An example of interfacing mxCuBE with a non-ESRF control system : the TINE case

Peter Konarev
Andres Pazos

EMBL

TINE

- TINE: Three-fold Integrated Networking Environment
- Control system designed and used at DESY
- Multi-platform
- Multi-protocol
- Multi-Architectrure (multicast capabilities)
- Different API supported: C/C++, Java, Labview, Matlab...
- ... but not Python
- More info at tine.desy.de

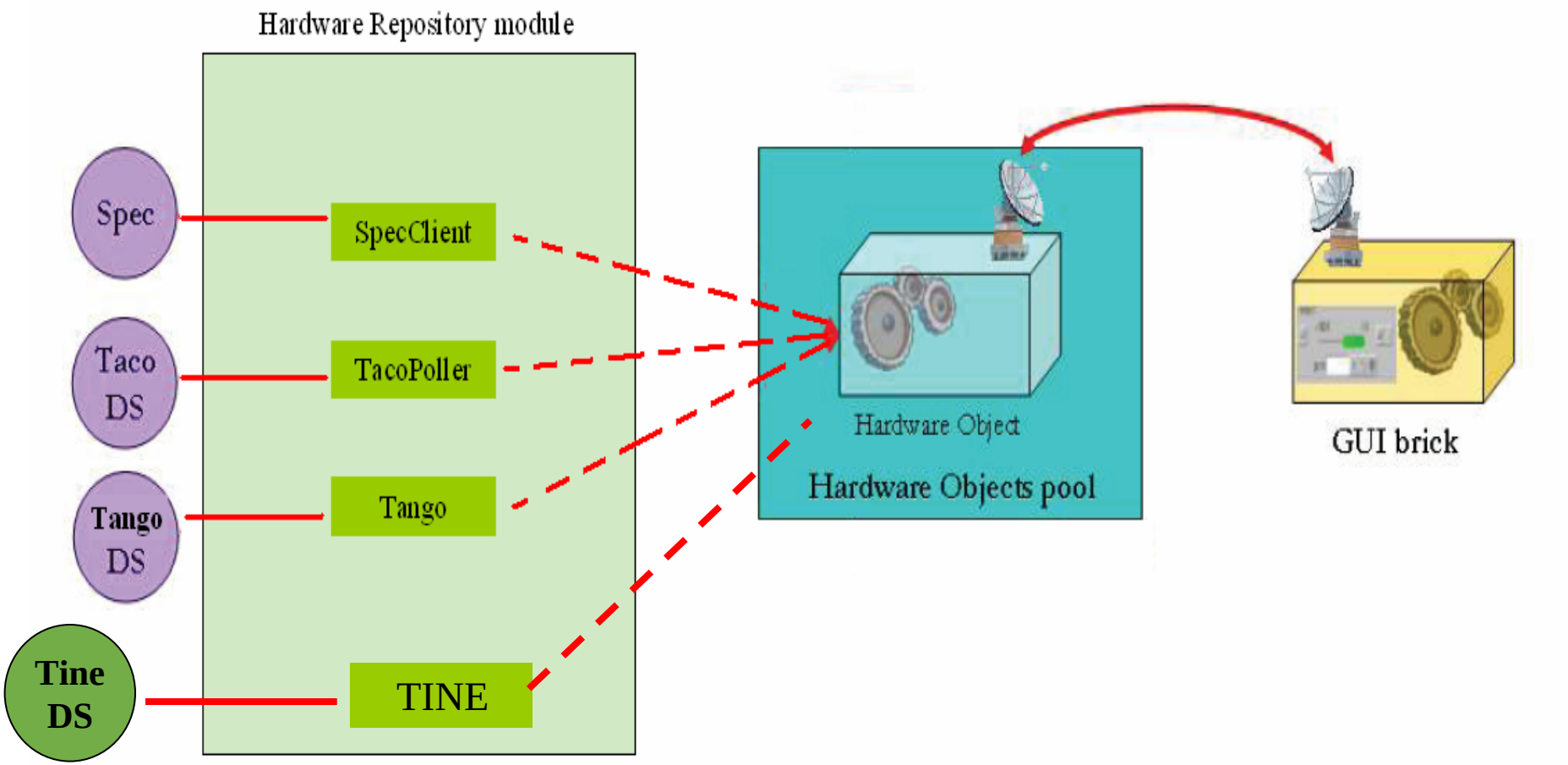
PyTine

- Python bindings for Tine (using Python.h)
- Exporting the functionality of the Tine C library
- Support of synchronous and asynchronous calls
- Support of data structures
- First release available
  - http://adweb.desy.de/mcs/tine/TineArchive/PyTine-0.9.tar.gz

