Tango kernel development

- Tango 9
  - What is implemented (at least in C++)
  - What is not yet implemented
- Tango 10

Tango 8 – Just a reminder

- Breaking change in ZMQ lib release 4
  - Don't use it with Tango 8
  - Supported with Tango 9

Tango 9

- New features approved by the EC
  - Forwarded attribute
  - Enumerated attribute
  - Pipe
- Major release means objects file INcompatible but network compatibility (including events) between all releases.
  - Tango 9: CORBA IDL release 5
  - ESRF machine control system:
    - Device or client using Tango 5(!), 6, 7 and 8

Tango 9 (Extra features from FR)

- Some documentation re-refurbishment
  - Doxygen generated for client AND server
  - Chapter 6 removed from the Tango book
- Dynamic commands in C++
  - DeviceImpl::add_command()
  - DeviceImpl::remove_command()

Tango 9 (Extra feature from FR)

- New event type: DEVICE interface change
  - Allow better management of dynamic command/attribute
  - Sent
    - By DeviceImpl::add_XXX(), DeviceImpl::remove_XXX() methods after a delay (events storm)
    - End of admin device RestartServer and DevRestart commands
    - Re-connection due to DS restart and at subscription time
      - Not sure that interface has changed!
  - Event transport
    - Commands description (command_list_query)
    - Attribute description (attribute_list_query_ex)
    - Boolean data: device started flag

Tango 9 (Extra feature from FR)

- DeviceImpl::is_there_subscriber() method
- DeviceProxy::write_read_attributes method

vector<DeviceAttribute> *DeviceProxy::write_read_attributes(vector<DeviceAttribute> &,vector<string> &);

- Memorized info in attribute configuration
  - With mem_init mode
- Code re-factoring
  - Attribute configuration management and storage in DB
  - Event compatibility

Tango 9 – Forwarded attribute

- Example:
  - Ski-lift in a ski resort
    - Ski-lift motor Tango class with Speed attribute
    - Ski-lift Tango class with MotorSpeed attribute
    - Motor Tango class Speed attribute: the **root attribute**
    - Ski-Lift MotorSpeed attribute: a **forwarded attribute**

- A forwarded attribute forwards to its root attribute
  - Its read / write / write_read requests
  - Its configuration
  - Its event subscription
  - Its locking behavior

Tango 9 – Forwarded attribute

- Association done
  - by property (__root_att) belonging to the forwarded attribute
    - root attribute name (FQAN)
    - Multi control system is supported
  - Root attribute name returned in the forwarded attribute config.

- Attribute configuration:
  - A forwarded attribute forwards its configuration requests to the root attribute
    - Only **label** and **name** stay local
  - The root attribute informs the forwarded attribute of eventual configuration change by events

Tango 9 – Forwarded attribute

- Forwarded attribute polling
  - Forbidden to poll forwarded attribute
    - Poll the root attribute
    - Read request from forwarded attributes get their value from root attribute polling buffer
- Forwarded attribute events
  - Again forwarded to the root attribute
    - The forwarded attribute subscribes to root attribute event.
    - The forwarded attribute changes attribute name before forwarding the event to the client
- When a device with fwd attribute(s) is locked, root device(s) are also locked

Tango 9 – Enumerated attribute

- Tango enumeration:
  - Value always start with 0
  - Values are all consecutive
- Sent on the network using a DevShort data

Tango 9 – Enumerated attribute

- One enumeration label associated to each enum value
  - The number of labels defined the possible enum value
    - 3 labels: value between 0 and 2
  - Two ways to define enum labels:
    - At attribute creation time (Pogo generated code)
    - In user code with a call to attribute *set_properties()* method
- Client retrieves the enumeration labels in the attribute configuration
- A client may change enum labels (like attribute unit, alarms,..) but not their number

Tango 9 – Enumerated attribute

- In a Tango class, you set the attribute value
  - Using a C++ enum
  - Using a DevShort
    - Value is checked according to defined enumeration label number

```
enum class Card: short
{
  NORTH = 0,
  SOUTH,
  EAST,
  WEST
};

vector<string> vs={“NORTH”,”SOUTH”,”EAST”,”WEST”};
att_prop.set_enum_labels(vs);
```

```
enum Card me;

MyDev::read_TheEnum(Attribute &att)
{
  …..
  me = Card::SOUTH;
  att.set_value(&me);
}
```

```
DevShort sh;

MyDev::read_TheEnum(Attribute &att)
{
  ….
  sh = 1;
  att.set_value(&sh);
}
```

Tango 9 – Enumerated attribute

- On a client side
  - Extract/Insert enum attribute value in/from DeviceAttribute with
    - A short (generic client)
    - One C++ enum

```
enum class Card: short
{
   NORTH = 0,
   SOUTH,
   EAST,
   WEST
};
```

```
DeviceAttribute da = dev.read_attribute("TheEnum");
Card ca;
da >> ca;
```

```
DeviceAttribute da = dev.read_attribute("TheEnum");
DevShort sh;
da >> sh;
```

Tango 9 – Device Pipe

- A new communication channel between Tango client and device
- Allow exchange of a **variable set of data from variable data type**
  - Image
  - Scan data
- A Tango 9 device supports
  - State
  - Commands
  - Attributes
  - **Pipes**

