Training TANGO

Users Session

04.02.2003
TANGO : introduction

A «computing tool» dedicated to the implementation of distributed systems, heterogeneous and oriented control/commande (switch)
TANGO : introduction

- **Distributed Systems ?**
  The system components are geographically distributed on machines connected through a computing network

- **Heterogeneous Systems ?**
  A coherent whole made from heterogeneous hardwares and softwares

- **oriented Systems ctrl/command ?**
  Services adapted to a control system (storage, logging, alarms, ...)
TANGO : introduction

How does TANGO solve the contraints of activity distribution and of interoperability (interaction) of the heterogeneous components?
TANGO : introduction

Machine A

Application

1

L_\infty

Application

SE_1

Machine B

Application

3

L_\beta

Application

SE_2

Application

4

L_\beta

?
TANGO: introduction

Machine A

Application 1

Application 2

L_\infty

SE_1

Machine B

Application 3

L_\beta

Application 4

L_\beta

SE_2
TANGO : introduction

Machine A

Application 1

Application 2

$\beta$

$\infty$

$SE_1$

Machine B

Application 3

$\beta$

$SE_2$

Application 4
TANGO: introduction

Machine A

Application 1

$1 \rightarrow L_{\infty}$

Application 4

$L_{\beta}$

SE$_1$

Machine B

Application 3

$L_{\beta}$

Application 4

$L_{\beta}$

SE$_2$
TANGO: introduction

Machine A

Application 1

Application 2

L₁

L₂

Applications

SE₁

Machine B

Application 3

Application 4

L₃

L₄

Applications

SE₂
TANGO: introduction
TANGO: introduction
TANGO : introduction
TANGO: introduction
TANGO : introduction
TANGO: introduction
A standard tool in charge of the communications between software components making up distributed and heterogeneous applications
TANGO : introduction

- CORBA = 1 generic tool
  - support to the development of distributed applications
  - A powerful but cumbersome tool
  - No functionalities «ctrl/cmd oriented»
TANGO : introduction

- a framework CORBA ctrl/cmd oriented
  - A toolbox to implement the system
  - A specialization of CORBA adapted to the needs
TANGO: introduction

- Unifier kernel (core) of the system
  - Overall consistency
TANGO : introduction

Control Systm
SOLEIL
(applications network)

Client application.
Server Application.

Interface
Application Logic
TANGO

Application Logic
Hardware
TANGO
TANGO : introduction

- Unifier kernel (core) of the system
  - Overall consistency

- factorize the services
  - factorize additions and corrections
TANGO: introduction

- Unifier kernel (core) of the system
  - Overall consistency

- factorize the services
  - factorize additions and corrections

- standardize the applications
  - harmonization of the applications structure
TANGO : introduction

- Unifier kernel (core) of the system
  - Overall consistency

- factorize the services
  - factorize additions and corrections

- standardize the applications
  - harmonization of the applications structure

- masks technical details
  - 1 interface of simplified programmation (APIs)
  - focus on the application logic
TANGO : introduction

Philosophy ...

- simplicity
  - mask the CORBA mechanisms
  - propose simplified APIs

- genericity
  - Enable the writing of generic clients
  - On the communications point of view: 1 single object type
TANGO : introduction
**TANGO : device**

- an abstract concept : the «device»
  - central component of the structure (architecture)
TANGO : device : definition

- device = 1 <entity> to be controlled
  - Hardware of software
  - device «physical» / device «logical»

- device = 1 polymorphous object
  - 1 equipment (ex: 1 power supply)
  - 1 collection of equipments (ex: 1 motor + 1 encoder)
  - 1 devices aggregate (ex: a beamline)
  - 1 application (ex: 1 agent of the storage service)
TANGO: device: définition

- device = 1 equipment
  - The most simple and widespread case

SC User

Device

BOO-B1/VI/PI55-3

Interface HW équipement

Ctrl/power supply box

Unique Identity

(SOLEIL nomenclature)

