip/c9-1 to sr/v-ip/c9-5

sr/v-ip/c10-1 to sr/v-ip/c10-5

Crate X

VP-DS VP-DS

Crate X+1

VP-DS VP-DS

sr/v-ip/c11-1

sr/v-ip/c11-5
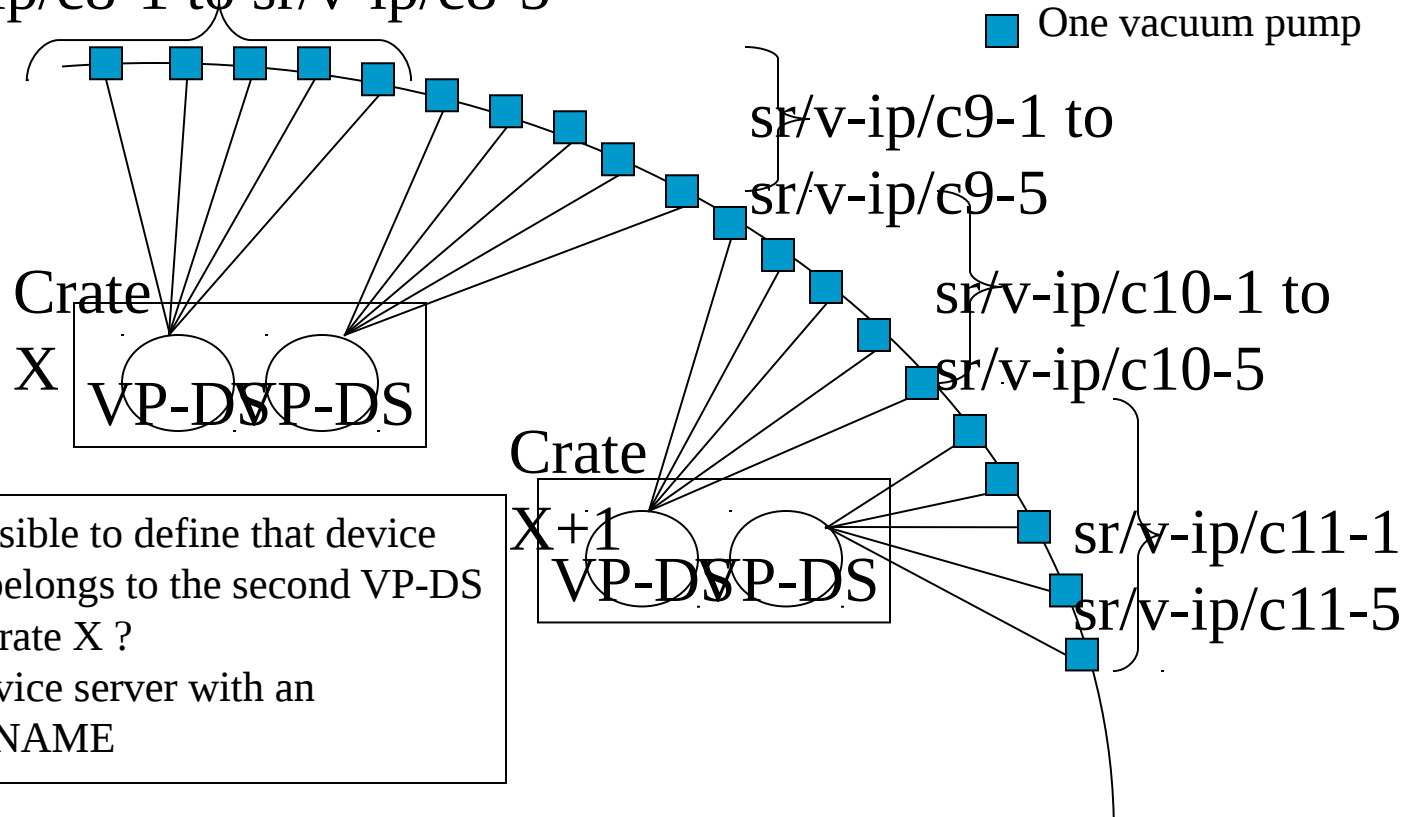
How is it possible to define that device sr/v-ip/c9-3 belongs to the second VP-DS running on Crate X ?
Start each device server with an INSTANCE NAME

The Tango Device Server

■ During its startup sequence, a Tango device server asks the database which devices it has to create and to manage (number and names)

■ Device servers are started like
  ➢ VP-DS c8
  ➢ VP-DS c10

DS exec name	Inst name	Class name	Device name
VP-DS	c8	RibberPump	sr/v-ip/c8-1
VP-DS	c8	RibberPump	sr/v-ip/c8-2
VP-DS	c8	RibberPump	sr/v-ip/c8-3

04/28/10

24

A minimum Tango System

- To run a Tango control system, you need
  - A running MySQL database
  - The Tango database server
    - It is a C++ Tango device server with one device
- To start the database server on a fixed port
- The environment variable **TANGO_HOST** is used by client/server to know
  - On which **host** the database server is running
  - On which **port** it is listening

A minimum Tango System

DataBaseds   2   −ORBendPoint   giop:tcp:host:10000

TANGO_HOST=host:port (Ex : TANGO_HOST=orion:10000)

```
                            Database
                             server
       Get device(s) IOR                    Send device(s) IOR

         Tango            CORBA requests          Device
         client                                   server
                  Execute cmd/read-write attribute
```
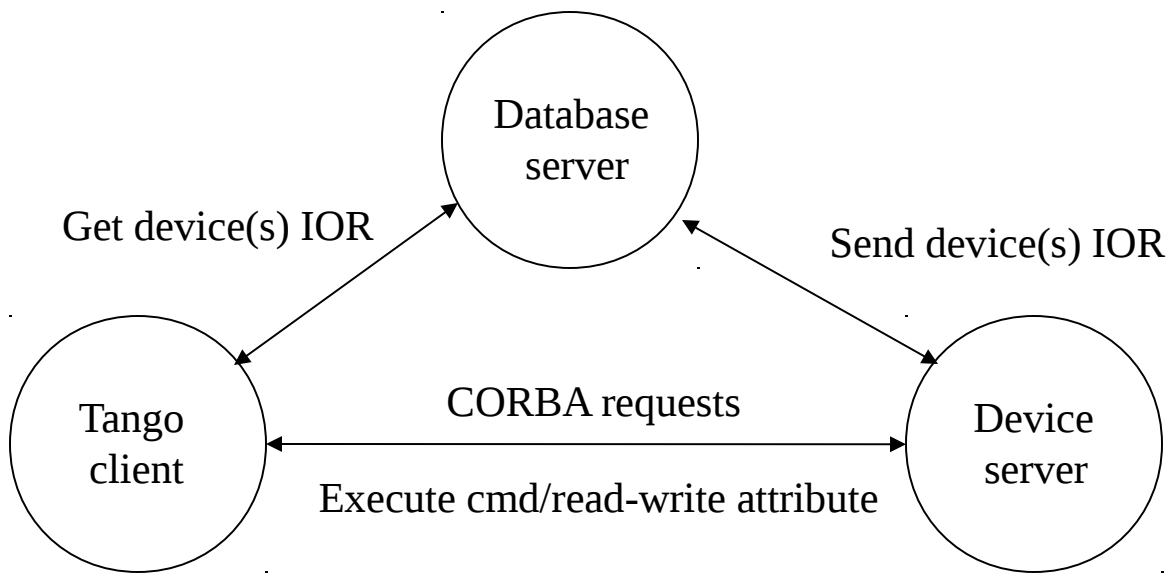
Tango Training: Part 3 : Writing a device server

- Tango device command/attributes
- Coding a Tango class
- Errors
- Properties

Tango Device

T
A
N
G

- Each Tango device is a CORBA object
- Each Tango device supports the same network interface
- What do we have in this interface ?

Command/Attribute

- On the network a Tango device mainly has
  - **Command**(s): Used to implement "action" on a device (switching ON a power supply)
  - **Attribute**(s): Used for physical values (a motor position)
- Clients ask Tango devices to execute a command or read/write one of its attributes
- A Tango device also has a **state** and a **status** which are available using command(s) or as attribute(s)
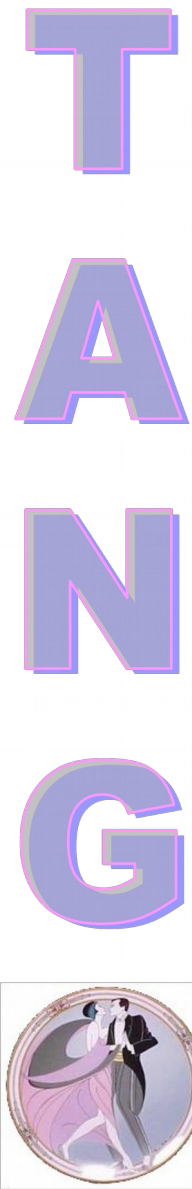
Tango Device Command

- A command may have one input and one output argument.
- A limited set of argument data types are supported
  - Boolean, short, long, long64, float, double, string, unsigned short, unsigned long, unsigned long64, array of these, 2 exotic types and State data type

Tango Device Attribute

- Self describing data via a configuration
- Thirteen data types supported:
  - Boolean, unsigned char, short, unsigned short, long, long64, unsigned long, unsigned long64, float, double, string, state and DevEncoded data type
- Three accessibility types
  - Read, write, read-write
- Three data formats
  - Scalar (one value), spectrum (an array of one dimension), image (an array of 2 dimensions)
- Tango adds 2 attributes which are state and status

Tango Device Attribute

- When you read an attribute you receive:
  - The attribute data (luckily…)
  - An attribute quality factor
    - ATTR_VALID, ATTR_INVALID, ATTR_CHANGING, ATTR_ALARM, ATTR_WARNING
  - The date when the attribute was read (number of seconds and usec since EPOCH)
  - Its name
  - Its dimension, data type and data format
- When you write an attribute, you send
  - The new attribute data
  - The attribute name

Device Attribute Configuration

■ Attribute configuration defined by its properties

  – Five type of properties

    • Hard-coded

    • Modifiable properties

      – GUI parameters

      – Max parameters

      – Alarm parameters

      – Event parameters

■ A separate network call allows clients to get attribute configuration (get_attribute_config)

Device Attribute Configuration

- The hard coded attribute properties (8)
  - name
  - data_type
  - data_format
  - writable
  - max_dim_x
  - max_dim_y
  - display level
  - (writable_attr_name)

Device Attribute Configuration

- The GUI attribute properties (6)
  - Description
  - Label
  - Unit
  - Standard_unit
  - Display_unit
  - Format (C++ or printf)
- The Maximum attribute properties (used only for writable attribute) (2)
  - min_value
  - max_value

Device Attribute Configuration

- The alarm attribute properties (6)
  - min_alarm, max_alarm
  - min_warning, max_warning
  - delta_t, delta_val
- The event attribute properties (6)
  - period (for periodic event)
  - rel_change, abs_change (for change event)
  - period, rel_change, abs_change (for archive event)

Tango Device State

- A limited set of 14 device states is available.
  - ON, OFF, CLOSE, OPEN, INSERT, EXTRACT, MOVING, STANDBY, FAULT, INIT, RUNNING, ALARM, DISABLE and UNKNOWN
- All defined within an enumeration.

Writing a Tango Device Class

- Writing Tango device class need some glue code. We are using a code generator with a GUI called **POGO** : **P**rogram **O**bviously used to **G**enerate **O**bjects

- Following some simple rules, it's possible to use it during all the device class development cycle (not only for the first generation)

- POGO generates
  - C++, Python and Java Tango device class glue code
  - Makefile (C++)
  - Basic Tango device class documentation (HTML)

A Tango Device Class (example)

■ A ski lift class
  - 3 states
    • ON, OFF, FAULT (OFF at startup)
  - 3 commands

Name	In	Out	Allowed
Reset	Void	Void	If FAULT
On	Void	Void	If OFF
Off	Void	Void	Always

  - 3 attributes

Name	type	format	Writable
Speed	double	scalar	Read/Write
Wind_speed	double	scalar	Read
Seats_pos	long	spectrum	Read

04/28/10

Exercise 1

■ Generate a MaxLabPowerSupply class with Pogo
  - 3 states:
    • **ON**, **OFF**, **FAULT**, **ALARM**
    • OFF at startup
  - 4 commands:
    - **On** to switch device ON
      - allowed when state is OFF
    - **Off** to switch device OFF
      - allowed only when state is ON or ALARM
    - **Reset** to reset the device in case of a FAULT
      - allowed only when state is FAULT
    - **SendCmd** to send low-level command. Expert only. Input arg = DEV_STRING, output arg = DEVVAR_LONGSTRINGARRAY
      - Allowed only when OFF
  - 3 attributes:
    • **Current**: read/write – scalar – double - memorized
    • **Voltage**: read/write – scalar - double
    • **CurrentSetPoint**: read – scalar – double
■ Generate the documentation

Python Binding

- Based on the C++ API and boost for the C++ to Python link (http://www.boost.org/)

Python
Boost library
Tango python binding library
Tango C++ libraries

libboost_python.so

_PyTango.so

libtango.so and liblog4tango.so

Network
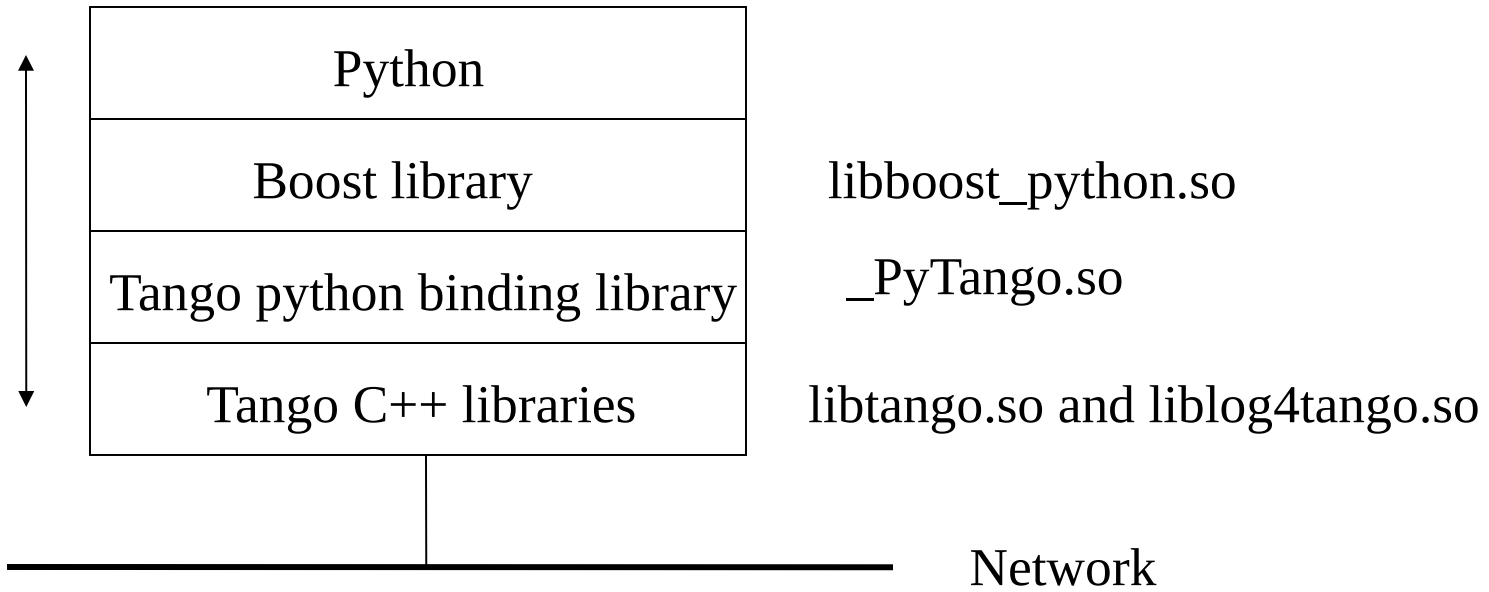
Python Binding

- Module name = **PyTango** and its actual release is 7.1.1 (PyTango.Release.version)
- To use it, you need to have:
  - In your LD_LIBRARY_PATH
    - The boost release 1.41 (or more) library
    - The Tango and ORB libraries
  - In your PYTHONPATH
    - The PyTango python package

Coding a Tango Device Class

■ Four things to code
- Device creation
- Implementing commands
- Reading attributes
- Writing attributes

04/28/10

43

Coding a Tango Class

- For the SkiLift class, Pogo has created 2 files
  - SkiLift.py
  - TangoClassID.txt
- Only SkiLift.py has to be modified

Coding a Tango Class

- Which methods can I use within a Tango class?
  - SkiLift class inherits from a Tango class called Device_<x>Impl
    - All the methods from Device_<x>Impl class which are wrapped to Python
  - Some methods received a Attribute or WAttribute object
    - All the methods of these two classes wrapped to Python
- Doc available at http://www.tango-controls.org
  - *Documents/Tango Kernel/PyTango* for Python classes
  - *Documents/Tango Kernel/Tango device server classes* for Cpp classes

Creating the Device (constructor)

- A init_device() method to construct the device
  - **SkiLift.init_device()**
- A delete_device() to destroy the device
  - **SkiLift.delete_device()**
- All resources acquired in init_device() must be returned in delete_device()

Creating the Device (constructor)

- The init_device() method
  - Init state and status
  - Init (create) local data

```
#-------------------------------------------------------------
-------
#   Device initialization
#-------------------------------------------------------------
-------
    def init_device(self):
        print "In ", self.get_name(),
"::init_device()"
        self.set_state(PyTango.DevState.OFF)
```

Creating the Device

■ The delete_device() method

  – Delete memory/resources allocated in init_device

```
#-------------------------------------------------------------------
#    Device destructor
#-------------------------------------------------------------------
   def delete_device(self):
      print "[Device delete_device method] for device",self.get_name()
```

