ABSTRACT DEVICE PATTERN

AND TANGO

A.Götz[1], P.Verdier[1], J.Coquet[2], A.Buteau[2], C.Scafuri[3]
[1]*ESRF, Grenoble, France,* [2]*Soleil, Paris, France, Lausanne,* [3]*Elettra, Trieste, Italy*

ABSTRACT

Abstraction gives you the power to take care of the big picture and worry about the details later. One of the main goals of a control system is interfacing to hardware and software. This paper discusses how the Abstract Pattern can be applied to Devices to implement standard interfaces for families of devices, deal with abstraction and generally improve the interoperability of client and server components within and between control systems. This paper presents concrete examples of the Abstract Device Pattern applied to the TANGO control system.

INTRODUCTION

New control systems like old ones need to interface hardware and software subsystems. Object oriented control systems like TANGO offers an excellent opportunity to implement abstraction via standard device classes. The standard device classes implement software interfaces for families of devices e.g. motors, power supplies, function generators etc. The standard interfaces introduce abstraction by decoupling client applications from device implementations. This means more generic code, which in turn means more code sharing. This paper explains how the Abstract Device Pattern has been implemented in TANGO to build families of device servers.

ABSTRACT DEVICE PATTERN

The Abstract Device pattern is derived from the Abstract Factory pattern. Let us remind ourselves what the Abstract pattern is:

> *"Provide an interface for creating families of related or dependent objects without specifying their concrete classes." (Gamma, E., R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995)*

The main idea of the abstract pattern is to provide a common interface to a wide range of concrete implementations. The abstract factory will return an abstract interface to a concrete implementation of the desired object.

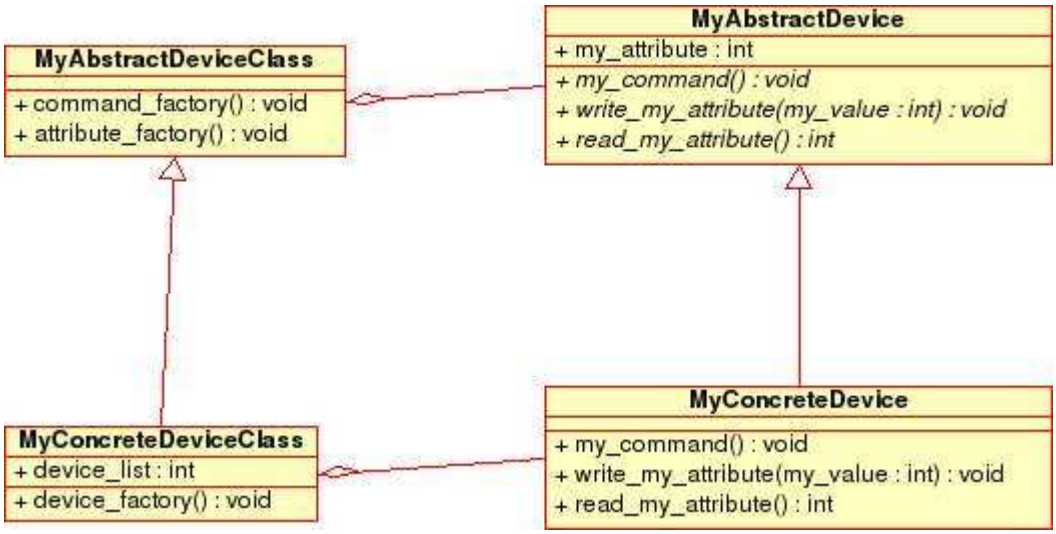
The Abstract Device pattern is the specific case of the Abstract pattern applied to devices. It can be summarised as follows:

> *"Provide an interface for creating families of devices which represent related devices without specifying their concrete implementation e.g. Motor, PowerSupply, FunctionGenerator etc."*

The main motivation of the Abstract Device pattern is:

1.That same client applications can communicate with multiple concrete implementations of related devices

2.To define standard interfaces to a families of devices
3.To avoid duplication of similar but incompatible interfaces for related devices e.g. two power supplies from different suppliers,
4.Guide new programmer's on what the essential methods need to be implemented for a family of devices



HOW TO ABSTRACT A DEVICE?