Ionical Pump

Equipement
TANGO : device : definition

- device = 1 application
- Logic device

Device 1
\( G_1 = f(\Psi_1) \)

Device 2
\( G_2 = f(\Psi_2) \)

Device 3
\( \Psi_3 = h(G_3) \)

Device 4
\( G_3 = g(\alpha, G_1, G_2) \)
TANGO: device: class

Belongs to a class
- member of a devices’ family
- derived from a basic (common) class
  - generical behavior / common core
TANGO : device : interface

Interface (public)

Implementation (private)
TANGO : device : interface

- Owns a communication interface
  - interface device <=> class
  - interface = commands + attributes
    - commandes ≈ actions
    - attributes ≈ physical units

<table>
<thead>
<tr>
<th>Implementation/Realization</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>Commands</td>
</tr>
<tr>
<td></td>
<td>attributes</td>
</tr>
<tr>
<td>Generic</td>
<td>Specific</td>
</tr>
<tr>
<td>Init</td>
<td>PowerOn</td>
</tr>
<tr>
<td>State</td>
<td>PowerOff</td>
</tr>
<tr>
<td>Status</td>
<td>-</td>
</tr>
<tr>
<td>private</td>
<td>current</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Implementation/Realization</td>
</tr>
</tbody>
</table>
TANGO: device: interface

- Owns a communication interface

- interface device <=> class
- interface = commands + attributes
  - commands ≈ actions
  - attributes ≈ physical units

---

**Interface**

<table>
<thead>
<tr>
<th>Commands</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic</td>
<td>Specific</td>
</tr>
<tr>
<td>Init</td>
<td>PowerOn</td>
</tr>
<tr>
<td>State</td>
<td>PowerOff</td>
</tr>
<tr>
<td>Status</td>
<td></td>
</tr>
</tbody>
</table>

**Implementation/Realization**

- Public
- Privé
TANGO : device : interface : command

- 1 action
- 0 ou 1 entry argument (argin)
- 0 ou 1 exit argument (argout)
- argin & argout = 1 of the 20 TANGO types
- Execution : indirect mecanism
  - -> generic approach of TANGO
  - -> 1 CORBA method : command_inout
    - belongs to the generic interface of the devices
    - dedicated to the fulfillment of non generic commands
      - Device’s specificity (PowerSupply, StepperMotor, ...)
    - only one signature : 400 combinations argin/argout !
    - generic containers (CORBA::any)
### About the argin & argout type...

<table>
<thead>
<tr>
<th>TANGO</th>
<th>Desc</th>
<th>Matlab</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV_VOID</td>
<td>no argin and/or no argout</td>
<td>-</td>
</tr>
<tr>
<td>DEV_STATE</td>
<td>Device status</td>
<td>1-by-n char array</td>
</tr>
<tr>
<td>DEV_STRING</td>
<td>Characters chain</td>
<td>1-by-n char array</td>
</tr>
<tr>
<td>DEV_BOOLEAN</td>
<td>boolean</td>
<td>1-by-1 uint16 array</td>
</tr>
<tr>
<td>DEV_SHORT</td>
<td>Integer 16 bits signed</td>
<td>1-by-1 int16 array</td>
</tr>
<tr>
<td>DEV_USHORT</td>
<td>Integer 16 bits non signed</td>
<td>1-by-1 uint16 array</td>
</tr>
<tr>
<td>DEV_LONG</td>
<td>Integer 32 bits signed</td>
<td>1-by-1 int32 array</td>
</tr>
<tr>
<td>DEV ULONG</td>
<td>Integer 32 bits non signed</td>
<td>1-by-1 uint32 array</td>
</tr>
</tbody>
</table>
TANGO : device : interface : commande

About the argin & argout type ...