Implementing a Command

■ One method always_executed_hook() for all commands

  – **SkiLift.always_executed_hook()**

■ If state management is needed, one is_xxx_allowed() method

  – **bool SkiLift.is_reset_allowed()**

■ One method per command

  – **SkiLift.reset()**

Implementing a Command

■ Reset command sequencing
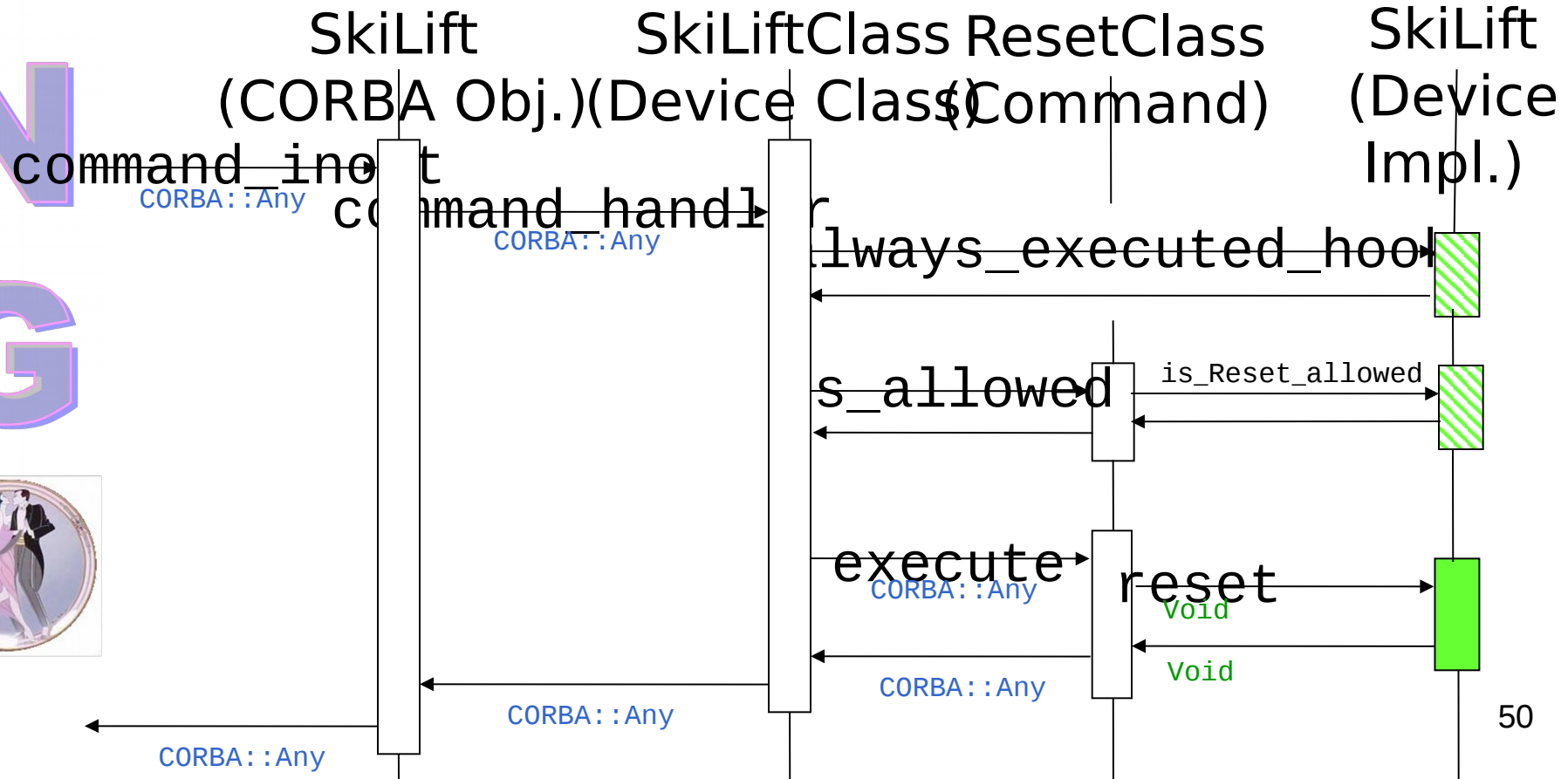
Implementing a Command

■ SkiLift.is_Reset_allowed method coding

```
#---- Reset command State Machine ---------------
  def is_Reset_allowed(self):
    if self.get_state() in [PyTango.DevState.ON,
                  PyTango.DevState.OFF]:
    #     End of Generated Code
    #     Re-Start of Generated Code
      return False
    return True
```

Implementing a Command

■ SkiLift.reset command coding

```
#----------------------------------------------------------------
#    Reset command:
#
#    Description: Reset the ski lift device
#
#----------------------------------------------------------------
   def Reset(self):
        print "In ", self.get_name(), "::Reset()"
        # Add your own code here
#        hardware.reset()
        self.set_state(PyTango.DevState.OFF)
        self.set_state('The ski lift is OFF')
```

Implementing a Command

■ General methods

Name	Input (with self)	return	mandatory
init_device	None	None	Yes
delete_device	None	None	No
always_executed_hook	None	None	No

■ Cmd methods

Name	Input (with self)	return	mandatory
is_<Cmd>_allowed	None	bool	No
<Cmd_name>	Depends on cmd arg type	Depends on cmd arg type	Yes

Command data type (PyTango)

Tango data type	Python type
DEV_VOID	No data
DEV_BOOLEAN	bool
DEV_SHORT	int
DEV_LONG	int
DEV_LONG64	long or int (32/64 bits computer)
DEV_FLOAT	float
DEV_DOUBLE	float
DEV_USHORT	int
DEV_ULONG	int
DEV_ULONG64	long or int (32/64 bits computer)
DEV_STRING	str

Command data type (PyTango)

Tango data type	Python type
DEVVAR_CHARARRAY	sequence<int> or numpy array (numpy.uint8)
DEVVAR_SHORTARRAY	sequence<int>or numpy array (numpy.int16)
DEVVAR_LONGARRAY	sequence<int>or numpy array (numpy.int32)
DEVVAR_LONG64ARRAY	sequence<int>or sequence<long> or numpy array (numpy.int64)
DEVVAR_FLOATARRAY	sequence<float>or numpy array (numpy.float32)
DEVVAR_DOUBLEARRAY	sequence<float>or numpy array (numpy.float64)
DEVVAR_USHORTARRAY	sequence<int>or numpy array (numpy.uint16)
DEVVAR_ULONGARRAY	sequence<int>or numpy array (numpy.uint32)
DEVVAR_ULONG64ARRAY	sequence<int>or sequence<long> or numpy array (numpy.uint64)
DEVVAR_STRINGARRAY	sequence<str>
DEVVAR_LONGSTRINGARARAY	sequence with ((sequence<int> or numpy array (numpy.int32)) + sequence<str>)
DEVVAR_DOUBLESTRINGARRAY	Sequence with ((sequence<float> or numpy array (numpy.float32)) + sequence<str>)

Exercise 2

T A N G

- Code the 4 commands of the MaxLabPS:
  - Cmd On. The PS automatically switches to FAULT after 10 seconds
  - Cmd Off
  - Cmd Reset
  - Cmd SendCmd
    - Print the received command string
    - Return 3 numbers and 2 strings

Back to the init_device method

```
#------------------------------------------------------------------
#     Device initialization
#------------------------------------------------------------------
    def init_device(self):
        print "In ", self.get_name(), "::init_device()"
        self.set_state(PyTango.DevState.OFF)
        self.get_device_properties(self.get_device_class())

        self.set_status('The ski lift is OFF')
        self.hardware_readings = []
```

Reading Attribute(s)

- One method to read hardware
  - **SkiLift.read_attr_hardware(data)**
- If state management is needed, one is_xxx_allowed() method
  - **bool SkiLift.is_Speed_allowed(req_type)**
- One method per attribute
  - **SkiLift.read_Speed(Attribute)**

Reading Attribute(s)

- Reading attribute(s) sequence

SkiLift
(CORBA Obj.)

SkiLift
(Device Impl.)

read_attribute**s**(Speed)

always_executed_hook

read_attr_hardware (Attr1, Attr2

is_Speed_allowed (Attr1)

read_Speed (Attr)
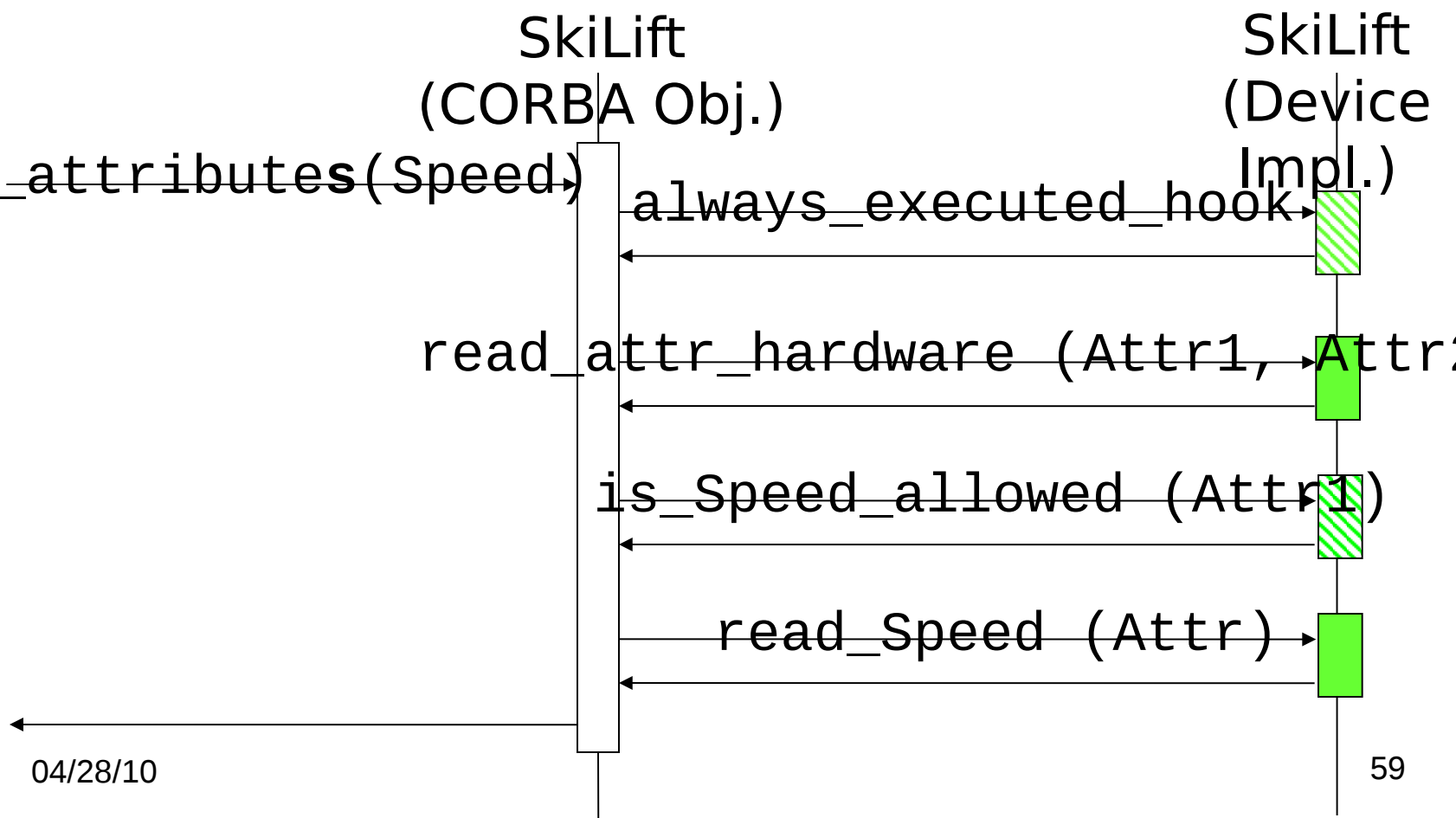
Reading Attribute(s)

■ Most of the attribute Tango feature are implemented in a Tango kernel class called "Attribute". The user only manage attribute data

■ Reading sequence

– read_attr_hardware

• 1 call even if several attributes must be read

• Rule: Reading the hardware only once

• Update internal variable

– is_<attribute>_allowed

• 1 call per attribute

• Rule: Enable/disable attribute reading

Reading Attribute(s)

- **Reading sequence**
  - read_<attribute>
    - 1 call per attribute to read
    - Rule: Affect a value to the attribute
    - Associate the attribute and a variable which represents it with :
      - **attr.set_value(data,…)**

Reading Attribute(s)

■ read_attr_hardware() method

```
#---------------------------------------------------------------
#     Read Attribute Hardware
#---------------------------------------------------------------
    def read_attr_hardware(self,data):
        print "In ", self.get_name(), "::read_attr_hardware()"

        self.hardware_readings = hardware.read()
```

Reading Attribute(s)

■ read_Speed() method

```
#------------------------------------------------------------------
#      Read Speed attribute
#------------------------------------------------------------------
def read_Speed(self, attr):
        print "In ", self.get_name(), "::read_Speed()"

        #      Add your own code here
        attr.set_value(self.hardware_readings[0])
```

Writing Attribute(s)

- If state management is needed, one is_xxx_allowed() method
  - **bool SkiLift.is_Speed_allowed(req_type)**
- One method per attribute
  - **SkiLift.write_Speed(Wattribute)**

Writing Attribute(s)

■ Writing attribute(s) sequence

SkiLift
(CORBA Obj.)

SkiLift
(Device
Impl.)

write_attribute(Speed)

always_executed_hook

is_Speed_allowed (Attr)

write_Speed (Attr)
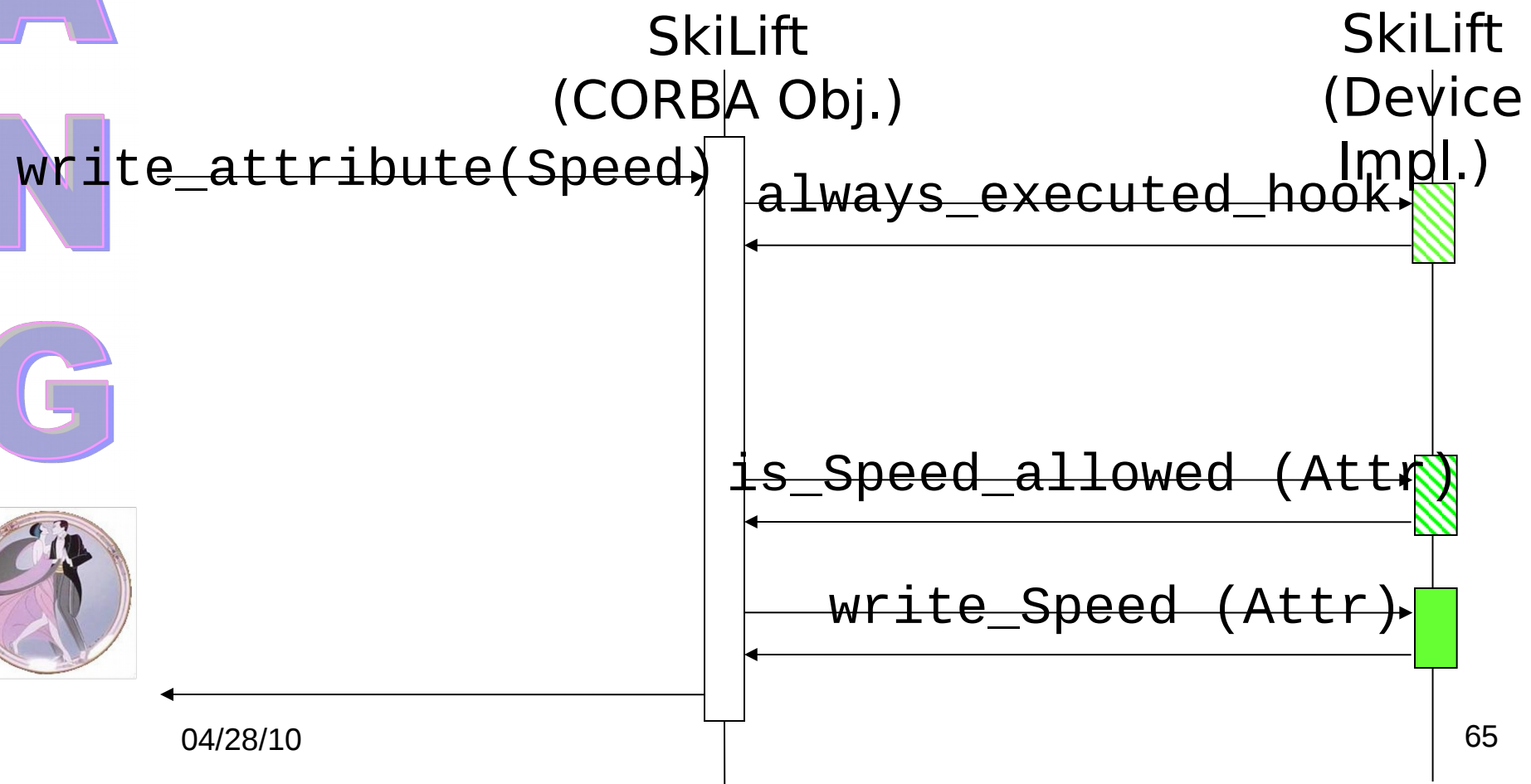
Writing Attribute(s)

- Writing sequence
  - is_<attribute>_allowed
    - 1 call per attribute
    - Rule: Enable/disable attribute writing
  - write_<attribute>
    - 1 call per attribute to write
    - Rule: Get the value to be written and set the hardware
    - Get the value to be written with :
      - **attr.get_write_value()**

Writing Attribute(s)

■ write_Speed() method

```
def write_Speed(self, attr):
        print "In ", self.get_name(), "::write_Speed()"
#       data=[]
#       attr.get_write_value(data)
        data = attr.get_write_value()
        hardware.write_speed(data)
```

Implementing attribute

- ## General methods

Name	Input (with self)	return	mandatory
always_executed_hook	None	None	No
Read_attr_hardware	List<int>	None	No

- ## Attribute methods

Name	Input (with self)	return	mandatory
is_<Attr>_allowed	req_type (int)	bool	No
write_<Attr>	WAttribute	None	Yes
read_<Attr>	Attribute	None	Yes

Scalar Attribute data type (PyTango)

Tango data type	Python type
DEV_BOOLEAN	bool
DEV_UCHAR	int
DEV_SHORT	int
DEV_LONG	int
DEV_LONG64	long or int (32/64 bits computer)
DEV_FLOAT	float
DEV_DOUBLE	float
DEV_USHORT	int
DEV_ULONG	int
DEV_ULONG64	long or int (32/64 bits computer)
DEV_STRING	str

Spectrum/Image data type (PyTango)

Tango data type	Python type
DEV_BOOLEAN	sequence<bool> or numpy.ndarray (numpy.xxx)
DEV_UCHAR	sequence<int> or numpy.ndarray (numpy.uint8)
DEV_SHORT	sequence<int> or numpy.ndarray (numpy.int16)
DEV_LONG	sequence<int> or numpy.ndarray (numpy.int32)
DEV_LONG64	sequence<long or int> or numpy.ndarray (numpy.int64)
DEV_FLOAT	sequence<float> or numpy.ndarray (numpy.float32)
DEV_DOUBLE	sequence<float> or numpy.ndarray (numpy.float64)
DEV_USHORT	sequence<int> or numpy.ndarray (numpy.uint16)
DEV_ULONG	sequence<int> or numpy.ndarray (numpy.uint32)
DEV_ULONG64	sequence<long or int> or numpy.ndarray (numpy.uint64)
DEV_STRING	sequence<str>
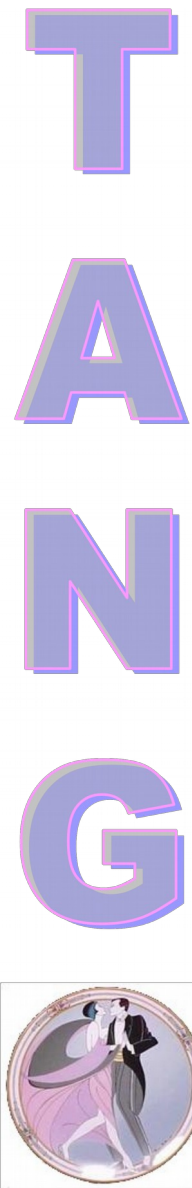
Memorised Attributes

- Only for writable scalar attributes!
- For every modification the attribute set point is saved in the database
- Memorized attributes initialization options (supported by Pogo)
  - Write hardware at init.