How to go about defining the abstract device class? This is the first step in defining what constitutes the abstract device for the family of devices which your concrete device belongs to. Often this is also the step where many attempts in the pas have faltered and failed to deliver. The main idea of the Abstract Device pattern is to define the most common interface for all devices which are members of this family of devices. The Abstract Device pattern is in many ways similar to the Union design pattern. The superclass represents an abstract representation of the union of all the subclasses. Due to this polymorphism, the subclasses can thus be used wherever the superclass is required.

When defining the Abstract Device class the following principles can help:

1. include only the main features which best represent the members of the superclass
2. do not get bogged down in detail, if you find the discussion is taking too long stop it
3. if in doubt, throw it out
4. the important thing is to arrive at a result
5. a bad abstract device class will eventually be replaced by a better one

THE CODE

How to go about implementing an abstract device class? The remainder of this paper explains how this has been done for TANGO. The technique described here has integrated in TANGO version 5.

First make sure you have identified which abstract class your device belongs to. Refer to the above section for tips on how to define the abstract class.

THE ABSTRACT DEVICE CLASS

The abstract class will be the super class for all concrete implementations of that device type.

Initially the abstract class will not implement any device specific code – the equivalent of an interface in Java. The reason behind this is to concentrate on defining the abstract class interfaces and not get bogged down in details of what should go in the code. Code should be delegated to the subclasses. Future versions of the abstract class could include code if it considered necessary. The abstract class serves as the super class for the implementation class. It could however at any time be considered as a standard TANGO device server. The abstract class is generated using Pogo – the graphical code generator for TANGO. Generating your abstract class with Pogo you end up with a set of C++ files :

MyAbstractDevice.h, MyAbstractDevice.cpp, MyAbstractDeviceClass.h, MyAbstractDeviceClass.cpp, MyAbstractDeviceStateMachine.cpp

Where MyAbstractDevice will be replaced by the name of your abstract class e.g. PowerSupply. The concrete class which will be derived from this class will link with the include files only. The next step is to implement the commands and attributes in the concrete class. This is also done with Pogo.

THE CONCRETE DEVICE CLASS

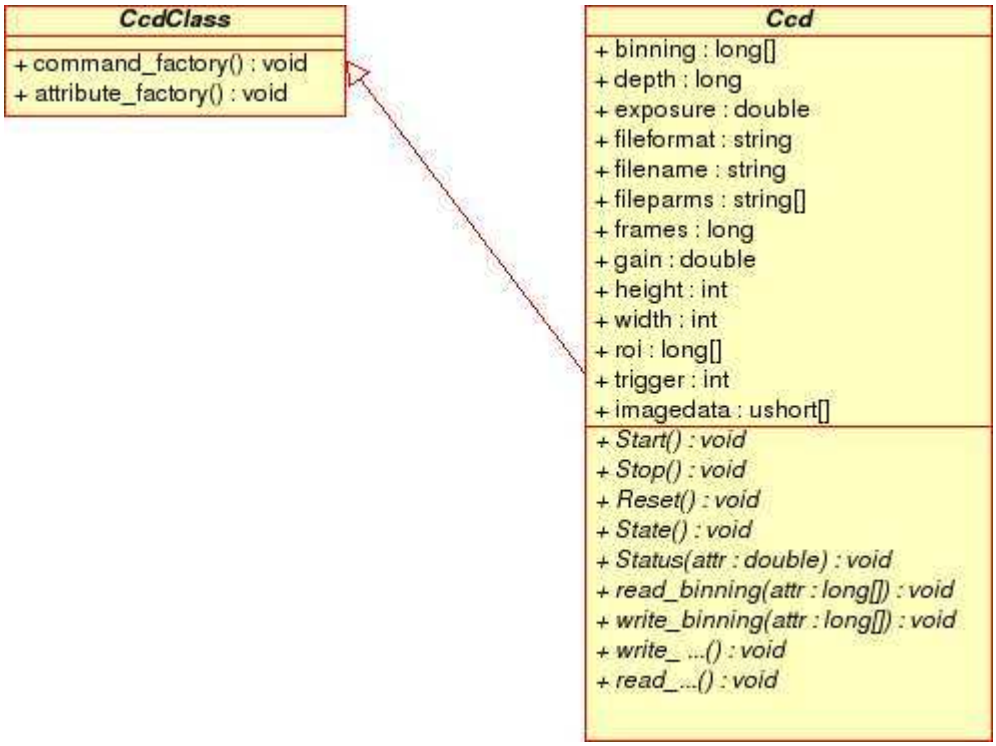
The concrete device class implements a real case of the device i.e. actually talks to real hardware if this is a hardware device server. The concrete device class inherits from of the abstract device class i.e. respectively the MyAbstractDevice and the MyAbstractDeviceClass classes. It is generated using pogo .