<table>
<thead>
<tr>
<th>TANGO</th>
<th>Desc</th>
<th>Matlab</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV_FLOAT</td>
<td>real 32 bits</td>
<td>1-by-1 single array</td>
</tr>
<tr>
<td>DEV_DOUBLE</td>
<td>real 64 bits</td>
<td>1-by-1 double array</td>
</tr>
<tr>
<td>DEVVAR_CHARARRAY</td>
<td>Octets chart (i.e. characters)</td>
<td>1-by-n char array</td>
</tr>
<tr>
<td>DEVVAR_SHORTARRAY</td>
<td>Integers chart 16 bits signed</td>
<td>1-by-n int16 array</td>
</tr>
<tr>
<td>DEVVAR_USHORTARRAY</td>
<td>Integers chart 16 bits not signed</td>
<td>1-by-n uint16 array</td>
</tr>
<tr>
<td>DEVVAR_LONGARRAY</td>
<td>Integers chart 32 bits signed</td>
<td>1-by-n int32 array</td>
</tr>
<tr>
<td>DEVVAR_ULONGARRAY</td>
<td>Integers chart 32 bits not signed</td>
<td>1-by-n uint32 array</td>
</tr>
<tr>
<td>DEVVAR_FLOATARRAY</td>
<td>reals chart 32 bits</td>
<td>1-by-n single array</td>
</tr>
</tbody>
</table>
### TANGO : device : interface : commande

#### A propos du type d’argin & argout...

<table>
<thead>
<tr>
<th>TANGO</th>
<th>Desc</th>
<th>Matlab</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVVAR_DOUBLEARRAY</td>
<td>Reals chart 64 bits</td>
<td>1-by-n double array</td>
</tr>
<tr>
<td>DEVVAR_STRINGARRAY</td>
<td>Non bounded characters chains chart</td>
<td>1-by-n cell array of {1-by-n char array}</td>
</tr>
</tbody>
</table>
| DEVVAR_LONGSTRINGARRAY       | structure containing an integers chart 32 bits signed and a characters chains chart | 1-by-n struct array {
|                              |                                                        |    field lvalue:
|                              |                                                        |        1-by-n int32 array
|                              |                                                        |    field svalue:
|                              |                                                        |        1-by-n cell array of
|                              |                                                        |        {1-by-n char array}
| DEVVAR_DOUBLESTRINGARRAY     | structure containing a reals chart 64 bits and a characters chains chart | 1-by-n struct array {
|                              |                                                        |    field dvalue:
|                              |                                                        |        1-by-n double array
|                              |                                                        |    field svalue:
|                              |                                                        |        1-by-n cell array of
|                              |                                                        |        {1-by-n char array}
|
**Syntaxe**

- Prog. env. OO (C++, Java, Python)
  
  ```
 argout = dev.command_inout (cmd_name, argin)
  ```

- User env. (Matlab, Igor Pro, ...)
  
  ```
 argout = tango_command_inout (dev_name, cmd_name, argin)
  ```

**Examples Matlab**

```matlab
>> help tango_command_inout
>> dev = 'tango/tangotest/1'
>> tango_command_inout(dev,'DevDouble',pi)
>> tango_command_inout(dev,'DevVarDoubleArray',[1,2,3])
>> s.dvalue = [pi, 2*pi, 3*pi]
>> s.svalue = {'dev', 'var', 'double', 'array', 'test'}
>> tango_command_inout(dev,'DevVarDoubleStringArray',s)
```
TANGO : device : interface : command

Name and signature of the commands ?