Exercise 3 (Arg !!…)

Add attributes to the MaxLabPowerSupply class

- **Voltage** (Double – Scalar – R/W): What you read is what has been written (if state is ON or ALARM, otherwise 0). 0 at init
- **Current** (Double – Scalar – R/W - Mem): What you read is what has been written + random between 0 and 1 (if state is ON or ALARM, otherwise 0). Take 100 mS.
- **CurrentSetPoint** (Double – Scalar - R): The Current attribute set point

Reporting Errors

■ Using exception

– The Tango exception DevFailed is an error stack

– Each element in the stack has 4 members :

- reason (string)
  – The exception summary
- desc (string)
  – The full error description
- origin (string)
  – The method throwing the exception
- Severity (string) (not used)
  – Set to WARN, ERR, PANIC

Reporting Errors

- Static methods to help throwing an exception
- Another method to re-throw an exception and to add one element in the error stack (Often used in a "except" block)

```
PyTango.Except.throw_exception('SkiLift_NoCable',
                        'Oups, the cable has fallen down !!',
                        'SkiLift.init_device()')


PyTango.Except.re_throw_exception(previous_exception,
                        reason, desc, origin)
PyTango.Except.print_exception(except)
```

Properties

- Properties are stored within the MySQL database
- No file – Use Jive to create/update/delete properties
- You can define properties at
  - Object level
  - Class level
  - Device level
  - Attribute level

Properties

- ■ Property data type
  - Simple type
    - bool, short, long, float, double, unsigned short, unsigned long, string
  - Array type
    - short, long, float, double, string
- ■ Pogo generates code to retrieve properties from the database and store them in your device
  - Method MyDev.get_device_property()
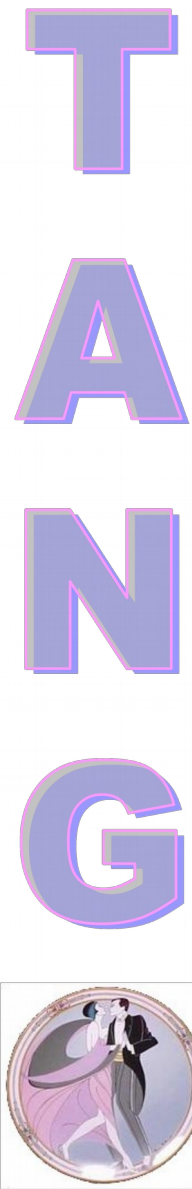
Properties

■ Algorithm generated by Pogo to simulate default property values

- /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/

Properties

■ PyTango creates a class python attribute for each device property

if self.MyProp is True:
Do What You Want

Attribute Properties

- Several ways to define them with a priority schema (from lowest to highest priority) :
  - There is a default value hard-coded within the library
  - You can define them at class level
  - You can define them by code (POGO) at class level
  - If you update them, the new value is taken into account by the device server and written into the database. Device level.

Exercise 4

- The SendCmd command returns exception if input arg != "calibrate"

- The time before the PS switches to Fault is a device property **TimeToFault** (default value 10)

- The Voltage attribute value at startup is a device property **DefaultVoltage** (default value 123)

Some code executed only once ?

- Yes, it is foreseen
- Each Tango class has a MyDevClass class (SkiLiftClass) with only one instance.
- Put code to be executed only once in its constructor
- Put data common to all devices in its data members
- The instance of MyDevClass is constructed before any devices

A Tango Device Server Process

- The main part

```
#================================================================
#
#       SkiLift class main method
#
#================================================================
if __name__ == '__main__':
        try:
                py = PyTango.Util(sys.argv)
                py.add_TgClass(SkiLiftClass,SkiLift,'SkiLift')

                U = PyTango.Util.instance()
                U.server_init()
                U.server_run()

        except PyTango.DevFailed,e:
                print '-------> Received a DevFailed exception:',e
        except Exception,e:
                print '-------> An unforeseen exception occured....',e
```

Automatically added Commands/Attributes

- **Three commands are automatically added**
  - **State** : In = void Out = DevState
    - Return the device state and check for alarms
    - Overwritable
  - **Status** : In = void Out = DevString
    - Return the device status
    - Overwritable
  - **Init** : In = void Out = void
    - Re-initialise the device (delete_device + init_device)
- **Two attributes are automatically added**
  - State and Status

The remaining Network Calls

- ping
  - Just ping a device. Is it available on the network?
- command_list_query
  - Returns the list of device supported commands with their descriptions
- command_query
  - Return the command description for one specific command
- info
  - Return general info on a device (class, server host….)

The remaining Network Calls

- get_attribute_config
  - Return the attribute configuration for x (or all) attributes
- set_attribute_config
  - Set attribute configuration for x attributes
- blackbox
  - Return x entries of the device black box
  - Each device has a black box (round robin buffer) where each network call is registered with its date and the calling host

The remaining Network Calls

- write_read_attribute
  - Write then read one attribute in one go

The remaining Network Calls

For completeness

- Five CORBA attributes
  - state
  - status
  - name
  - description
  - adm_name

Tango Training: Part 4 : The Client Side

- The PyTango client API
- Error management
- Asynchronous call
- Group call

Tango on the Client Side

- A C++, Python and Java API is provided to simplify developer's life
    - Easy connection building between clients and devices
    - Manage re-connection
    - Hide some IDL call details
    - Hide some memory management issues
- These API's are a set of classes

PyTango Client

■ On the client side, each Tango device is an instance of a **DeviceProxy** class

■ DeviceProxy class
  – Hide connection details
  – Hide which IDL release is supported by the device
  – Manage re-connection

■ The DeviceProxy instance is created from the device name

PyTango.DeviceProxy   dev("id13/v-pen/12");

PyTango Client

- The DeviceProxy *command_inout()* method sends a command to a device
- The class DeviceData is used for the data sent/received to/from the command.

DeviceProxy.command_inout(name, cmd_param)

```
dev = PyTango.DeviceProxy("et/s_lift/1")

dev.command_inout('On')
dev.on()

print dev.command_inout('EchoShort',10)

print dev.EchoShort(10)
```
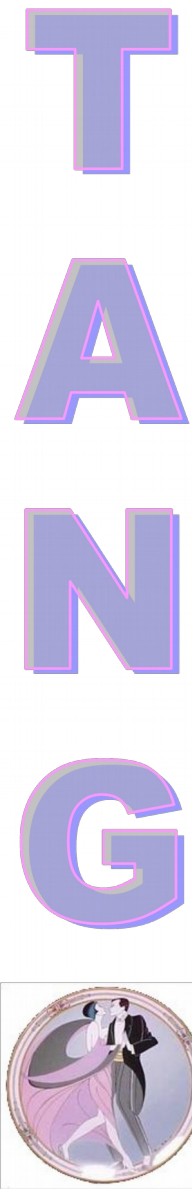
PyTango Client

- The DeviceProxy *read_attribute()* method reads a device attribute (or *read_attributes()*)
- The class DeviceAttribute is used for the data received from the attribute.

```
DeviceAttribute DeviceProxy.read_attribute(name);
```

```
dev = PyTango.DeviceProxy('et/s_lift/1')
da = dev.read_attribute('SpecAttr')
print da.value

print dev['SpecAttr'].value

seq_da = dev.read_attributes(['SpecAttr','ImaAttr'])
```

PyTango Client

- The DeviceProxy *write_attribute()* method writes a device attribute (or *write_attributes()*)

DeviceProxy.write_attribute(name,value)

```
dev = PyTango.DeviceProxy('et/s_lift/1)
dev.write_attribute('SpecAttr',[2,3])

dev.write_attribute('SpecAttr',numpy.array([6,7]))

dev['SpecAttr'] = [3,4]

dev.write_attributes((['Speed',5],['SpecAttr',[2,3]]))
```

PyTango Client

- The API manages re-connection
  - By default, no exception is thrown to the caller when the automatic re-connection takes place
  - Use the *DeviceProxy.set_transparency_reconnection()* method if you want to receive an the exception
- Don't forget to catch the PyTango.DevFailed exception!

PyTango Client

- 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…..
- If you are interested only in attributes, use the **AttributeProxy** class
- Look at PyTango doc (Pink site)

Errors on the Client Side

- All the exception thrown by the API are PyTango.DevFailed exception
- One catch (except) block is enough
- Ten exception classes (inheriting from DevFailed) have been created
  - Allow easier error treatment
- These classes do not add any new information compared to the DevFailed exception

Errors on the Client Side

- Exception classes :
  - ConnectionFailed, CommunicationFailed, WrongNameSyntax, NonDbDevice, WrongData, NonSupportedFeature, AsynCall, AsynReplyNotArrived, EventSystemFailed, NamedDevFailedList

- Documentation tells you (or should) which kind of exception could be thrown.

Errors on the Client Side

- A small example

```
try:
    att = PyTango.AttributeProxy('et/s_lift/1Pres')
    print att.read()
except PyTango.WrongNameSyntax:
    print 'Et couillon, faut 3 / !'
except PyTango.DevFailed,e:
    PyTango.Except.print_exception(e)
```

Exercise 5

- Write a **MultiMaxLabPowerSupply** Tango class
  - 5 states (ON, OFF, FAULT, ALARM, UNKNOWN)
  - 2 commands (On, Off)
  - 1 attribute (Currents: Spectrum – DEV_DOUBLE – R/W)
  - 1 Device property (ChannelsName: string array – default = "Not defined")
- This Tango class is a client of the individual power supply device (channel)

Exercise 5

- Refuse to start if no channel name defined
- State management:
  - If one channel in FAULT -> FAULT
  - Idem for OFF and ALARM, otherwise ON
  - UNKNOWN in case of exception
- On Allowed only when OFF/ON
  - Switches ON all channels
- Off Allowed only when ON/OFF/ALARM
  - Switches OFF all channels
- Currents attribute
  - Return individual channels value (as a Numpy array)
  - Write individual channels. Exception if wrong inputs number
- Create 3 MaxLabPowerSupply devices and connect them to a single MultiMaxLabPowerSupply device.

Asynchronous Call

- Asynchronous call :
  - The client sends a request to a device and does not block waiting for the answer.
  - The device informs the client process that the request has ended
- Does not request any changes on the server side
- Supported for
  - command_inout
  - read_attribute(s)
  - write_attribute(s)

Asynchronous call

■ Tango supports two models for clients to get requested answers

– The **polling** model

• The client decides when it checks for requested answers

– With a non blocking call

– With a blocking call

– The **callback** model

• The request reply triggers a callback method

– When the client requested it with a synchronization method (Pull model)

– As soon as the reply arrives in a dedicated thread (Push model)

Group Call

- Provides a single point of control for a Group of devices
- Group calls are executed asynchronously!
- You create a group of device(s) with the PyTango.Group class
  - It's a hierarchical object (You can have a group in a group) with a forward or not forward feature
- You execute a command (or R/W attribute) on the group

Group Call

- Using groups, you can
  - Execute one command
    - Without argument
    - With the same input argument to all group members
    - With different input arguments for group members
  - Read one attribute
  - Write one attribute
    - With same input value for all group members
    - With different input value for group members
  - Read several attributes

Group Call

- Three classes to get group action result
  - PyTango.GroupCmdReplyList
    - For command executed on a group
  - PyTango.GroupAttrReplyList
    - For attribute(s) read on a group
  - PyTango.GroupReplyList
    - For attribute written on a group

Tango Training: Part 5 : More info on Device Servers

- The Administration Device
- The Logging System
- The Polling

The Administration Device

- Every device server has an administration device
- Device name
  - dserver/<exec name>/<instance name>
- This device supports 27 (30) 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
  - 4 commands to lock/unlock device

The administration device

- Miscallaneous commands
  - DevRestart destroy and re-create a device. The client has to re-connect to the device
  - RestartServer to restart a complete device server
  - QueryClass to get the list of Tango classes embedded within the process
  - QueryDevice to get the list of available devices
  - Kill to kill the process
  - State, Status, Init

04/28/10

The Tango Logging System

■ Send device server messages to a target
- A file
- The console
- A centralized application called **LogViewer**
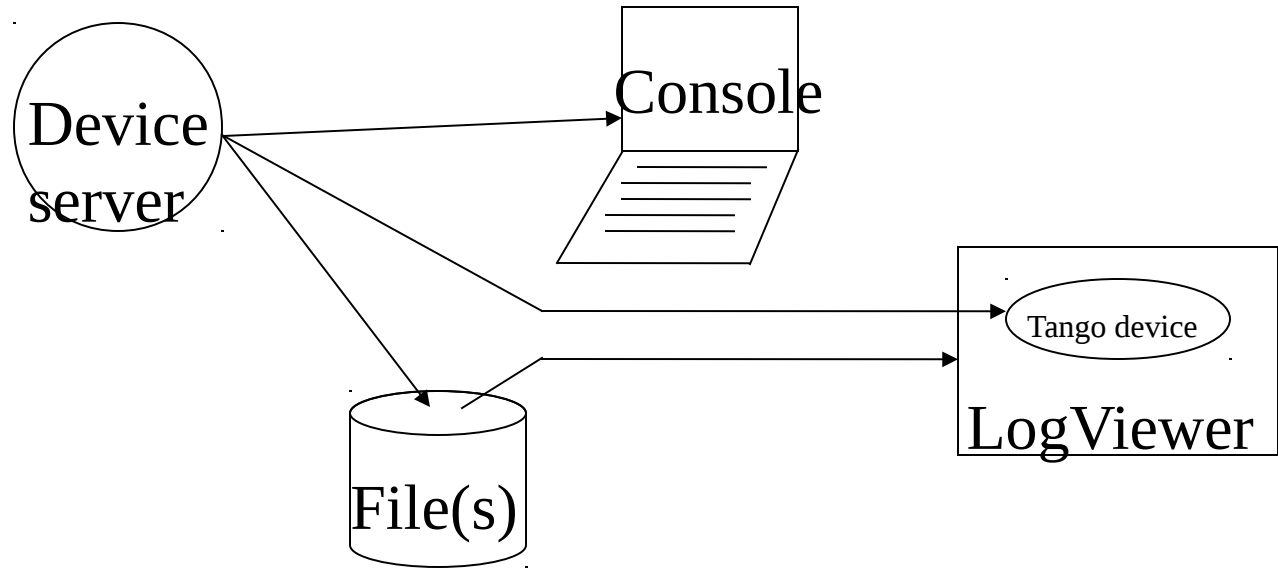
The Tango Logging System

- Each Tango device has a logging level
- Each logging request also has a logging level
- Six ordered logging levels are defined
  - DEBUG < INFO < WARN < ERROR < FATAL < OFF
- Each logging request with a level lower than the device logging level is ignored
- Device default logging level is WARN
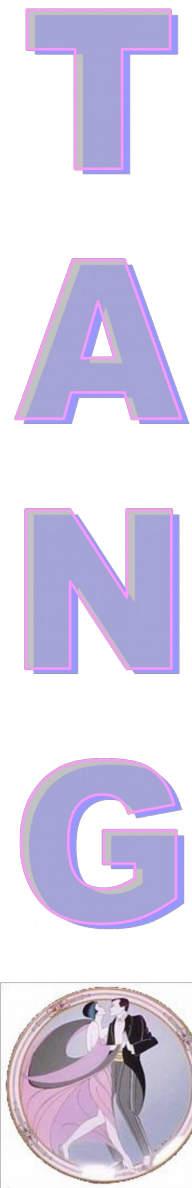
The Tango Logging System

■ Five functions to send logging messages

– print like

• self.{fatal, error, warn, info, debug}_stream()

■ Usage :

```
self.debug_stream("Hola amigo, que tal ?")

self.debug_stream('In read_Speed method for device',self.get_name())
```

The Tango Logging System

- **Logging on a console**
  - Send messages to the console on which the device server has been started

- **Logging in a file**
  - Logging message stored in a XML file
  - Manage 2 files
  - Swap files when file size is greater than a pre-defined value (a property). Rename the old one as "xxx_1". Default file size threshold is 2 MBytes
  - Default file names: "/tmp/tango/process/instance/device.log" or "C:\tango\….." (create directory by hand…)
  - Read files with the "LogViewer" application

