

A Multiple Inheritance Mechanism for PyTango Device Classes

Sergi Rubio i Manrique, 21st Tango Meeting, May 2009

Motivations

Inheritance between device classes should allow:

- Fixing, using abstract classes, a common behaviour to a range of classes:

AdlinkAnalogInput Vs. AnalogInput Abstract Class

- Extending an existing class:

ContinuousAnalogInput Vs. SingleAnalogInput

- Customizing an existing class with an specific behaviour or state-machine:

AlbaPLC Vs. PyPLC

Inheritance using Pogo

- Only works when generating the template.
- If the parent class is modified the inheriting classes aren't.
- It's not real inheritance, code and functionality are not inherited.
- Multiple inheritance is not allowed.

Inheritance using Python

- A new Device Class can be declared as inheriting from other class instead of `DeviceImpl`.
- It provides **all** the functionality of the parent class.
- But if new properties/attributes are declared; the parent declarations must be copied/rewritten.
- The same fact prevents multiple inheritance.
- Pogo does not recognize this kind of inheritance, and children classes often are not readable.

Tango Interfaces Proposal

It is an inheritance mechanism focused on:

- Automatic upgrade of existing classes when the abstract/parent class is modified.
- Certain compatibility with Pogo, children devices must be editable and documented.
- Must not be needed to rewrite Parent attributes and commands declarations.
- Multiple inheritance between device classes, providing some device-plugin behaviour.
- Class creation at runtime to manage inheritance using device/class properties.

Implementation

In fact it is a python implementation of Manu's howto:

How to create inheritance link between Tango classes (C++)

http://www.tango-controls.org/howtos/cpp_inheritance

Summarizing:

- It creates a new class inheriting from existing ones.
- It adds all declared attributes, commands and properties to the final class; following the inheritance order and filtered by name.
- Although read/write_attribute and commands are automatically enabled, init_device and hook methods of the inherited class must be explicitly added.

Implementation

- It takes profit of Python dynamicity to modify classes inheritance tree at runtime.
- Multiple inheritance between PyTango classes is added before instance creation:

```
if __name__ == '__main__':  
    ...  
    Dev, DevClass =  
    PyTango_utils.AddTangoInterface((Dev, DevClass),  
    (ParentDev, ParentDevClass), ..)  
    py.add_TgClass(DevClass, Dev, 'Dev')  
    ...
```

Inheritance vs. multi-class server

Pros:

- **Seems the right way for AbstractClasses.**
- Inheritance allows some applications where performance is critical.
- Sometimes a device has more sense if it inherits that if it collaborates (e.g. ContinuousAI vs AnalogAI).
- Inheritance is suitable when a customization of a device is needed, without needing a complete rewrite of the thing.

Cons:

- Knowledge about internal behaviour and members of each class is needed ... **it is not suitable as plugin mechanism.**
- Use of private members must be considered, but it adds additional problems.
- Device creation, State machines and Status management of individual classes is lost; must be rewritten or encapsulated in state_machine() methods.

Pros:

- Knowledge about internal behaviour and members of each class is not needed.
- Commands/Attributes documentation is all that is needed to connect two devices.
- Device creation, State machines and Status are managed independently for each class.
- **Is the right way for connecting two stand-alone device classes.**

Cons:

- All connection/reconnection issues must be solved again for each new device. It forced us to embed **Tau.Core** in some devices.
- Pogo does not document different classes if they are not added as independent modules. This is a problem when classes share source files.

Devices using inheritance:

Devices inheriting from DynDS or Dev4Tango instead of DevImpl:

PyPLC, PySignalSimulator, PLCValve, FS_OTR, PyStateComposer, MKSGaugeController, VarianDUAL, LOCOSplitterBox, AdlinkIODS and recent data-acquisition devices (Oscilloscopes, Spectrum Analyzers and Function Generators).

Customized Devices:

AlbaPLC (PyPLC), PySignalProcessor (PySignalSimulator), PfeifferGaugeController (VacuumController)

Using this method these devices can be documented by Pogo,
... but could Pogo understand inheritance from class declaration?

```
class Locum(AnalogInputDS, SciPyDS, DynDS, DevImpl3):  
    ...
```

Thank you for your attention

*And the work, ideas and suggestions of Tau
Coutinho, Manu Taurel, Ramón Suñé, Alejandro
Homs and Sergi Blanch*