* Device’s Documentation

* Prog. env.. OO (C++, Java, Python)
  * cmd_list_info = dev.command_list_query()
  * cmd_info = dev.command_query(cmd_name)

* User env. (Matlab, Igor Pro, ...)
  * cmd_list_info = tango_command_list_query(dev_name)
  * cmd_info = command_query(dev_name, cmd_name)

* Examples Matlab
  * >> tango_command_list_query(dev)
  * >> tango_command_query(dev,‘DevDouble’)
TANGO : device : interface : command

Errors processing

- Prog. env. OO (C++, Java, Python):
  - exceptions : mechanism try/catch (DevFailed & derived)
- User env. (Matlab, Igor Pro, ...):
  - error code : updating after each execution of a «command»

Examples Matlab:

```matlab
>> result = tango_command_inout(dev,'dummy',pi);
>> tango_error
if tango_error == -1 then ...
    • result is invalid, indéfini
    • result can be not of the expected type !
    • Do not use it !
>> help tango_error
• an example to follow !
```
TANGO : device : interface : commands
TANGO : device : interface

- Has a communication interface
  - interface device <=> class
  - interface = commands + attributes
    - commands ≈ actions
    - attributes ≈ physical units

<table>
<thead>
<tr>
<th>Interface</th>
<th>Commands</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generics</td>
<td>Specifics</td>
<td>Generics</td>
</tr>
<tr>
<td>Init</td>
<td>PowerOn</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>PowerOff</td>
<td>-</td>
</tr>
<tr>
<td>Status</td>
<td></td>
<td>Implementation/Realisation</td>
</tr>
</tbody>
</table>

public

privé
TANGO : device : interface : attribute

Definition
- Physical unit produced or administrated by the device
- ex: a motor’s position, alimentation power supply., ...

Format
- From 0 to 2 dimensions
  - SACLAR
  - SPECTRUM (i.e. vector)
  - IMAGE (i.e. matrix)

Type
- DEV_SHORT, DEV_LONG, DEV_DOUBLE
  • scalar, spectrum or image
- DEV_STRING
  • scalar only
TANGO : device : interface : attribute

Accessibility

- **READ**
  - accessible in read only

- **WRITE**
  - accessible in write only

- **READ_WRITE**
  - accessible in read AND in write only
  - Consigne (instructions) vs effective value

- **READ_WITH_WRITE**
  - 1 attribute READ linked to 1 attribute WRITE
  - exotic (prefer READ_WRITE)
TANGO : device : interface : attribute

- **Features : autodescriptive & parametrizable**
- 1 attribute -> 18 properties
  - generic properties (attribute)
  - 8 non-modifiable properties (developer)
    - name : attribute’s name
    - data_type : data type (DEV_SHORT, DEV_LONG, ...)
    - data_format : data format (SCALAR, SPECTRUM or IMAGE)
    - writable : access mode (READ, WRITE, ...)
    - max_dim_x, max_dim_y : dimensions max
      - dim_x <= max_dim_x
      - dim_y <= max_dim_y
    - disp_level : expert or operator
    - wrt_attr_name : name of the attribute WRITE associated
TANGO : device : interface : attribute

10 modifiable properties (user)

- **description**: attr. Description (text)
- **label**: label associated to the attr. (text)
- **unit**: unit in which is expressed the value associated to the attribute (text)
- **standard_unit**: conversion factor to the units MKSA (text)
- **display_unit**: unit * standard_unit (text)
- **format**: display format for the «numerical» attributes (texte)
  - Key-words: fixed, scientific, uppercase, showpoint, showpos, setprecision(), setw()
  - ex: scientific;uppercase;setprecision(3)
- **min_value**: min. value of an attribute WRITE or READ_WRITE (text)
- **max_value**: max. value of an attribute WRITE or READ_WRITE (text)
- **min_alarm**: alarm threshold <low> of an attribute READ or READ_WRITE (text)
- **max_alarm**: alarm threshold <high> of an attribute READ or READ_WRITE (text)
List of the attributes?

- Documentation of the device
- Prog. env. OO (C++, Java, Python)
  - attr_list = dev.get_attribute_list()
- User env. (Matlab, Igor Pro, ...)
  - attr_list = tango_get_attribute_list_list(dev_name)
- Example Matlab
  - >> attr_list = tango_get_attribute_list_list(dev)
TANGO : device : interface : attribute

Standard configuration of an attribute?