The Tango Logging System

- Logging with the LogViewer
  - Send messages to a Tango device embedded in the LogViewer application
- LogViewer (Java appl.)
  - Graphical application to display, filter and sort logging messages
  - Two modes
    - Static: Memorize a list of Tango devices for which it will get/display messages
    - Dynamic: The user (with a GUI) chooses devices for which messages must be displayed

The Tango Logging System

■ Seven administration device commands dedicated to logging
  – AddLoggingTarget
  – RemoveLoggingTarget
  – GetLoggingTarget
  – GetLoggingLevel
  – SetLoggingLevel
  – StopLogging
  – StartLogging

The Tango Logging System

■ Logging configuration with Jive
- current_logging_level
  - Not memorized
- logging_level
  - Memorized in db
- current_Logging_target
  - Not memorized
  - console::cout, file::/tmp/toto or device::tmp/log/xxx
- logging_target
  - Memorized in db

The Tango Logging System

■ Each device server has a "-v" option
  – v1 and v2
    • Level = INFO and target = console::cout for all DS devices
  – v3 and v4
    • Level = DEBUG and target = console::cout for all DS devices
  – v5
    • Like v4 plus library messages (there are many) on target = console::cout
  – Without level is a synonym for –v4
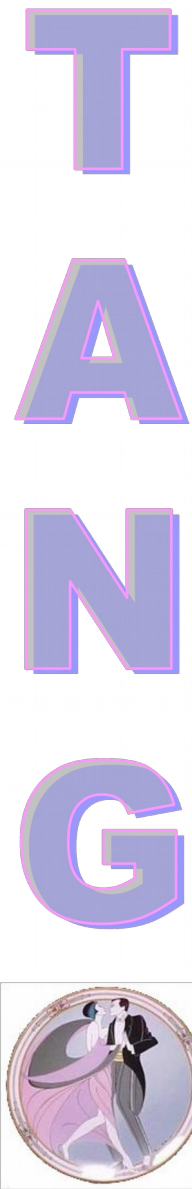
The Polling

- Each Tango device server has a polling thread pool
- It's possible to poll attributes and/or commands (without input parameters)
- The polling result is stored in a polling buffer (round robin buffer)
- Each device has its own polling buffer
- Polling buffer depth is tunable
  - By device (default is 10)
  - By command/attribute

The Polling

- By default, there is only one polling thread in the pool
- You assign polled device to a thread
- Two admin device properties to manage polling thread pool
  - polling_thread_pool_size
  - polling_thread_pool_conf
- The Tango admin tool (astor) has a graphical panel to tune device server polling
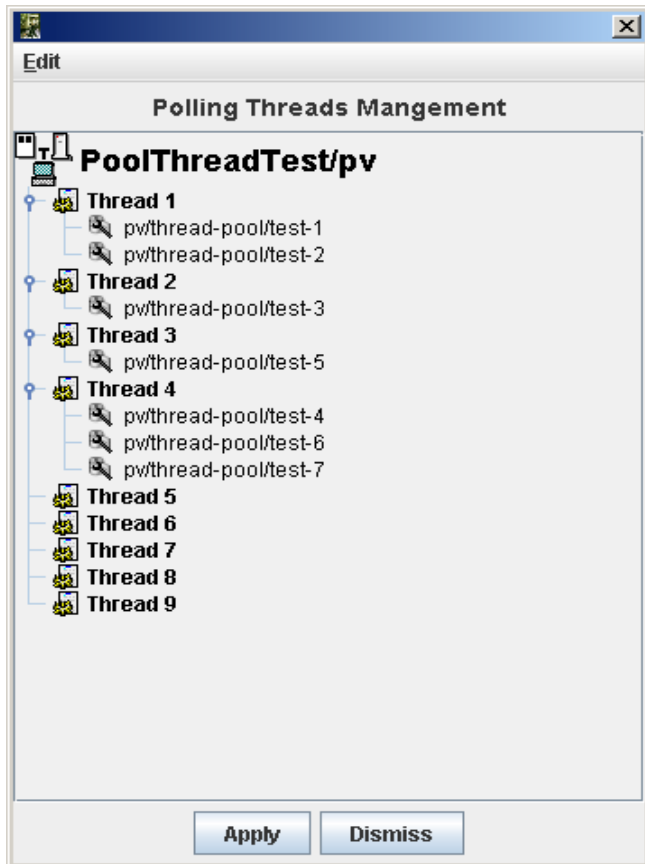
The Polling

The Polling

- A client is able to read data from
  - The real device
  - The last record in the polling buffer
  - The polling buffer and in case of error from the real device
  - The choice is done with the *DeviceProxy.set_source()* method
- A network call to read the complete polling buffer is also provided (command_inout_history or read_attribute_history defined in the Tango IDL)
  - *Not wrapped to Python…*

The Polling

■ Seven administration device commands allow the polling configuration
- AddObjPolling
- RemObjPolling
- UpdObjPolling
- StartPolling
- StopPolling
- PolledDevice
- DevPollStatus

The Polling

- How it starts ?
  - At device startup
- For completeness
  - Externally triggering mode (C++ DS only)
  - External polling buffer filling (C++ DS only)
    - Get data with the command_inout_history or read_attribute_history calls

The Polling

- The polling has to be tuned
  - Do not try to poll a command with a polling period of 200 mS if the command needs 250 mS !!!
  - If a polling thread is late (for one reason or another), it discards polling
  - Leave your device available for around 50 % for external world requests
    - For a command needing 250 mS, minimum polling period around 500 mS

Exercise 6

- Poll the Current attribute of one MaxLabPowerSupply device
  - Play with the source parameter
- Add some Tango logging messages in the MaxLabPowerSupply Tango class
  - Start device server process using –vx option
  - Start the LogViewer appli

Tango Training:
Part 6 :
Events

Events

■ Another way to write applications

  – Applications do not poll any more

  – The device server informs the applications that "something" has happened

■ Polling done by the device server polling thread(s)

■ Uses a CORBA service called "Notification Service"

■ Tango uses omniNotify as Notification Service

Events

- One Notification service daemon (notifd) running on each host

- Event propagation

  - The event is sent to the notification service

    - When detected by the polling thread(s)

    - On request (push_event() call family)

  - The notification service sends the event to all the registered client(s)

- It is possible to ask the notification service to filter events

Events

Dev Server

Dev Server

Notification service (simplified)

Event table in the Tango database

Event Channel factory

Event channel

Event channel

Filter 1   Filter 2

Filter 3 per   Filter 4 change

Filter factory

Name	IOR

Client 1 per

Client 2 per

Client 3 per/change

Client 4 change

Client 5 change

04/28/10

128
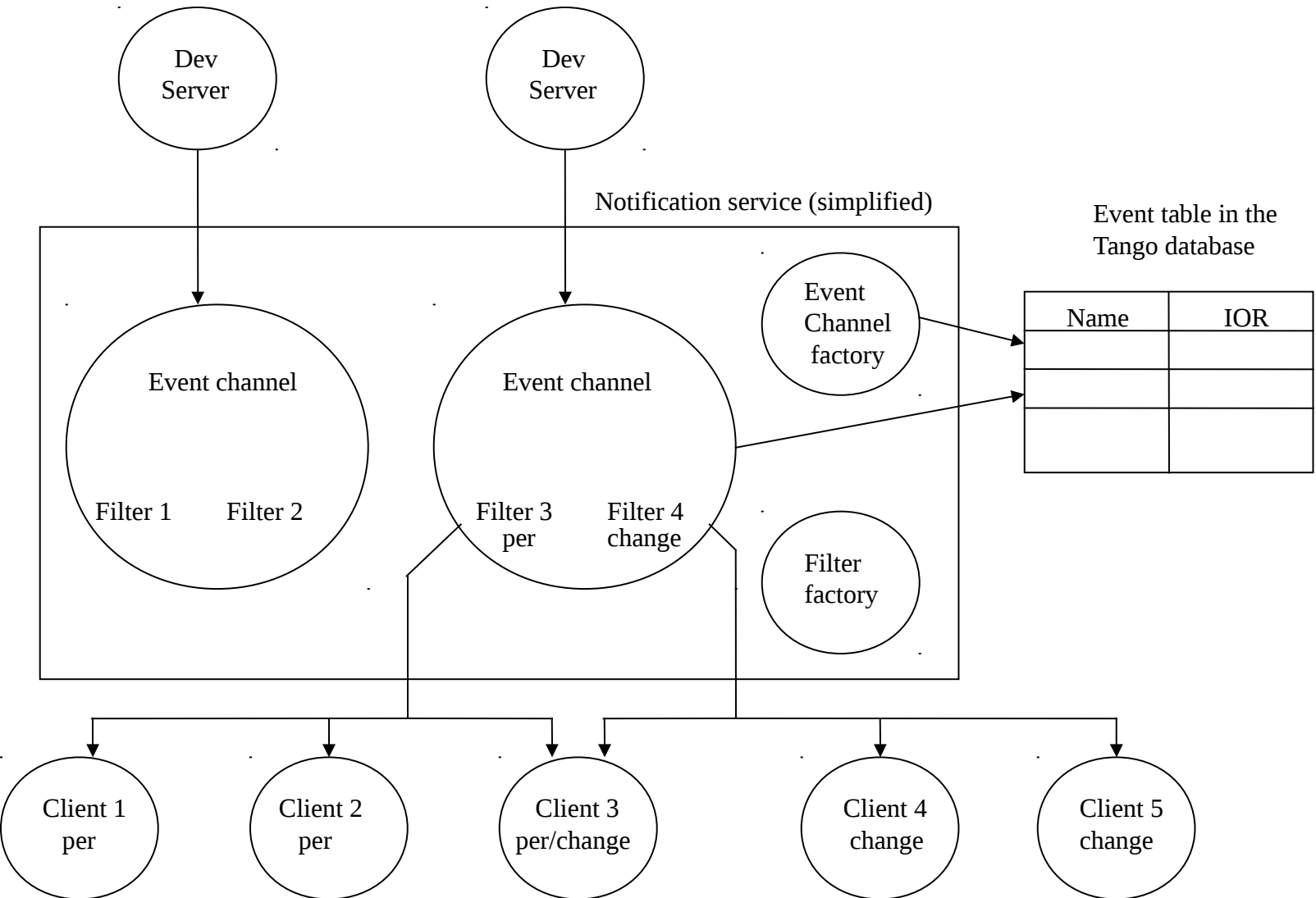
Events

■ Only available on attributes!

■ Does not requires any changes in the device server code

■ Based on callbacks. The client callback is executed when an event is received

- Event data or an error stack in case of an exception

■ 6 types of events

- Periodic, Change, Archive

- Attribute configuration change, Data ready

- User defined

Events

- **Periodic event**
  - Event pushed:
    - At event subscription
    - On a periodic basis
- **Change event**
  - Event pushed when
    - a change is detected in attribute data
    - a change is detected in attribute size (spectrum/image)
    - At event subscription
    - An exception was received by the polling thread
    - the attribute quality factor changes
    - When the exception disappears

04/28/10
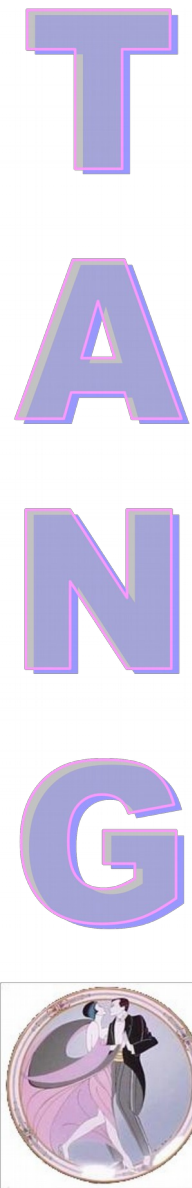
Events

- **Archive event**
  - A mix of periodic and change events
- **Attribute configuration change**
  - Event pushed when:
    - At event subscription
    - The attribute configuration is modified with set_attribute_config()
- **User defined event / Data ready event**
  - Event pushed when the user decides it
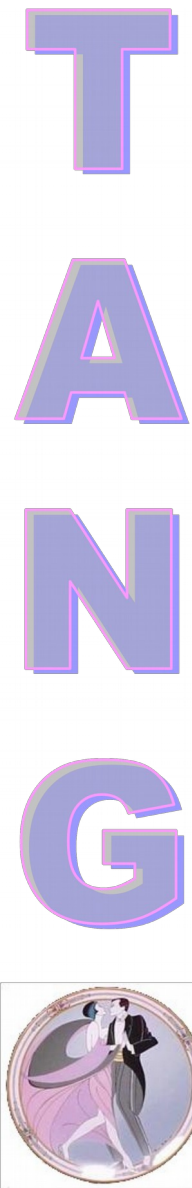
Events (configuration)

- **Periodic event configuration**
  - event_period (in mS).
    - Default is 1000 mS
    - Cannot be faster than the polling period
  - Polling period != event period
  - The event system does not change the attribute polling period if already defined

Polling
400 mS

Event
(1000)

Client
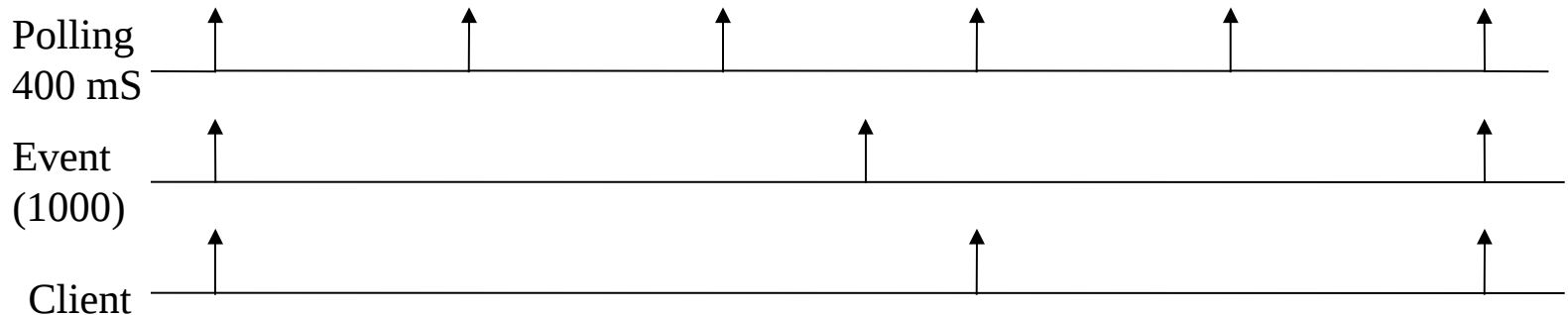
Events (configuration)

■ Change event configuration

- Checked at the polling period

- rel_change and abs_change

  • Up to 2 values (positive, negative delta)

  • If both are set, relative change is checked first

  • If none is set -> **no change event!**

Events (configuration)

■ Archive event configuration

– Checked at the polling period

– event_period (in mS).

 • Default is 0 mS -> **no periodic archive event**!

– rel_change and abs_change

 • Up to 2 values (positive, negative delta)

 • If both are set, relative change is checked first

 • If none is set -> **no archive event on change!**

Events (configuration)

- Event configuration parameters (event_period, abs_change, rel_change…) are part of the attribute configuration properties

- Can be configured with Jive

Events (pushed from the code)

- Possible for change, archive, user and data ready events

- To push events manually from the code a set of data type dependent methods can be used:

DeviceImpl.push_xxx_event (attr_name, ...)

xxx = {change, archive, data_ready, 'nothing'}

- It is possible to push events from the code and from the polling thread at the same time

- Attribute configuration with Pogo

Events (pushed from the code)

- To allow a client to subscribe to events of non polled attributes the server has to declare that events are pushed from the code

DeviceImpl.set_change_event(attr_name, implemented, detect = true)

DeviceImpl.set_archive_event(attr_name,implemented, detect = true)

  - *implemented*=true inidcates that events are pushed manually from the code
  - *detect*=true triggers the verification of the same event properties as for events send by the polling thread.
  - *detect*=false, no value checking is done on the pushed value!