Our Scenario

- Connect mxCuBE with our system (Tine-based)
- Control the data collections from this interface
- Main problems
  - We do not know the system in detail
  - The HO are sometimes dependent of the CS
  - TINE does not support Python directly
- Solutions
  1. Get support from ESRF
  2. Look into the source code and learn about system
  3. Provide a generic connection between TINE and Python: PyTine
  4. Provide a generic connection between the HO and PyTine
  5. Rebuild some of the HO

Our Scenario



Our Approach: Command package

- Command package is the place to put command launchers and channel readers/writers modules
- Command launchers and channels derive from CObject class of the HardwareRepository. CommandContainer module
- The modules are organised by control software (Spec, Taco, Tango) and should emit the appropriate Qt signals.
- In our case we need similar Tine-controlled module that calls Tine through PyTine interface.

Our Approach: Command package - Tine.py

- Implementation of Tine.py inside the HardwareRepository (CommandContainer)
- Follows the same architecture as the Tango.py
- Definition of the connection inside the XML
- The HO access Command & Channels
  - In a generic way
  - Independently of the Control System
- Implemented together with Matias Guijarro

Example1: Hardware Object - Attenuators

Adapted to TINE without code modification

```
import logging

from HardwareRepository.BaseHardwareObjects import Device

class Attenuators(Device):
    def __init__(self, name):
        Device.__init__(self, name)
        self.labels  = []
        self.bits    = []
        self.attno   = 0

    def init(self):

        self.cmdsetTransmission = self.getCommandObject('setTransmission')
        self.cmdsetTransmission.connectSignal('connected', self.connected)
        self.cmdsetTransmission.connectSignal('disconnected', self.disconnected)


        self.chanAttState = self.getChannelObject('attstate')
        self.chanAttState.connectSignal('update', self.attStateChanged)
        self.chanAttFactor = self.getChannelObject('attfactor')
        self.chanAttFactor.connectSignal('update', self.attFactorChanged)
```

Example1: Hardware Object - Attenuators

Needs only modification inside XML

ESRF Configuration XML

```
<device class = "Attenuators">
  <username>Attenuators</username>
  <command type="spec" name="setTransmission">transmission</command>
  <channel type="spec" name="attstate">MATT_STATE</channel>
  <channel type="spec" name="attfactor">ATT_FACTOR</channel>
</device>
```

EMBL-HH Configuration XML

```
<device class = "Attenuators">
  <username>Attenuators</username>
  <command type="tine" name="setTransmission" tinename="DC/Transmission" format="FLOAT"
size="1" >axis_setPosition</command>
  <channel type="tine" name="attstate" tinename="DC/Transmission" format="INTEGER" size="1"
polling="events">axis_status</channel>
  <channel type="tine" name="attfactor" tinename="DC/Transmission" format="FLOAT" size="1"
polling="events">axis_position</channel>
</device>
```

Example2: Hardware Object – Shutter

Needs reimplementation of HO code

EMBL-HH Configuration XML

```
<device class="Shutter">
    <username>Safety Shutter</username>
    <channel type="tine" name="dev_state" tinename="BW7A/DataCollection/"
     format="INTEGER" size="1" polling="events">shutter_status</channel>
    <command type="tine" name="set_in"    tinename="BW7A/DataCollection/"
     format="INTEGER" size="1">shutter_close</command>
    <command type="tine" name="set_out"   tinename="BW7A/DataCollection/"
     format="INTEGER" size="1">shutter_open</command>
</device>
```

ESRF Configuration XML

```
<device class = "Shutter">
  <username>Saf. Shutter</username>
  <taconame>id14/bsh/9</taconame>
  <interval>2000</interval>
</device>
```

Open discussion

- HO should be control system independent
  - Use of commands and channels
  - What happens with different data types and data sizes
  - Ideally we will only want to change the XML config files
- Do we reimplement the control system dependent HO or we keep doing TineHO (example TineShutter)

- Documentation
- How we share our code (under discussion)