Tango 9 – Device Pipe

- A pipe transports data **blob**

- A blob is a set of **data elements**
- Each data element has
  - A name
  - Some data (Only Tango basic data type or array thereof)

- Using a pipe, a device
  - Sends blob dynamically defined
    - It's a one way data channel: From device to client

Tango 9 – Device Pipe

- Each device pipe has a description with
  - Pipe name
  - Label and description
  - Display level
  - Output blob description
- Device communication using pipe
  - Synchronously
  - Using a new event type

Tango 9 – Device Pipe

- Server side: Similar to attribute
  - Kernel provides Pipe class instance(s)
  - Two methods for each pipe
    - *void <pipe_name>(Tango::Pipe &);*
    - *bool is_<pipe_name>_allowed(Tango::Pipe &);*
  - Device send data to the pipe with method
    - *void Pipe::set_value(vector<string> &,T *val,Args ...args);*
    - In case of data element which are arrays
      - *void Pipe::set_pipe_blob_elt_size(int elt_nb,size_t length);*
      - *void Pipe::set_pipe_blob_elt_release(int elt_nb,bool release);*
  - Device pushes data to a pipe for event communication
    - *void DeviceImpl::push_pipe_event(string &pipe_name,.....);*

Tango 9 – Device pipe

```
DevShort sh;
DevDouble db;
vector<DevLong> v_lg;
DevLong64 dl64;
vector<string> v_s={“One”,”Two”};

MyDev::ThePipe(Pipe &pi)
{
  if (first_call == true)
  {
    pi.set_value(v_s,&sh,&db);
  }
  else
  {
    v_s={“First”,”Second”,”Third”};
    v_lg={2,3,4,5};
    pi.set_pipe_blob_elt_size(0,v_lg.size());
    pi.set_value(v_s,&(v_lg[0]),&sh,&dl64);
  }
}
```

Tango 9 - Device Pipe

- Client side
  - Synchronous usage similar to attribute
    - *DevicePipe DeviceProxy::read_pipe(string &pipe_name);*
    - Data blob managed through the DevicePipe class
    - To get data from a blob
      - size_t DevicePipe::get_data_elt_nb();
      - vector<string> DevicePipe::get_data_elt_names();
      - int DevicePipe::get_data_elt_type(int elt_nb); // Also from elt name
      - void DevicePipe::get_data_elt(int elt_nb,T &data); // Also from elt name

Tango 9 – Device pipe

```
Tango::DevicePipe d_pipe;
d_pipe = dev.read_pipe("ThePipe");

size_t nb_elt = d_pipe.get_data_elt_nb();
// vector<string> elt_names = d_pipe.get_data_elt_names();
vector<short> i_vals;
double db_val;

for (size_t loop = 0;loop < nb_elt;loop++)
{
   int elt_type = d_pipe.get_data_elt_type(loop);    // d_pipe.get_data_elt_type("elt_name");

   switch (elt_type)
    {
   case DEVVAR_SHORTARRAY:
      d_pipe.extract_data_elt(loop,i_vals);     // d_pipe.extract_data_elt("elt_name",i_val);
      break;

   case DEV_DOUBLE:
      d_pipe.extract_data_elt(loop,db_val);    // d_pipe.extract_data_elt("elt_name",i_val);
      break;
   }
}
```

Tango 9 – Device Pipe

- Client side (through event)
  - New event type in the DeviceProxy::subscribe_event() method
    - PIPE_EVENT
  - A new push_event() method in the CallBack class
    - *void CallBack::push_event(PipeData *event_data);*
  - The PipeData structure is similar to what we receive for a classical attribute change event
    - DeviceAttribute structure member replaced by a DevicePipe.
- Getting pipe description
    - *PipeInfoList *DeviceProxy::pipe_list_query();*

Tango 10

- Tango full ZMQ
  - Better performance for synchronous / asynchronous call?
    - Performance measurement
      - The two computers:
        - Xeon 3 Ghz – 12 cores – 100 Gbyte mem – 64 bits – Debian 6
        - Network link = 1 Gbit/sec
    - Synchronous request (CORBA - omniORB)

	cmd (µS)	attr (µS)
loop back	40	50
network	85	90

	Bandwidth (MBytes/sec)
loop back	2200
network	107

Tango 10

- Event perf

	Latency
loop back	**240 µS**
network	!!

	Scalar	Spectrum (8 KB)	Image (1 MB)
loop back	92 kHz (**10.8 µS**)	81 kHz	1.5 kHz
network	85 kHz (**12 µS**)	70 kHz	110 Hz

- ZMQ
  - Pros : Event system – Availability on modern platforms / languages – Community
  - Cons: Perf in some cases
    - Help from P. Hintjens

Tango 10 : Tasks

- T1 : Protocol / Serialization system choice / string encoding
  - Object finding in server
  - Allow future evolution
- T2 : Basic design in term of ZMQ socket / threading model
  - Synchronous and asynchronous requests
  - Check performance
- T3 : System compatibility
  - Mixing Tango 10 and Tango < 10 devices in the same CS
- T4 : Tools compatibility
  - DevVarXXXArray, omni_thread, omni_mutex,CORBA::Any,..
- T5 : Implementation
  - Based on C++11 ?

Conclusion

- Tango 9
  - Done at 75 %. Ready for summer
- Tango 10
  - Quite some work (> 1 year)
  - Take care on performance in some cases