Events (filtering)

- When you subscribe to an event, you may ask for a filters
- **All filters are compared to the last event value send and not to the actual attribute value!**
- Periodic event filter
  - Filterable data name : "counter"
  - Incremented each time the event is sent
  - Ex : "$counter % 2 == 0"

Events (filtering)

■ Change event filters are

- "quality" is true when the event was pushed on a quality change
  - "Ex: $quality == 1
- "forced_event" is true when the event was pushed due to an exception, an exception change or when the exception disappears
- "delta_change_rel" and "delta_change_abs" contain the change detected by server compared to the last event pushed
  - Ex : "$delta_change_abs >= 2"

Events (filtering)

■ Archive event filters are

– "counter" as for the periodic event

– "quality" and "forced_event" as for the change event

– "delta_change_rel" and "delta_change_abs" as for the change event

– "delta_event" contains the delta time in ms since the last archive event was pushed

- Ex: "$delta_event >= 2000"

04/28/10

140

Events (heartbeat)

- ## To check that the device server is alive
    - A specific "heartbeat event" is sent every 10 seconds to all clients connected 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

Events (heartbeat)

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

Events (threading)

- On the client side
  - As soon as you create a DeviceProxy -> 2 threads (main thread + omniORB scavenger thread)
  - First event subscription adds 3 threads:
    - (orb thread, omniORB thread and KeepAliveThread)
  - Clients are servers : One more thread per Notification service sending events to the client
  - thread number: 5 + n   (n = Notif service connected (+1 for linux))
  - Warning : Callbacks are not executed by the main thread !
- On the server side
  - No changes

04/28/10

143
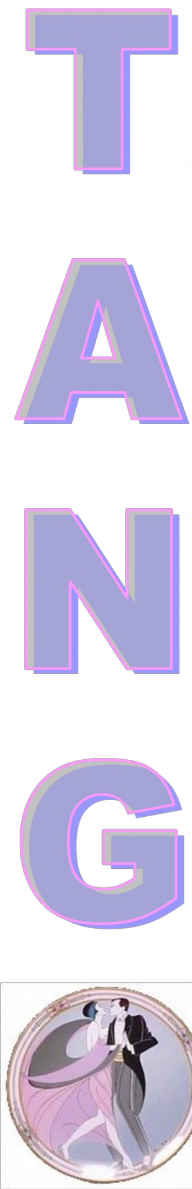
Events (client side)

- Event subscription with the *DeviceProxy.subscribe_event()* method

- Event un-subscription with the DeviceProxy.*unsubscribe_event()* method

- Call-back (idem to asynchronous call)
  - Method *push_event()* to overwrite in your class
  - This method receives a pointer to an instance of a PyTango.EventData class

Events (client side)

```python
import PyTango
import time

class MyCb:
    def push_event(self,ev_data):
        if ev_data.err is True:
            print "Error received in event callback"
        else:
            if  len(ev_data.attr_value.get_err_stack() ) == 0:
                print ev_data.attr_value.value

if __name__ == '__main__'
    cb = MyCb()

    dev = PyTango.DeviceProxy('et/s_lift/1')
    ev_id = dev.subscribe_event('Speed',PyTango.EventType.CHANGE_EVENT,cb,[])

    time.sleep(30)

    dev.unsubscribe_event(ev_id)
```

```
class EventData:
    device (DeviceProxy)
    attr_name (string)
    event (string)
    attr_value (DeviceAttribute)
    err (bool)
    errors (sequence<DevError>)
```

04/28/10

145
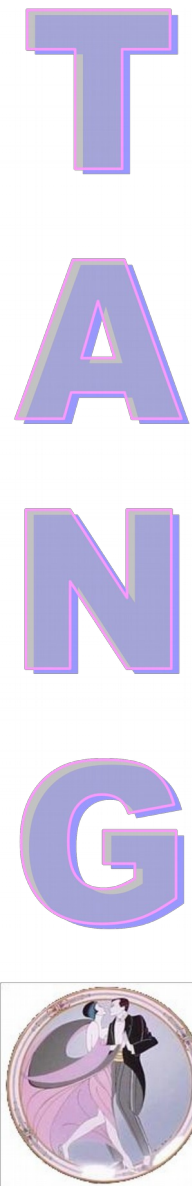
Events (client side)

- The event subscription can be stateless (in case the device server process does not run)
- You can also manage an event queue to decuple the application from the events
  - Defined at event subscription time
    - Queue size defined in the DeviceProxy.subscribe_event() call
  - The user calls DeviceProxy.get_events() to get the events from the queue

04/28/10

Exercise 7

■ Test set up

   – Add a command which increments by 2 the Current attribute (IncrCurrent – void –void)

■ Start the notification service and register the service to the Tango database

   – notifd –n

   – notifd2db

■ Write a client which subscribes to a change event and sleeps waiting for events

Tango Training: Part 7 : Device Server Level 2…

- C++ specific features
- Attribute Alarms
- Several classes in the same device server
- Threading model
- Abstract classes
- Device servers on Windows

C++ : Creating the Device

- A init_device() method to construct the device
  - **void SkiLift::init_device()**
- A delete_device() to destroy the device
  - **void SkiLift::delete_device()**
- All memory allocated in init_device() must be deleted in delete_device()

C++ : Command Memory Management

■ For string dynamically allocated (Pogo style)

– Memory allocated in the command code and freed by the Tango layer

```
Tango::DevString MyDev::dev_string(Tango::DevString argin)
{
    Tango::DevString argout;

    cout << "The received string is " << argin << endl;

    string str("Am I a good Tango dancer?");
    argout = new char[str.size() + 1];
    strcpy(argout,str.c_str());

    return argout;
}
```
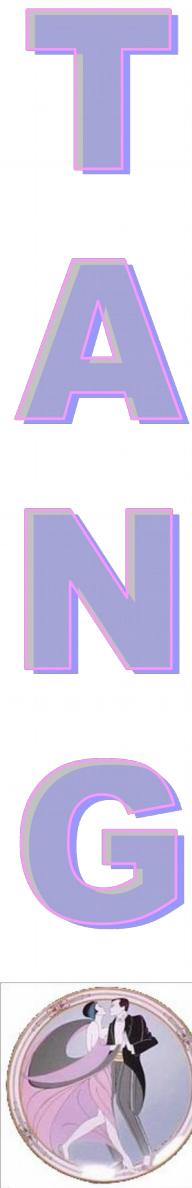
C++ : Command Memory Management

■ For string statically allocated

– ConstDevString is not a new type, just to allow type overloading

– Pogo gives you the choice (for free !)

```
Tango::ConstDevString MyDev::dev_string(Tango::DevString argin)
{
    Tango::ConstDevString argout;

    cout << "The received string is " << argin << endl;
    argout = "Hola todos";

    return argout;
}
```
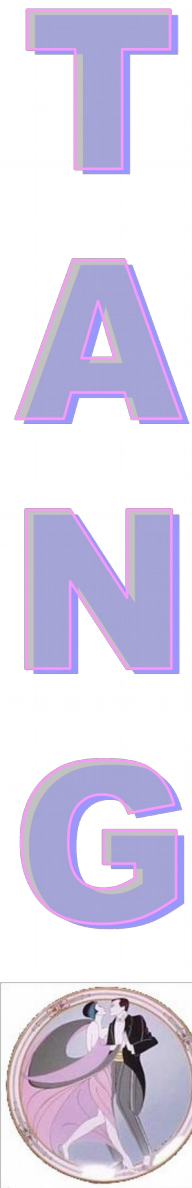
C++ : Command Memory Management

■ **For array dynamically allocated (Pogo)**

  – Memory freed by Tango (how lucky are the users!)

```
Tango::DevVarLongArray  *MyDev::dev_array()
{
    Tango::DevVarLongArray   *argout = new Tango::DevVarLongArray();

    output_array_length = …..;
    argout->length(output_array_length);
    for (unsigned int i = 0;i < output_array_length;i++)
        (*argout)[i] = i;

    return argout;
}
```

C++ : Command Memory Management

■ For array statically allocated

   – Tango provides a simple function to build Tango array types from a pointer (create_xxxx)

```
Tango::DevVarLongArray  *MyDev::dev_array()
{
    Tango::DevVarLongArray  *argout;

    long argout_array_length = ….;
    argout = create_DevVarLongArray(buffer, argout_array_length);
    return argout;
}
```

C++ : Command Memory Management

■ For string array dynamically allocated

– Again memory will be freed by Tango layer

```
Tango::DevVarStringArray *MyDev::dev_str_array()
{
    Tango::DevVarStringArray *argout = new Tango::DevVarStringArray();

    argout->length(3);
    (*argout)[0] = CORBA::string_dup("Rumba");
    (*argout)[1] = CORBA::string_dup("Waltz");
    string str("Jerck");
    (*argout)[2] = Tango::string_dup(str.c_str());

    return argout;
}
```

C++ Attribute Memory Management

■ Designed to reduce data copy
– Uses a pointer to a memory area which by default is not freed

```
void MyDev::read_LongSpecAttr(Tango::Attribute &attr)
{
    …..
    attr.set_value(buffer);
}
```

But it is possible to ask Tango to free the allocated memory

```
void MyDev::read_LongSpecAttr(Tango::Attribute &attr)
{
    long length = …..
    long *buffer = new long[length];

    attr.set_value(buffer,length,0,true);
}
```
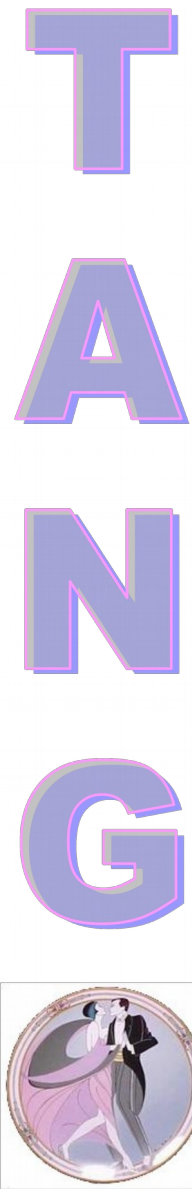
C++ : **Attribute Memory Management**

■ What about a string spectrum attribute ?

```
Class MyDev:…..
{
….
DevString attr_str_array[2];
};
```

```
void MyDev::read_StringSpecNoRelease(Tango::Attribute &attr)
{
    attr_str_array[0] = "Donde esta";
    attr_str_array[1] = "la cerveza?";

    attr.set_value(attr_str_array,2);
}


void MyDev::read_StringSpecRelease(Tango::Attribute &attr)
{
    Tango::DevString *str_array = new Tango::DevString [2];

    str_array[0] = Tango::string_dup("La cerveza");
    str_array[1] = Tango::string_dup("esta en la nevera");

    attr.set_value(str_array,2,0,true);
}
```

OS signals in a Device Server

- It is UNSAFE to do what you want in a signal handler

- Device servers provide a dedicated thread for signal handling
  - You can code what you want in a Tango device signal handler

- Use the *register_signal()* and *unregister_signal()* methods to register/unregister signal handlers

OS signals in a Device Server

- Code your handler in the *signal_handler()* method
- You can install a signal_handler on a device basis if you filter the registering/un-registering methods
- It is also possible to install a signal handler at class level

Attribute Alarms

- Two types of alarms
  - On value
  - On read different than set
- Alarm on value
  - Two thresholds called ALARM and WARNING
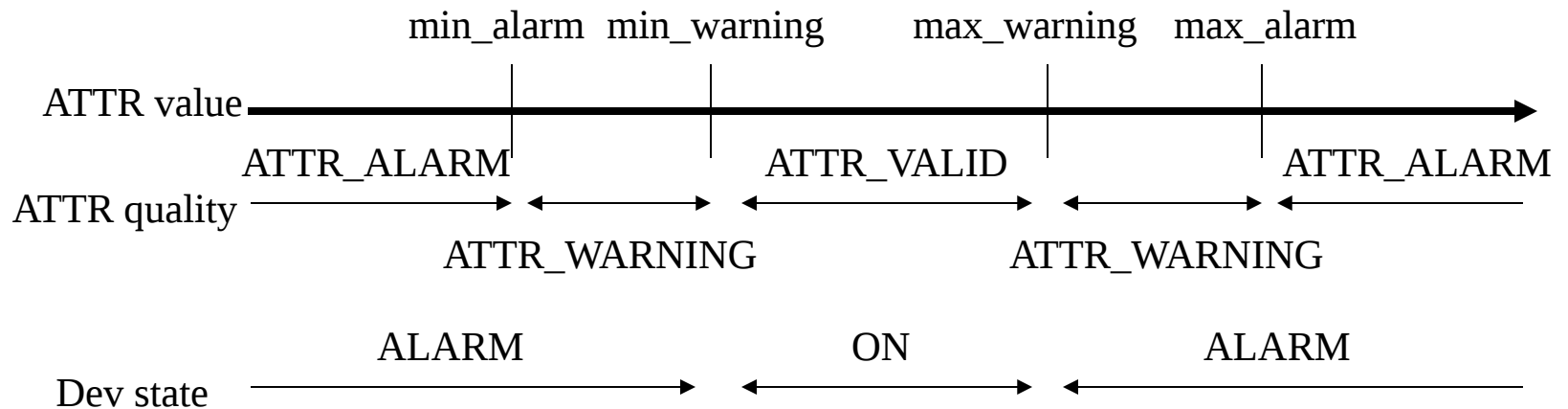
Attribute Alarms

■ Read value different from set value

– Two parameters to tune this alarm

• The authorized delta on value

• The delta time between the last attribute setting and the attribute value check

– Obviously, only on Read-Write attributes and not available for string and boolean

Attribute Alarms

- Six parameters to tune the alarm part of the attribute configuration
  - min_alarm, min_warning, max_warning, max_alarm
  - delta_t, delta_val
- Attribute alarms are cheked during the State command (attribute) execution

Multi Classes Device Server

- Define which Tango classes are embedded in your server
  - C++ : in the class_factory file
  - Python : in the script 'main' part
- To communicate between classes, use the DeviceProxy instance
- All devices of all classes are "exported"
- Classes are created in the defined order and destroyed in the reverse order

Multi Classes Device Server

■ C++ example of a multi classes device server

```
#include <tango.h>
#include <SerialClass.h>
#include <ParagonClass.h>
#include <PLCmodbusClass.h>
#include <IRMirrorClass.h>

void Tango::DServer::class_factory()
{
    add_class(Serial_ns::SerialClass::init("Serial"));
    add_class(Paragon_ns::ParagonClass::init("Paragon"));
    add_class(PLCmodbus::PLCmodbusClass::init("PLCmodbus"));
    add_class(IRMirror_ns::IRMirrorClass::init("IRMirror"));
}
```

Multi Classes Device Server

■ Python example of multi classes device server

```
import PyTango
import CableCar
import SkiResort

if __name__  == '__main__':
    py = PyTango.Util(sys.argv)
    py.add_TgClass(SkiLiftClass, Skilift, 'SkiLift')
    py.add_TgClass(CableCar.CableCarClass, CableCar.CableCar, 'CableCar')
    py.add_TgClass(SkiRessort.SkiResortClass, SkiRessort.SkiResort, 'SkiResort')
```

Multi Classes Device Server

- C++ server build:
  - The classes need to linked together
  - For C++, Pogo generates a Makefile with the options
    - make lib : to add the class to the static class library libtgclasses.a
    - make shlib : to create a shared libray per class. For a class called MyClass the shared library will have the name MyClass.so

Multi Classes Device Server

T
A
N
G

- Python server build:
  - It is possible to mix C++ and Python classes within the same python device server
  - The C++ class has to be compiled as shared library
  - The shared library has to be in the LD_LIBRARY_PATH environment variable
  - Use the add_Cpp_TgClass() method

Multi Classes Device Server

- C++ class in Python server:

```
import PyTango
import CableCar
import SkiResort

if __name__  == '__main__':
    py = PyTango.Util(sys.argv)
    py.add_Cpp_TgClass('Modbus','Modbus')

    py.add_TgClass(SkiLiftClass,Skilift,'SkiLift')
    py.add_TgClass(CableCarClass,CableCar,'CableCar')
    py.add_TgClass(SkiResortClass,SkiResort,'SkiResort')
```

Exercise 8

- Join the classes MaxLabPowerSupply and MultiMaxLabPowerSupply in one device server process

The Threading Model

- omniORB is a multi-threaded ORB
  - A Tango device server also…
- One thread is created in a device server for each client
- A scavenger thread destroys thread(s) associated to unused connections (omniORB feature)
- Not always adapted to hardware access
- Tango also has its own polling and event threads

The Threading Model

■ Each Tango device has a monitor to serialize the device access.

■ Four modes of serialization
  – By device (the default)
  – By class (one monitor for a Tango class)
    • Access to all devices of a class is serialized
    • Use this model if your Tango device needs to access a non threadsafe library
  – By process (one monitor for the whole Tango device server)
  – No serialization (**extreme care**)

The Threading Model

- C++ :
- The *Util::set_serial_mode()* method is used to set the serialization model in the main function

```
int main(int argc, char *argv[])
{
    try
    {
        Tango::Util *tg = Tango::Util::init(argc,argv);

        tg->set_serial_model(Tango::BY_CLASS);

        tg->server_init();
        …..
```

The Threading Model

- Python :
  - The *Util.set_serial_mode()* method is used to set the serialization model in the main part

```
If __name__ == '__main__':
   try:
       py = PyTango.Util(sys.argv)
       py.add_TgClass(SkiliftClass,SkiLift,'SkiLift')

       U = PyTango.Util.instance()
       U.set_serial_model(PyTango.SerialModel.BY_CLASS)
       U.server_init()
       ……
```

Abstract Classes

- Based on the C++ abstract classes (or Java interfaces)

- A way to standardize interfaces
  - What is the minimum number of commands/attributes that my kind of device should provide
  - Write an abstract class which defines only this minimum (no code) with Pogo
  - Write the concrete class which inherits from the abstract class

Abstract Classes

- This allows to have a minimum common interface/behavior for the same type of device
- If possible, an application uses only the minimum interface defined in the abstract class and is independent of the real hardware
- Pogo also supports writing of the abstract class itself.