- **Prog. env. OO (C++, Java, Python)**
  - `attr_config_list = dev.get_attribute_config(attr_name_list)`
  - `attr_config_list = dev.attribute_list_query()`
  - `attr_config = dev.attribute_query(attr_name)`

- **User env. (Matlab, Igor Pro, …)**
  - `attr_config_list = tango_attribute_list_query(dev_name)`
  - `attr_config_list = tango_get_attributes_config(dev_name, attr_name_list)`
  - `attr_config = tango_attribute_query(dev_name, attr_name)`
  - `attr_config = tango_get_attribute_config(dev_name, attr_name)`

- **Example Matlab**
  - `>> help tango_attribute_list_query`
  - `>> acl = tango_attribute_list_query(dev)`
  - `>> acl(2)`
TANGO : device : interface : attribute

Modify an attribute’s configuration ?

- Take care of the consequences !
  - acts upon all the clients
  - sensitive parameters : min/max_value, min/max_alarm

- Prog. env.. OO (C++, Java, Python)
  - dev.set_attr_config (attr_config_list)

- User Env. (Matlab, Igor Pro, ...)
  - tango_set_attribute_config (dev_name, attr_config)
  - tango_set_attributes_config (dev_name, attr_config_list)

Example Matlab

```matlab
>> help tango_set_attributes_config
>> scc = tango_get_attribute_config(dev, ‘short_scalar’)
>> scc.min_value = num2str(str2num(scc.min_value) / 2)
>> scc.max_value = num2str(str2num(scc.max_value) / 2)
>> scc.description = ‘This is a dummy attribute’
>> tango_set_attribute_config(dev, scc)
```
**Obtain the standard value of an attribute?**

- attributes READ, READ_WRITE ou READ_WITH_WRITE
- Result of the lecture => structure {read value + infos }
  - name : name of the attribute
  - quality : quality of the returned value
    - ATTR_VALID : ok, the returned value is valid
    - ATTR_ALARM : an alarm threshold has been crossed (cf. min_alarm and max_alarm)
    - ATTR_INVALID : error, undefined value
  - dim_x : dim.x of the value (dim_x <= attr_config.max_dim_x)
  - dim_y : dim.y of the value (dim_y <= attr_config.max_dim_y)
  - timestamp : value stamp
  - value : value of the attribute at this very instant <timestamp>
    - SCALAR
      - READ : [0 : val]
      - READ_WRITE et READ_WITH_WRITE : [0:measure, 1:consigne]
    - SPECTRUM
      - [0 : measure, ..., dim_x – 1 : measure]
    - IMAGE
      - [0 : measure, ..., dim_x – 1 : measure] x [0 : measure, ..., dim_y – 1 : measure]
TANGO : device : interface : attribute

- Obtain the standard of an attribute?
  - Prog. env. OO (C++, Java, Python)
    - dev.read_attributes (attr_name_list)
  - User Env. (Matlab, Igor Pro, ...)
    - tango_read_attribute (dev_name, attr_name)
    - tango_read_attributes (dev_name, attr_name_list)

- Example Matlab
  - >> help tango_read_attribute
  - >> scv = tango_read_attribute (dev, ‘short_image’)
  - >> datestr(scv.time)
  - >> for i=1:10 s=tango_read_attribute(dev, ‘short_spectrum’);
     plot(s.value); drawnow; end;
TANGO : device : interface : attribute

Modify the value of an attribute?