POGO

Pogo is the code generator tool developed for generating and maintaining device server classes. It has a repository of all abstract classes which is searched when new classes are created. New classes can be generated as abstract or concrete classes. Here is a screen shot of a CCD camera concrete class for an IEEE 1394 (Firewire) ccd which is derived from the CCD abstract class :

EXAMPLE – CCD

Example of an abstract TANGO interface for CCD's (charge coupled devices) :



This abstract CCD class has been used to build a concrete class for a real ccd camera with an IEEE 1394 interface. The source code for the Ccd for the IEEE1394 camera class can be found on SourceForge in the CVS repository of the tango-ds project:

http://cvs.sourceforge.net/viewcvs.py/tango-ds/Acquisition/Ccd/Ieee1394/

Apple iSight – example of a low cost ccd camera interfaced with Ieee 1394

ABSTRACT CLASSES

In TANGO the following abstract classes have been defined so far :

Motor, Powersupply, SignalGenerator, NeutronDetector, Vacuum Gauge, Ccd, Adc

We are working on the definitions of the following classes inter alia :

Ion Pumps, Pumping Stations, Beam Position Monitors, Counters, Voltmeters, Ammeters, Insertion Devices, Linac, Temperature Controllers, Slits, Attenuators

ABSTRACT CONTROL SYSTEM

Abstraction is part of any good object oriented design. Implementing Abstract Device Patterns for directly identifiable hardware families in a control system are only the first step. Once hardware families are standardised it is possible to group them together and identify even more abstract families of devices which represent a group of lower level device e.g. a linac, a radiofrequency system. Eventually one arrives at a single interface which represents the entire control system. It can be used by high level clients to connect to the control system and browse the concrete instances of the control system.

IMPROVEMENTS

The main improvements which will be added in the future is to extend Pogo so that it can manage multiple hierarchies of abstract classes, implementing shared code in abstract classes as superclasses, and generate Java interfaces.

CONCLUSION

This paper has shown that abstraction is a very useful tool for control systems. The authors think it is still under utilised in most control systems. This is partly due to the lack of support for implementing abstract classes easily. The TANGO control system is one of the few control systems to offer the technology to implement concrete device classes which respect abstract interfaces. Hopefully this will incite more people to follow this route and discuss how we can achieve a common set of interfaces for concrete devices belonging to the same abstract family for TANGO and other control systems.

ACKNOWLEDGEMENTS

The authors would like to acknowledge discussions with the TANGO community especially E.Taurel, J-M.Chaize, J-L.Pons, J.Klora, N.Leclercq and M.Ounsey.

REFERENCES

[1] TANGO home page – http://www.esrf.fr/tango
[2] TANGO device server project on SourceForge – http://tango-ds.sourceforge.net
[3] *Design Patterns: Elements of Reusable Object-Oriented Software.* by E.Gamma, R. Helm, R. Johnson, and J. Vlissides. Reading, MA: Addison-Wesley, 1995