Abstract Classes

Abstract Classes

■ The next major version of Pogo will allow real inheritance of Tango classes

– Base classes are not only interface classes

– Base classes can be easily extended

■ C++ version in beta test

■ Python not yet started

Abstract Classes
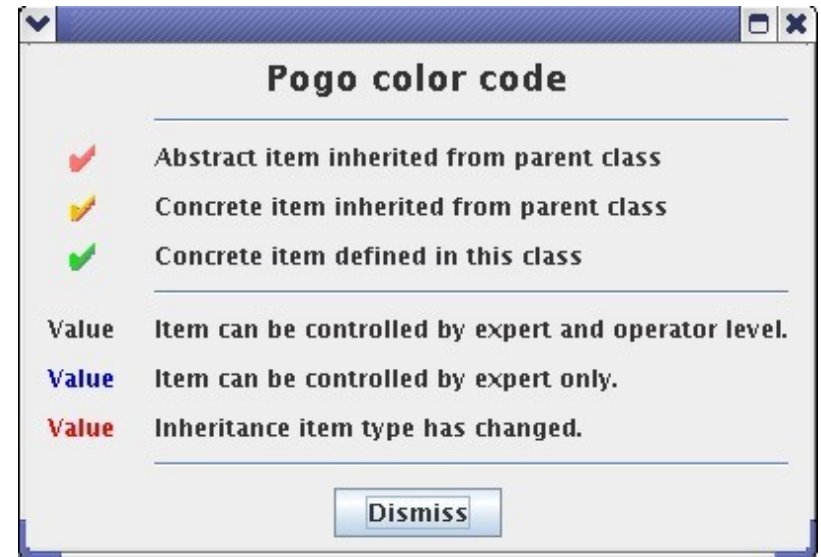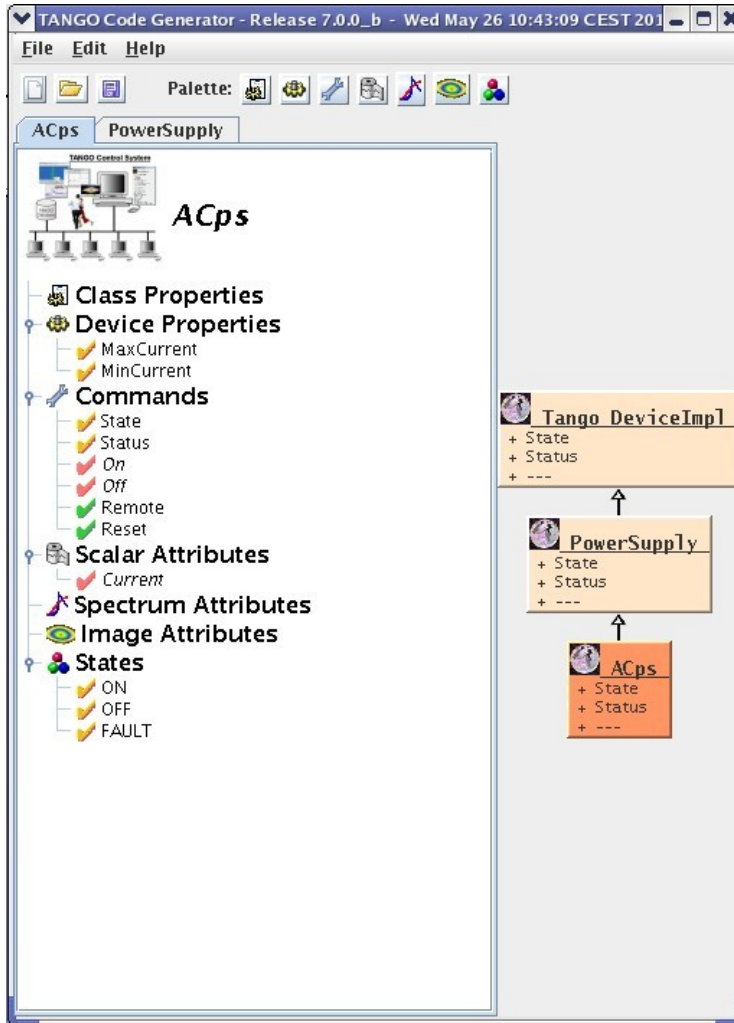
Device Server on Windows

■ Two kinds of Tango device servers on Windows

  – Running as a Windows console application

    • No changes

  – Running as a Windows application

    • Written using MFC

    • Written using Win32 API

DS on Windows

TANGO device server

Device server : tst_mfc/et
TANGO release : Release_5_1
TANGO IDL definition release : 3
Server release : x.y

European Synchrotron Radiation Facility (ESRF)

[OK]

Tango Device Server

European Synchrotron Radiation Facility (ESRF)
CORBA based device server
Developped by Tango team

tst_mfc/et - Console

```
1117783317 [2224] DEBUG dserver/tst_mfc/et In add_obj_polling method
1117783317 [2224] DEBUG dserver/tst_mfc/et Input string = et/mfc/1
1117783317 [2224] DEBUG dserver/tst_mfc/et Input string = command
1117783317 [2224] DEBUG dserver/tst_mfc/et Input string = readvalue
1117783317 [2224] DEBUG dserver/tst_mfc/et Input long = 2000
1117783317 [2440] DEBUG dserver/tst_mfc/et Received an exit command
1117783317 [2224] DEBUG dserver/tst_mfc/et Entering Util::unregister_server method
1117783317 [2224] DEBUG dserver/tst_mfc/et Leaving Util::unregister_server method
1117783317 [2224] DEBUG dserver/tst_mfc/et Entering DeviceClass destructor for class Tst_m
1117783317 [2224] DEBUG dserver/tst_mfc/et Entering DeviceImpl destructor for device et/mfc,
1117783317 [2224] DEBUG dserver/tst_mfc/et Leaving DeviceImpl destructor for device et/mfc/
1117783317 [2224] DEBUG dserver/tst_mfc/et Leaving DeviceClass destructor for class Tst_m
1117783317 [2224] DEBUG dserver/tst_mfc/et Going to shutdown ORB
1117783317 [2224] DEBUG dserver/tst_mfc/et ORB shutdown
```

02/09/17

179

Device server on Windows

■ With the Win32 API

- – Very similar to a traditional "main" but

  - • Replace main by WinMain
  - • Display message box for errors occurring during the device server start-up phase
  - • Code the Windows message loop

- – See example in doc chapter 8.5.3

■ With MFC, see chapter 8.5.2

■ Don't forget to link your device server with the Tango windows resource file

Device Server on Windows

- Take extreme care with the kind of libraries used for linking (No mix)
- Tango supports
  - Multithreaded (/MT)
  - Debug Multithreaded (/MTd)
  - Multithreaded DLL (/MD)
  - Debug Multithreaded DLL (/MDd)

Device Server on Windows

■ A Tango device server is able to run as a **Windows service** but

– Needs changes in the code (See doc chapter 8.5.4)

– Needs to be registered in the Windows service manager

 • A new set of options is available when a device server is used as a Windows service

 – -i, -u or -s

Tango Training:
Part 8 : Advanced Features

- Tango without database
- Multi CS / Multi DB
- Tango adminstration
- Server Wizard

DS using a File as Database

- Tango device server supports using a file instead of the database
- Generate the file with Jive
  - Choose server -> right click -> save server data
- It is possible
  - Get, update, delete class properties
  - Get, update, delete device properties
  - Get, update, delete class attribute properties
  - Get, update, delete device attribute properties

DS using a File as Database

DS using a File as Database

■ Start the device server on a specified port

> MyDs inst –file=<file_path> -ORBendPoint giop:tcp::<port>

■ Device name used in a client must be changed

– With database:

- sr/d-fuse/c04

– With file as database:

- tango://<host>:<port>/sr/d-fuse/c04#dbase=no

DS using a File as Database

■ Limitations
- Modifications are not reported back to the database
- No check that the same device server is running twice
- Manual management of host/port
- No alias

DS not using a Database at all!

- It is also possible to start a device server without using a database at all
  - Do not code database access within the device server…
- The option is **–nodb**
- Another option –dlist allows the definition of device names at the command line for the highest tango class

DS not using a Database at all

- A method DeviceClass::device_name_factory is used to define device names for a class, when it is not possible to define them at command line

MyDs inst –nodb –dlist id13/pen/1,id13/motor/2
-ORBendPoint giop:tcp::<port>

DS not using a Database at all

- Change of device name
  - tango://<host>:<port>/sr/d-fuse/c04#dbase=no
- Limitation
  - The same as for a server with file database
  - No properties at all
  - No events

Multi TANGO_HOST

- A client running in control system A is able to access devices running in control system B by specifying the correct name

- Full Tango device name syntax

[protocol://][host:port]device_name[/attribute][->property][#dbase=xx]

- Examples
  - tango://freak:1234/id00/pen/c11#dbase=no
  - tango:://orion:10000/sr/d-vlm/1

Tango Control System with Several Database Servers

- Defined using the TANGO_HOST environment variable
- Client and servers will automatically switch from one server to the other if one dies

TANGO_HOST=controls01:10000,controls01:15000

DB 1
on port
10000

DB 2
on port
15000

MySQL
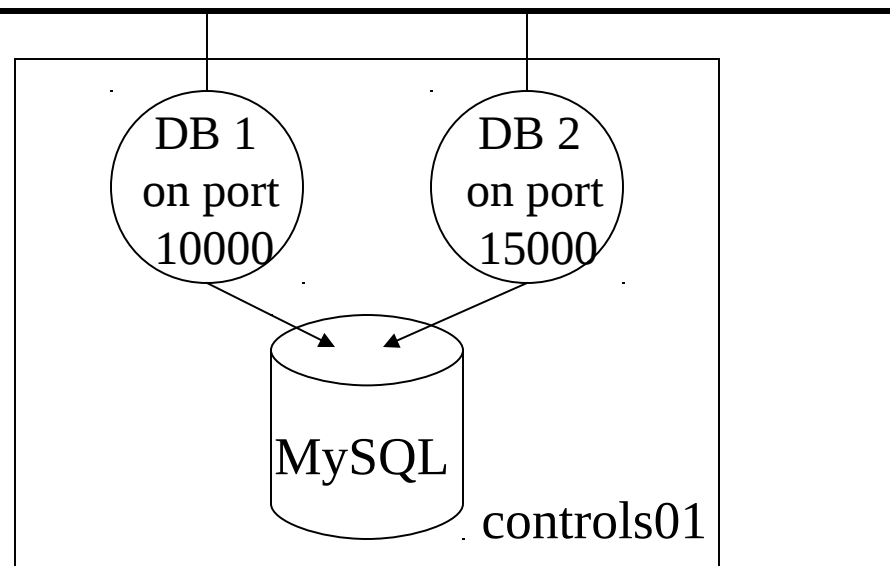
controls01

Tango CS Administration

■ The goal:

   – Overview of all hosts in a control system and all running device servers

   – Start/stop device servers in the control system from a central point

   – Diagnose rapidly problems or failures

■ To administrate a Tango control system you need:

   – The Starter device server on every host

   – Astor, the administration application

Tango CS Administration

■ The Starter server is able to
  – Start even before the database is running and wait for it
  – Get the list of device servers configured for the host from the database
  – Start device server(s)
    • Manage 5 (default) startup levels for ordered startup
  – Kill a device server (command "kill" of the admin device)
  – Check that a device server is running.
  – Ping the device server process admin device to check if it is alive
  – Check if the notifd is running

Tango CS Administration

■ Run one Starter device server per host in the control system

■ Start the Starter device server using the host name as instance name

Starter <host>

■ The starter device name is (only one device)

tango/admin/<host>

Tango CS Administration

- Astor is a graphical interface to the starter device(s) and is able to
  - Manage host(s) in a tree structure
  - Display the state of hosts and device servers
  - Start / Stop several servers on several host(s) with some clicks
  - See the device server output
  - Open a window on a host
  - Help you creating a new Starter entry for a new host

Tango CS Administration

■ Tools available within Astor:

– Jive

– Polling thread manager

– Polling thread configuration and profiling

– Event configuration and testing

– Device dependency tree

Tango CS Administration

Astor - Starter - Database
Principle

Tango CS Administration

Tango CS Administration

■ **Host (Starter) actions:**

- Open a control panel (see servers)
- Remote login (not for win32)
- Starter test
- Clone (create a new Starter in database)
- Cut /Paste (to manage tree)
- Edit properties (Starter $PATH, comments…)
- Remove

200

Tango CS Administration

- ■ **<u>Server actions:</u>**
  - – Start / kill server
  - – Restart (kill wait a bit and start)
  - – Set startup level
  - – Polling management
  - – Configuration (using the server wizard)
  - – Server and class info
  - – Test a device
  - – Check states
  - – See standard error

The Device Wizard

■ Available from Jive or Astor

■ Allows a user to create and configure a new device server dynamically in the database without knowledge on

- Available classes in the server
- Usable device properties when creating new devices

■ The wizard will

- Automatically retrieve class properties and will ask for new values
- Automatically retrieve device properties and will ask for new values

The Device Wizard

Class Selection

The server has been succesfully started and has 1 class(es). Keep in mind that modifying exiting class property may affect other running server.
Click [Edit Class] to edit properties of the selected class
Click [Declare device] to continue with device declaration.

Server:
PsImpl/jm

PsImpl

< Back   Edit Class   Declare device   Cancel

Property: InterfaceBoard

The interface board to be used. Can be 1 to 8.

Server:
PsImpl/jm
Class:
PsImpl
Device:
ps/tes/jm

1

Set Default
View Default

< Back   Next >   Cancel

Polling Management

- Available from Astor
- Thread pool management
- Polling configuration
- Polling profiling

Polling Management
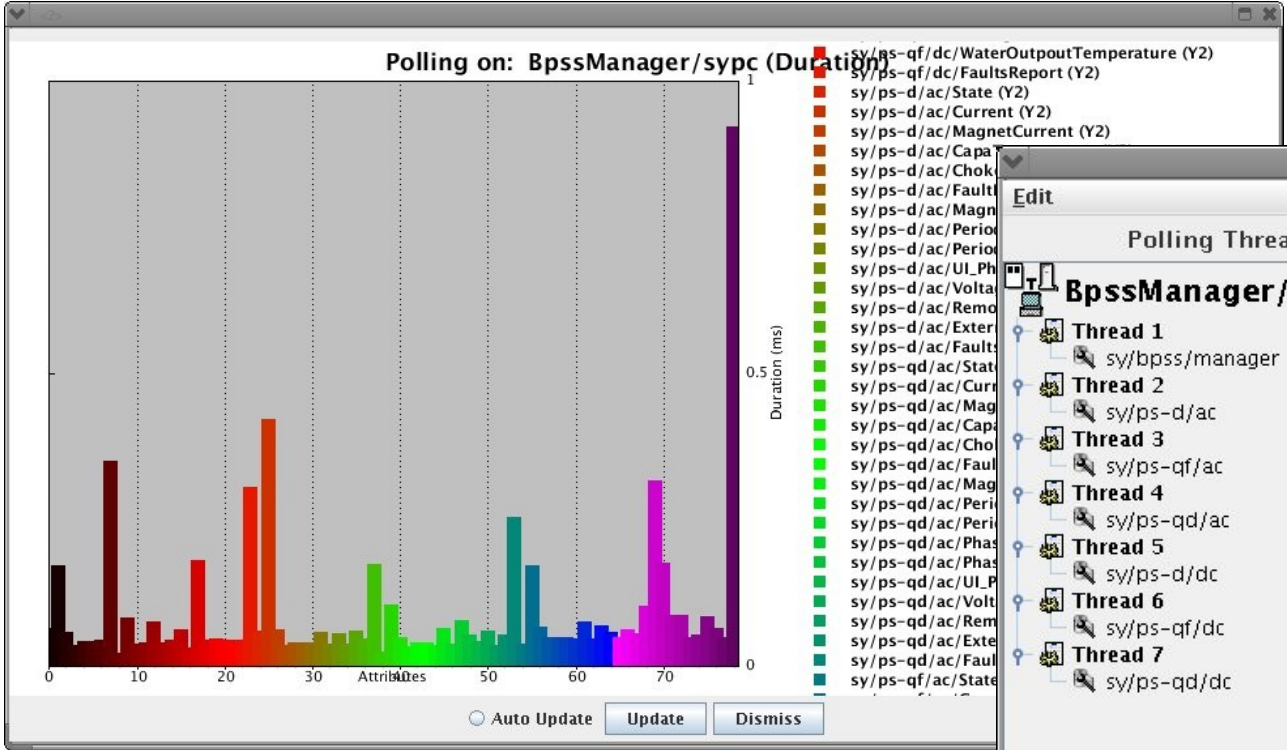
Event Manager

- Available from Astor
- Configure periodic, change and archive events
- Subscribe and test a set of events

02/09/17

Event Manager

Device Dependency Tree

- Available from Astor

- Shows all open connections to sub devices for every device in a device server

- Connections which cannot be directly attributed to a device are listed under the administration device name

02/09/17

208

Device Dependency Tree

Access Control

- Allows to restrict user access on devices:
  - Reading is always possible
  - Writing must be allowed
- A default access need to be defined
- For a user can be defined:
  - A list of allowed host or network addresses
  - A list of READ_WRITE or READ_ONLY devices

Access Control

- To enable access control:
  - Create the free property **CtrlSystem** (if not yet available)
  - Start the TangoAccessControl service as

    TangoAccessControl 1

  - Execute the command **RegisterService** on the device **sys/access_control/1**
  - Start Astor and open the **Access Control** panel from the tools menu

Access Control

How to configure TANGO access control:

Tango Access Control Manager

File   Action

Access to   pcantares:10000

All Users
- **Allowed Addresses**
  - `*.*.*.*`
- **Devices**
  - `*/*/*`

id01
- **Allowed Addresses**
  - `160.103.21.*`
- **Devices**
  - `fe/*/*`

operator
- **Allowed Addresses**
  - `160.103.10.*`
  - `160.103.11.*`
  - `160.103.12.*`
- **Devices**
  - `*/*/*`

verdier
- **Allowed Addresses**
  - `160.103.*.*`
- **Devices**
  - `*/*/*`
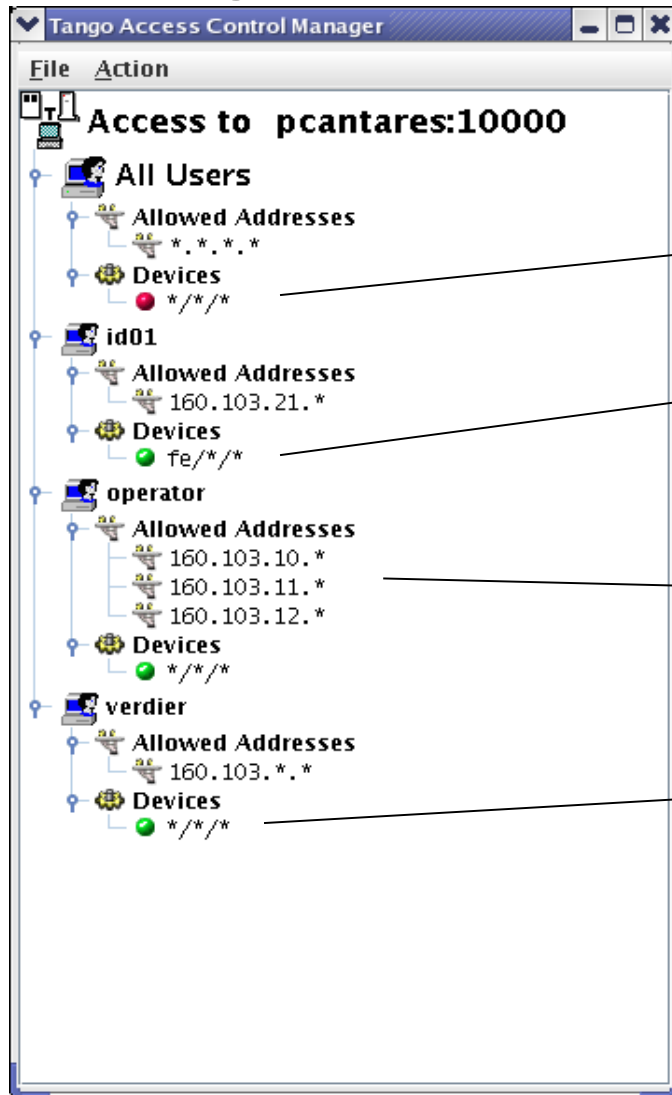
**All user are in read only mode.
(default mode)**