- attributes WRITE, READ_WRITE et READ_WITH_WRITE
- attr_config.min_value <= set value <= attr_config.max_value
  - exception API_WAttrOutsideLimit
- Prog. env. OO (C++, Java, Python)
  - dev.write_attributes (attr_val_list)
- User Env. (Matlab, Igor Pro, ...)
  - tango_write_attribute (dev_name, attr_name, value)
  - tango_write_attributes (dev_name, attr_name_attr_value_struct_list)
- example Matlab
  - >> help tango_write_attribute
  - >> tango_write_attribute (dev, 'short_scalar', 123456789)
  - >> tango_print_error_stack
  - >> tango_write_attribute (dev, 'short_scalar', int16(123456789))
  - >> tango_print_error_stack
  - >> tango_write_attribute (dev, 'short_scalar', int16(1024))
  - >> tango_read_attribute (dev, 'short_scalar')
TANGO : device : Status

1 device -> 1 status

- behavior = f (internal status)
  - request -> internal status -> execution or exception
  - Internal status run by the device

- 14 predefined status
  - ON, OFF, CLOSE, OPEN, INSERT, EXTRACT, MOVING, STANDBY, FAULT, INIT, RUNNING, ALARM, DISABLE, UNKNOWN
  - known and run by the clients (particularly generic)

Obtain the current status of a device ?

- Prog. env. OO (C++, Java, Python)
  - dev.state ()

- User env. (Matlab, Igor Pro, ...)
  - dev_state = tango_state (dev_name)

- example Matlab
  - >> help tango_state
  - >> tango_state(dev)
  - >> tango_status(dev)
TANGO: properties

**Definition**

- **Configuration Data**
  - concept spread to all TANGO entities
    - attribute, device, classe, system

- **Attribute’s property**
  - 18 properties TANGO predefined + ...
  - ... properties defined by the developer
  - ex: initial value of an attribute

- **Device’s property**
  - specific to the device
  - defined by the developer
  - ex: address GPIB of a peripheral

- **Class property**
  - shared with all the devices of the class
  - defined by the developer
  - ex: URL of the documentation

- **System’s property**
  - shared with all the devices of SC
  - ex: an info related to a centralized service (port nr of the storage service)
TANGO : properties

**Ex : Manipulate the value of a device’s property ?**

- Beware the consequences !
  - initialization of the devices

- Prog. env.. OO (C++, Java, Python)
  - indirect way (cf. TANGO doc)

- User env. (Matlab, Igor Pro, ...)
  - prop_val = tango_get_property (dev_name, prop_name)
  - prop_val_list = tango_get_properties (dev_name, prop_name_list)
  - prop_val = tango_put_property (dev_name, prop_name, prop_val)
  - prop_val_list = tango_put_properties (dev_name, prop_name_list)
  - tango_del_property (dev_name, prop_name)
  - tango_del_properties (dev_name, prop_name_list)

- example Matlab
  - >> help tango_get_property(dev, ‘mthreaded_impl’)
  - >> tango_get_property(dev, ‘mthreaded_impl’)
TANGO : device
TANGO : device : summary

Where to store the properties and all the configuration data?
TANGO: database (static)

Database of the configuration
- critical element of the system
- The only information source for the devices and the clients

Content: 7 tables
- server: infos related to the servers (admin. du SC)
- device: infos related to the devices (IOR=@particularly the network)
- property: global properties associated to SC
- property_class: properties associated to a class of devices
- property_device: properties associated to a particular device
- property_attribute_class: properties associated to an attribute (for any device)
- property_attribute_device: properties associated to an attribute of a particular device

Implementation
- 1 dedicated device = interface TANGO of a SGBD
- TANGO_HOST = host_name:host_port (ex: localhost:20000)
TANGO: device server

Control system SOLEIL
(applications network)

Client Application
Server Application

Admin. SC

Device Server

DServer
Device
TANGO system
≈
{Device Servers {Devices}}
+
Static DB
TANGO: APIs and platforms

- **APIs/Programming Languages**
  - C++ (performances)
  - Java (portability)
  - Python (scripts)
  - Others (Matlab, Igor Pro, LabView)

- **Platforms**
  - Linux
  - Windows NT/2000/XP
  - Sun-Solaris
Questions...
TANGO : Java Tools

- LogViewer
  - Management of messages generated by the devices

- DeviceTree
  - Generic Client : tests, monitoring, ...

- Jive
  - Administration of the TANGO database

- Pogo
  - Code generator (dev. devices)