**User id01 can access only devices of the domain FE
from it's network.**

**Operator can access all Tango devices
from control room networks.**

**User verdier (administrator) can access all Tango devices
from all ESRF network.**

Exercise 9

- Add the device server with a start-up level in Astor

- Create a polling thread for every MaxLabPowerSupply device and configure the polling of the Current attributes

- Configure change events for the Current attributes and test the events

Tango Training: Part 9: Graphical User Interfaces

- GUI Toolkits
- ATK
- Synoptic Views
- Panel Builder

GUI Toolkits

■ Java :

– ATK based on Java Swing

– Widgets a Java Beans

■ C++ :

– Qtango based on Qt

– Can be used in QtDesigner

■ Python

– Tau based on PythonQt

– Can be used in QtDesigner

GUI Toolkits

- All toolkits follow the MVC model
- All toolkits are based on a core and a widget libray
- All toolkits implement a device and an attribute factory (DeviceProxy only once)
- All toolkits abstract data reception
  - Use events when available
  - Otherwise polling

GUI Toolkits

- Provides a framework to speed up the development of Tango Applications

- Helps to standardize the look and feel of the applications

- Implements the core of "any" Tango Java client

- Is extensible

ATK Software Architecture

Application

Tango ATK

Java Swing

Tango Java API
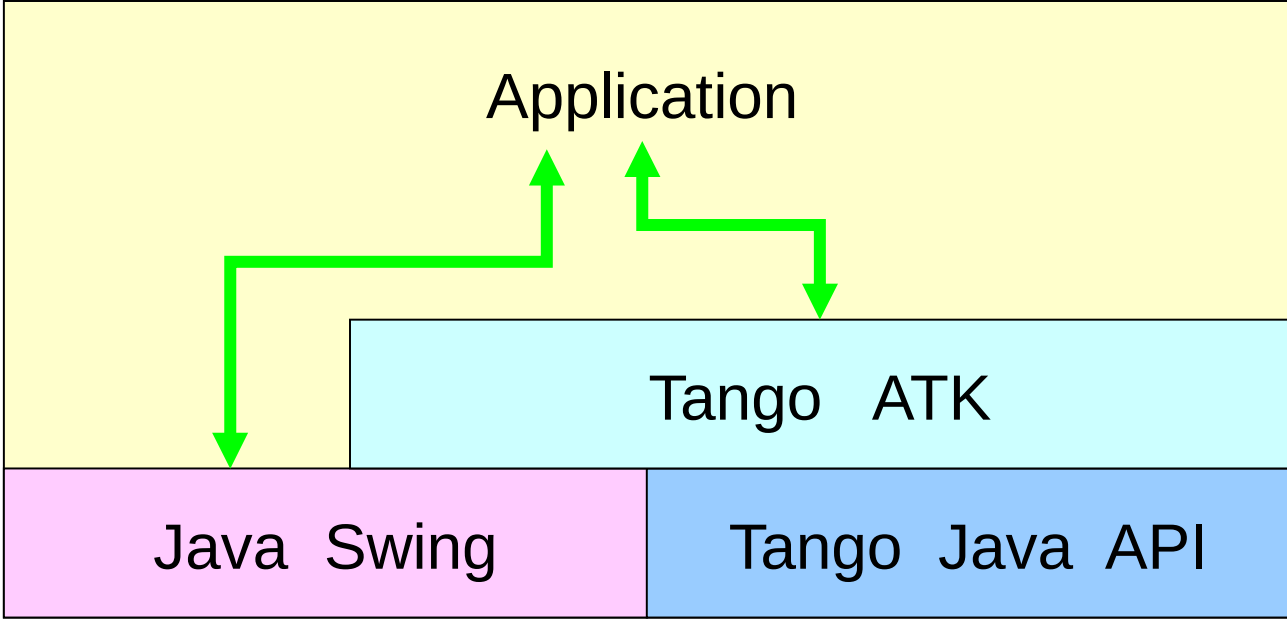
ATK Software Architecture

Control

Application

Myviewer.setModel(coreObject)

Model

ATKCore

Attribute

AttributeList

Command

CommandList

… etc.

Tango Java API

View

ATKWidget

NumberSpectrumViewer

NumberScalarListViewer

CommandComboViewer

StateViewer

… etc.

Java Swing
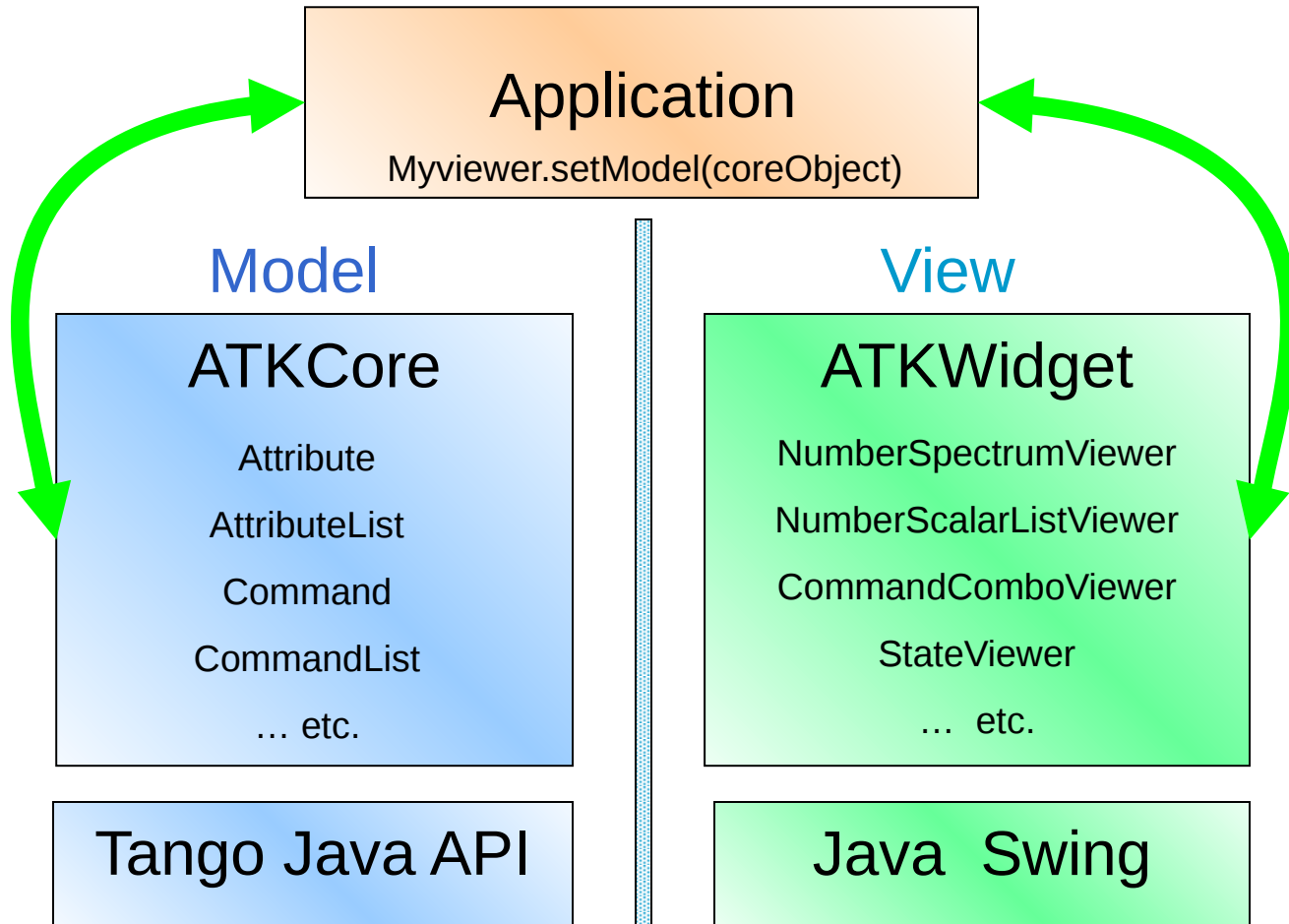
Inside ATK

ATKCore sub-package provides the classes which implement the model

View

Model

ATKWidget

ATKCore

Tango Java API

ATK Attribute Viewer

ATK Attribute Viewer

-2.31 kV

Notify all it's attribute listeners

Attribute Change Event

Attribute

Connects to

Tango Device Attribute

Subscribes to Tango Events

Error occurs

Notify all it's error listeners

ATK Error Viewer

Inside ATK

ATKWidget sub-package provides the classes to view and to interact with ATKCore objects

View

ATKWidget

Java Swing
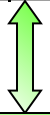
NumberSpectrumViewer
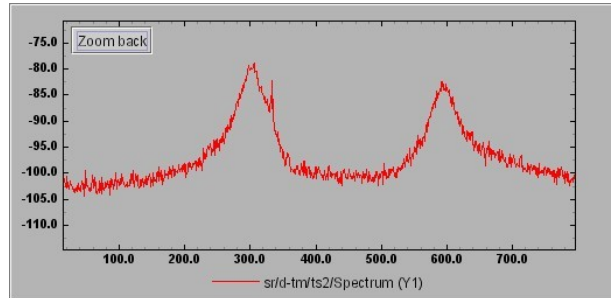
NumberImageViewer
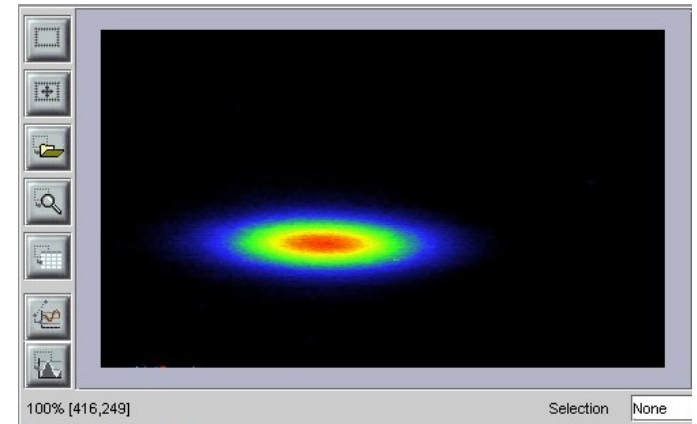
CommandComboViewer

ScalarListViewer
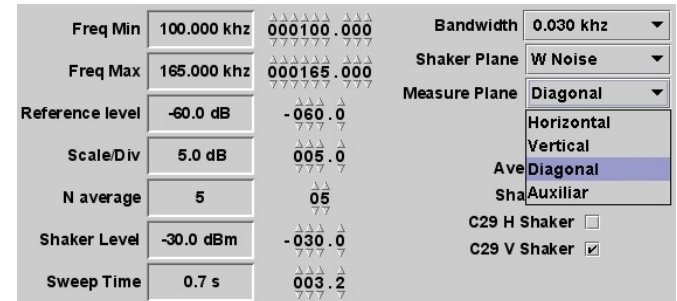
Synoptic

Jdraw editor to draw the synoptic with vector graphics

Give the "panel" class name to be popped up when this object is clicked
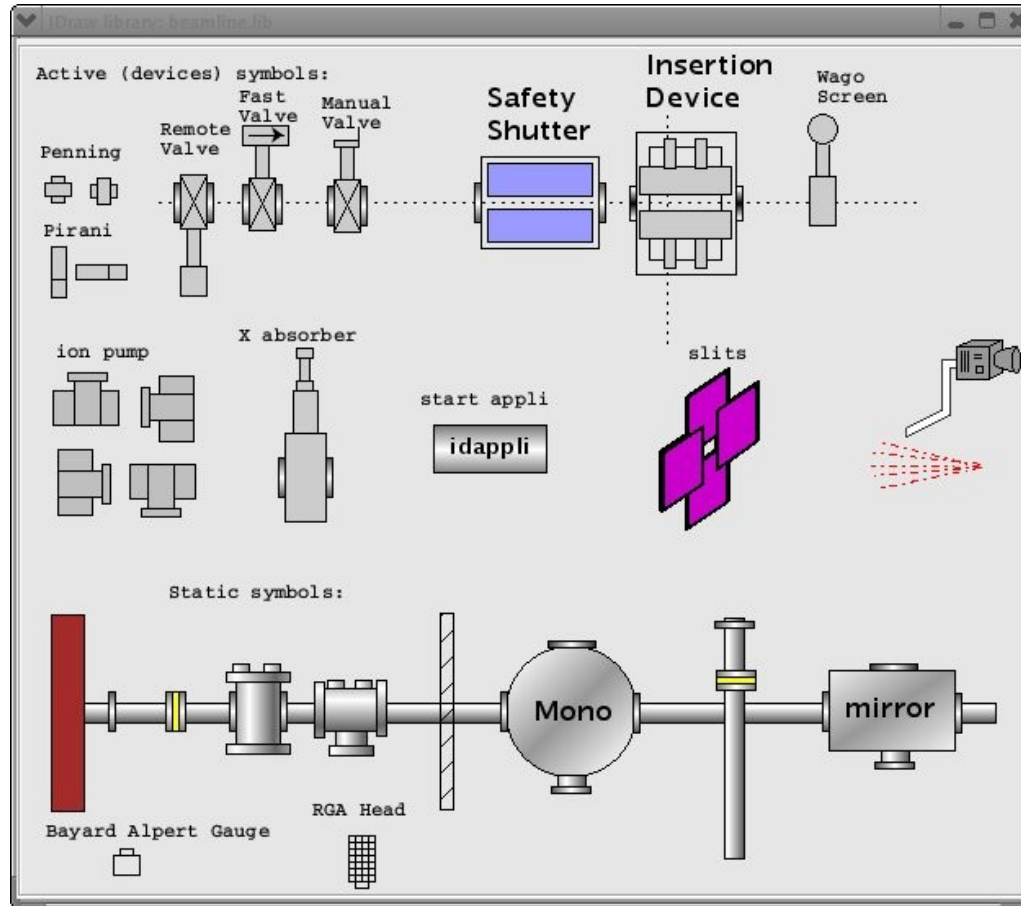
Synoptic

Graphical component libraries can be created

Synoptic

Launch the ready to use ATK application "SimpleSynopticAppli" to test the synoptic at run time
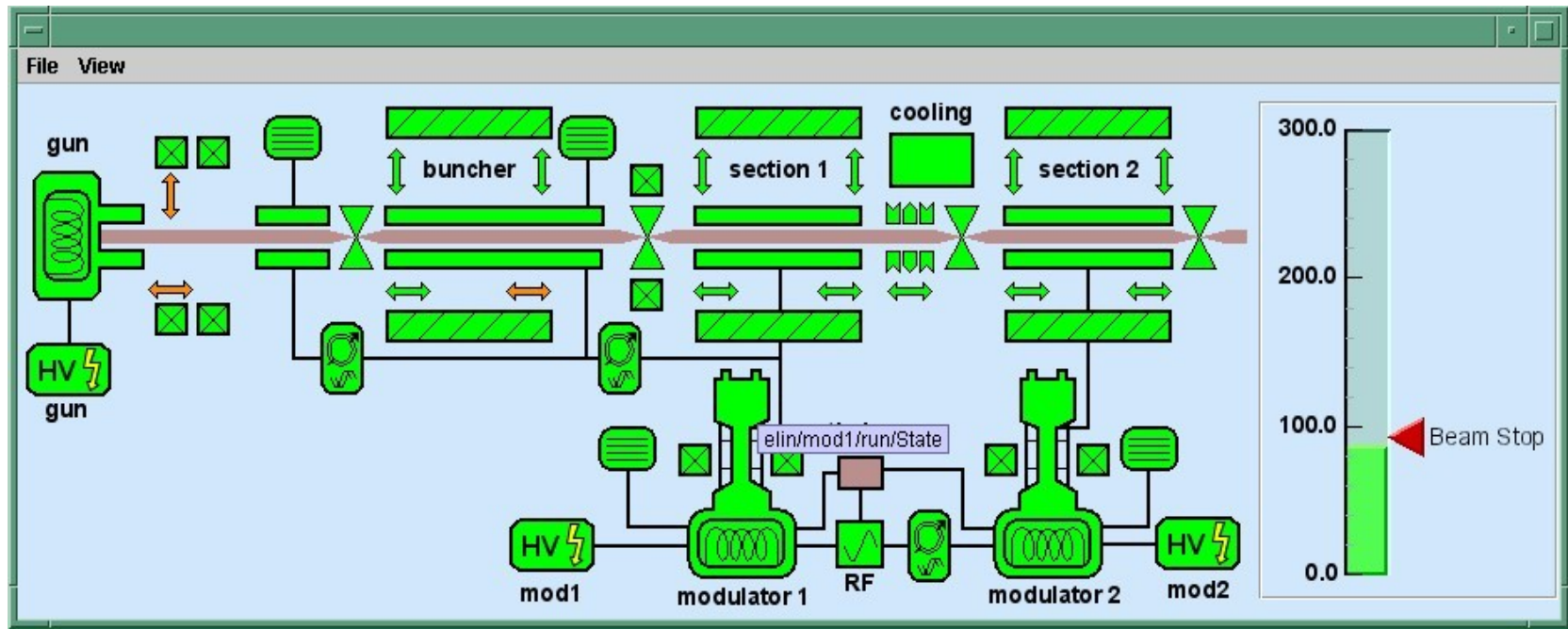
Synoptic

Design your own specific ATK application using your favorite Java IDE
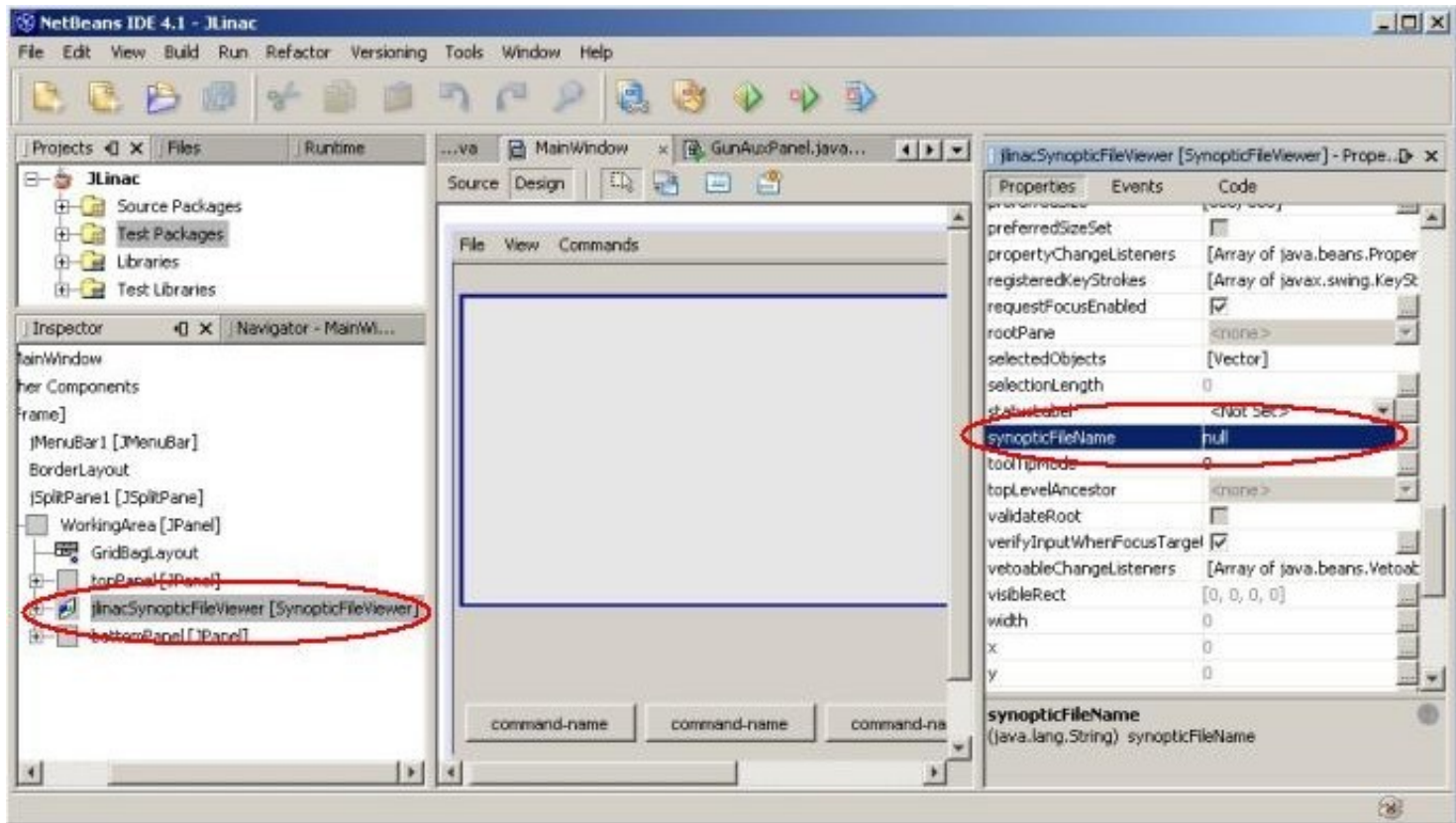
Synoptic

Final synoptic application
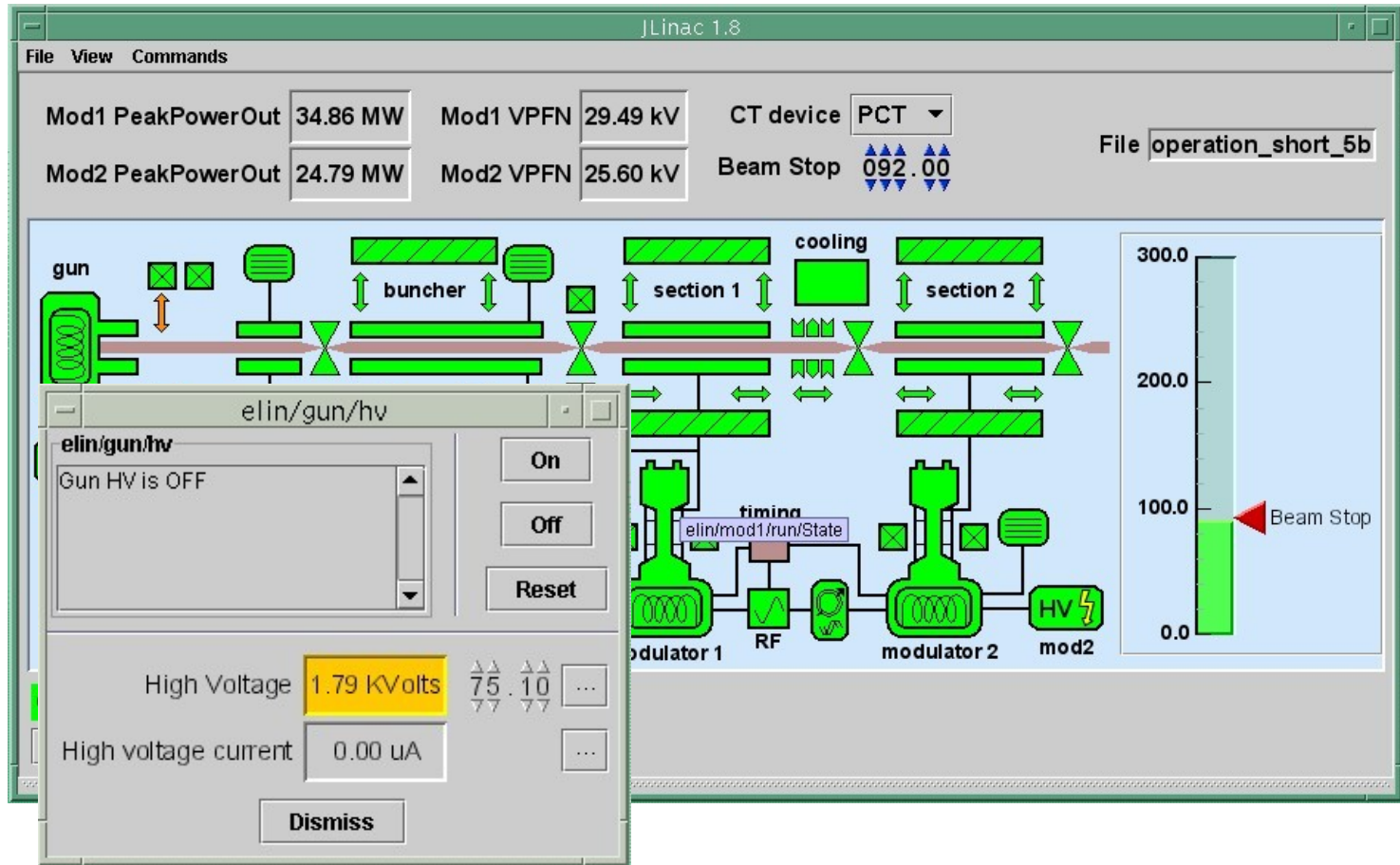
Panel Builder JDDD

- JDDD = Java Doocs Data Display
  http://jddd.desy.de
- Developed at DESY (MCS group)
- Interactive panel builder
- Stores panels in XML format
- Can use ATK widgets as plugin
- A JDDD version with the configured ATK plugin can be downloaded from:
  - http://www.tango-controls.org/download

Panel Builder JDDD

- **Interesting concepts**
  - Hierarchical panel usage
  - Can handle several application layers
  - Address inheritance through the components is possible
    - Configure a device name only once for the whole panel
  - Allows the use of a SVN repository to store and retrieve panel files
  - Easy to use logic and animation features
  - Wild card addressing for ATK widgets

02/09/17

Panel Builder JDDD

Examples

More information : ➢http://www.tango-controls.org
                              ➢http://www.tango-controls.org/tutorials
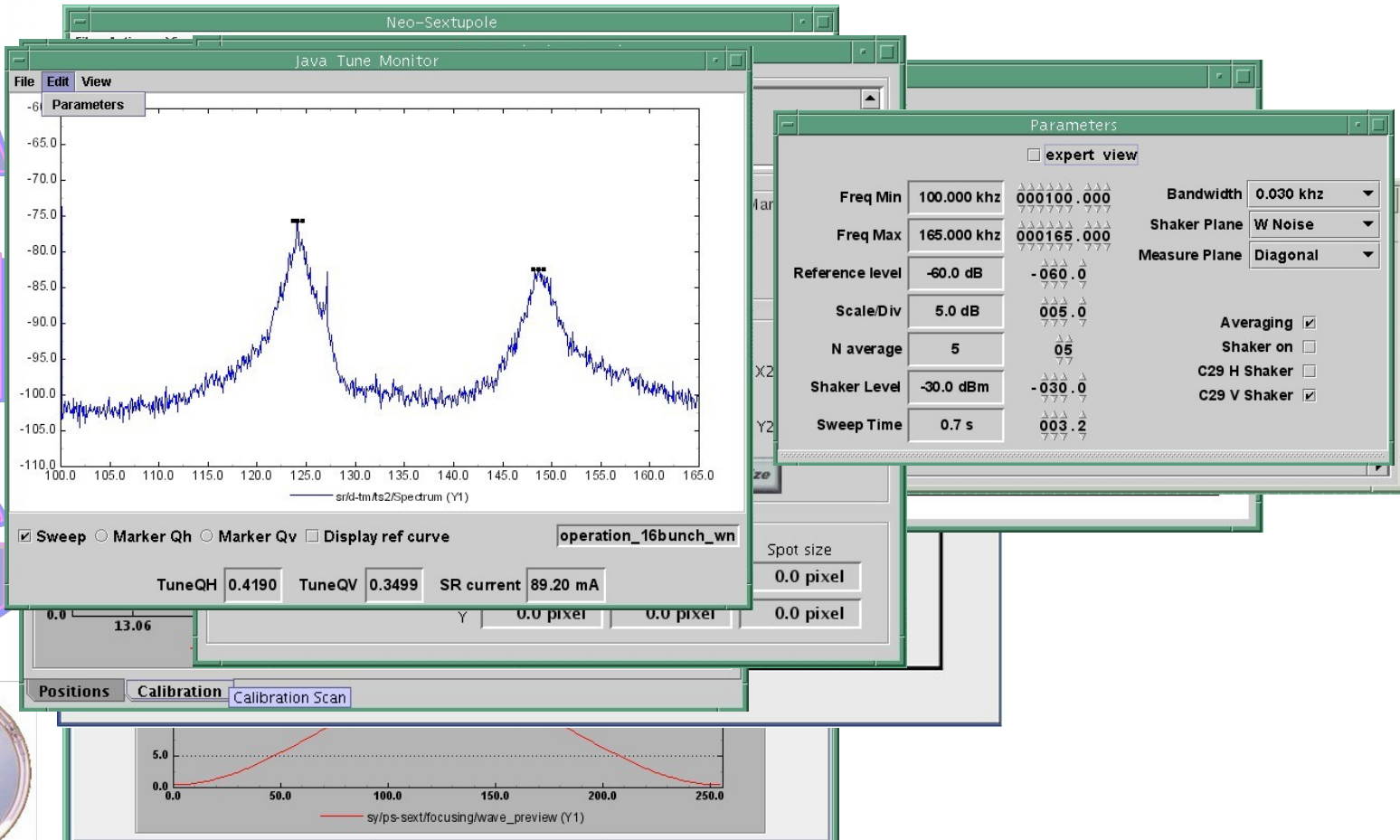
Exercise 10

- Create a panel or synoptic to drive three MaxLabPowerSupply devices
  - Commands On, Off
  - Current reading and writing
  - State and status

Tango Training: Part 10 : Archiving System

- HDB
- TDB
- Snapshots

Archiving System

- A set of three databases to keep history of what's going on in the control system
  - HDB (History Database)
  - TDB (Temporary Database)
  - Snap (Snapshot database)
- Two supported underlying database systems
  - Oracle (Soleil)
  - MySQL (Alba, Elettra, ESRF)

Archiving System

- Implemented using
  - A set of Java device servers to
    - Get data from the control system
    - Send extracted data to the requesting client
  - JDBC to access the database itself
- Running 7 days a week, 24 hours a day

HDB / TDB

- Storage of data coming from attributes only
  - Command result storage is not supported
- HDB is dedicated to long term storage
  - Data are never deleted
  - Smallest storage period = 10 sec (0.1 Hz)
- TDB is dedicated to temporary storage
  - 3 days max (configurable)
  - Smallest storage period = 0.1 sec (10 Hz)

HDB / TDB

- **Several storage modes:**
  - Periodic: Data stored at a fixed period (mandatory)
  - Different:
    - Data stored when reading is different from the last stored value
    - Data stored when the difference between read value and last stored value is greater/lower than an absolute limit
    - Data stored when the difference between read value and last stored value greater/lower than a limit in %
  - Threshold: Data stored greater/lower than a pre-defined threshold

HDB / TDB

■ Device servers common for HDB / TDB

– ArchivingManager

• Provide global command(s)

• Load balancing

HDB

- **Device servers for HDB**
  - HdbArchiver(s)
    - Collect data from the control system and store them in the database
      - Uses polling of devices
      - Can be configured to receive archiving events
        - » Not yet documented
        - » Only handled by Mambo for data extraction
  - HdbExtractor(s)
    - Extract data from the database and send them to caller
  - HdbArchivingWatcher
    - Diagnosis tool : detecting abnormal archiving interruption
    - Recovery : reactivate archiving on failed attributes

TDB

■ **Device servers for TDB**

- TdbArchiver(s)
  - Collect data from the control system and store them in the database
    - Uses only polling
- TdbExtractor(s)
  - Extract data from the database and send them to caller
- TdbArchivingWatcher
  - Diagnosis tool : detecting abnormal archiving interruption
  - Recovery : reactivate archiving on failed attribute
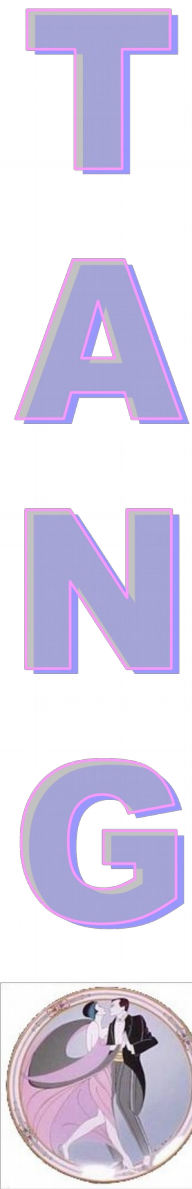
HDB / TDB

- Mambo
  - Configure HDB and TDB
  - Display of data stored in HDB / TDB
  - Handle user configurations
- Mambo as web-start application
  - Uses the Tango web protocol
- E-Giga
  - Display of data coming from HDB in your WEB browser

HDB / TDB

MAMBO : Configuration and Extraction application

Exercise 11

- Store the currents of the MaxLabPowerSupply devices
  - in HDB
    - Every 60 seconds
    - On value change, check every 10 seconds
  - In TDB
    - Every second
- Read stored data with Mambo
- Read stored data with AtkMoni from the HDB extractor server

SNAP

- Capability to take a picture of a set of attributes at a time
  - Motors positions before a planned electric halt
- Compare quickly and easily the attributes values
  - Before and after an experience to analyse the beamline parameters evolution
- Send instructions easily to several equipments
  - Set the beamline in a configuration reference

02/09/17

SNAP

■ Device servers for Snap

– SnapManager
  • Manage snapshot configuration
  • Send command(s) to SnapArchiver

– SnapArchiver
  • To take the snapshot and send the data to the database

– SnapExtractor
  • To extract snapshot data from database

SNAP

BENSIKIN : Configuration and Exploitation

Exercise 12

- Configure a snapshot to store the actual Current values of the MaxLabPowerSupply devices

- Change the power supply Current set points

- Apply the stored snapshot to the power supplies

Tango Training: Part 11 : Miscellaneous

- Getting software
- Who is doing what

Getting the Tango Core

- You can download Tango from the ESRF Tango WEB page (http://www.tango-controls.org/download)
  - As a source package for UNIX like OS
  - As a Windows binary distribution
- For Unix (and co), do not forget to first download, compile and install
  - omniORB
  - omniNotify
- For Windows all libraries and binaries for omniORB and omniNotify are included in the distribution.

Getting the Tango Core

- In both distributions, you have
  - Tango core (libraries and jar files)
  - Database device server and a script to create the Tango database for MySQL
  - Pogo, Jive, LogViewer, jdraw
  - Astor and Starter device server
  - A test device server (TangoTest)
  - ATK

Getting the Tango Core

■ For the UNIX like OS source distribution, you have to compile everything with the famous three commands

- configure
- make
- make install

Tango Core Sources

■ All Tango core sources are stored in a CVS server hosted by SourceForge called   Tango-cs (http://sourceforge.net/projects/tango-cs/)

■ On this project, you find sources for
  – C++ libraries and Java API
  – Database, Starter and TangoTest device servers
  – Pogo, Astor, Jive, LogViewer and ATK
  – Binding for Python, Matlab and Igor
  – The Tango archiving system

Getting Tango Classes

■ Nearly all Tango classes (> 200) are available for download on the WEB from Tango related WEB sites

■ Two kind of classes

– Common interest classes and interfaces to commercial hardware

– Specific classes to interface institute specific hardware

Getting Tango Classes

- On the WEB for each class, you find the HTML pages generated by Pogo
- Common interest classes sources are stored in a CVS server hosted by SourceForge
  - Project name = tango-ds
  - http://sourceforge.net/projects/tango-ds/
- Local classes sources are stored in a local CVS repository at each institute

Getting Tango Classes

Tango Core Development

■ ELETTRA:

– Alarm system

– Canone: A WEB interface using PHP

– E-Giga: A WEB interface above the Tango archiving system

– QTango: ATK like GUI toolkit in C++

• Using QT

Tango Core Development

■ SOLEIL:

– Archiving system

• Using ORACLE or MySQL

– Snapshot system

• Using ORACLE or MySQL

– Matlab and Labview bindings

– WEB protocol for ATK

Tango Core Development

■ ALBA:

- Python binding (PyTango release 4.x)
- Sardana: Control software for experiments
- Tau: ATK like GUI toolkit in Python
  - Using QT

Tango Core Development

■ ESRF:
- Tango libraries (C++ and Java)
- Pogo
- Jive
- Astor / Starter
- ATK