Report

Karen Dreyer Thomas Traub

September 16, 2008

Contents

1 Introduction

1.1 Synchrotron-facilities at DESY

DORIS III DORIS (**Do**ppel **Ri**ng **S**peicher) is a storage ring for electrons which was dedicated for collision experiments. After the building of more powerfull storage rings DORIS was modified to a synchrotron source, now of third generation, which means that radiation is partly produced by bending magnets, undulators and wigglers. With a circumference of 289m it stores the e^- at an energy of 4,45GeV. 33 Beamlines are situated around the ring where the radiation is used for different experiments (e.g. GISAXS, XAFS, EXAFS, flourescence).

PETRA III PETRA (**P**ositron **E**lektron **T**andem **R**ing **A**nlage) is a much bigger ring. It was also designed for collision experiments til HERA (**H**adron **E**lektron **R**ing **A**nlage), an even bigger ring was built up. The PETRA-ring is about 2.3km long. It is currently under reconstruction to serve as a third generation synchrotron source as well. Approxamatly 20 beamlines will be installed in one octant of the ring. The process of renewing and setting up new experiments is supposed to be finished in 2009. PETRA will then deliver even more powerful radiation and the setups will focus the beam at some places to micrometer size. By this it gets harder to adjust the beam to the desired point on the sample. To succeed anyway better microscopes and more precise motor controlling have to be used. Moreover the experiments get more complex and need a lot more detectors. That's why a new communication system between the different devices, called TANGO, is going to be installed which is supposed to make the usage easier and more efficient (see also chapter 2.1).

1.2 Aim of the project

At some PETRA III beamlines a new system will be introduced for controlling the different devices. This system is called Tango (see also chapter 2.1). The idea is to start using Tango on Beamline L to get some experiences with the system and improving the current controlling software.
The aim was to write a program which controls the motors of the sample-table, the ones of the microscope and a motor for changing the magnification of the microscope. Moreover a program should be written to display the camera pictures which also provides methods to save pictures and some more mouse-click-events.

1.3 Experimental setup

Beamline L is dedicated to Hard X-ray Micro-Probe experiments (μ-fluorescence, μ-XANES). With its two different sets of monochromators a multilayer beam with high flux can be produced as well as a beam with very small energy range which can be used for XAFS. Behind the monochromator the beam passes a

collimation system. Moreover different materials can be inserted to weaken the beam. An ionisation chamber in front of the sample can be used to determine the beam intensity.

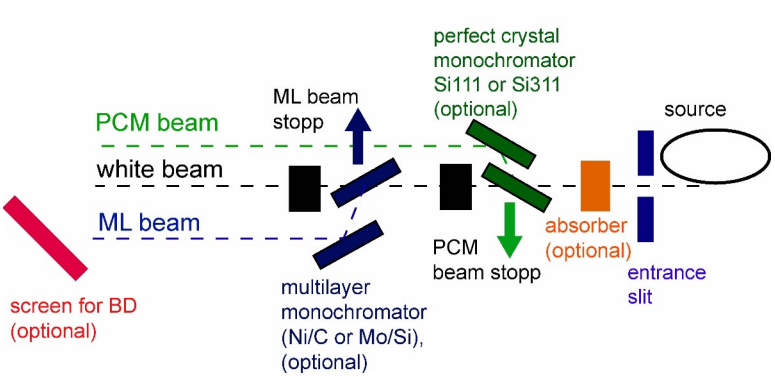

Figure 1: Layout of the experimental setup at Beamline L

The sample is put on a x-y-z-table in an angle of 45° to the incoming beam. With a microscope and an attached camera, which are placed in front of the sample, the sample can be observed on a screen outside the experimental hutch (see figure 2).

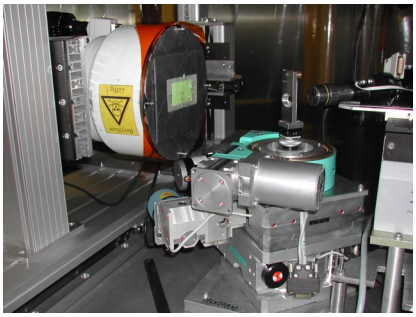

Figure 2: Experimental setup with detector and microscope

For fluorescence experiments the detector is placed at an angle of 90° to the incoming beam, where the background signal produced by inelastic scattering is at its minimum. For absorption experiments a second ionisation chamber behind the experimental table analyses the beam intensity after passing the sample.
Currently the motor controlling is managed by a computer program, called Online. The reading out of the camera for observing the sample is done by a frame grabber which is installed on a second computer. This computer also

serves the saving of images.
All controlling is done by computers which are situated outside the experimental hutch due to radiation protection for the users. All together the hutch contains more than 60 motors which are controlled by online, which was implemented by a DESY employee.

2 Implementation

2.1 Tango

Four our project we used the TANGO Programming Interface. Why do we use TANGO and what is it? For a starting point imagine a typical problem an experimental physicist is facing. You have an experimental set-up. For simplicity let us assume it is a motor and one detector that needs to be controlled. In our modern times we want to use the computer for all our data acquisition and experiment control. Now we are facing some problems. First of all our program will consist of several parts running on different machines. Second, the software to control a detector will be quite different from controlling a motor but they have to communicate with your data acquisition and monitoring software. You might even use different programming languages for different parts e.g. write hardware controlling and software in a machine near language such as C or C++ and your monitoring software which has to care more about displaying information in a more convenient language such as Java or even a scripting language like Python. A first solution would be using CORBA (Common Object Request Broker Architecture). CORBA is an interface which allows different distributed programs to communicate using a common interface. There are bindings to various programming languages thus server and client don't need to be written in the same language. This nicely solves our above mentioned problems, but using CORBA is a quite cumbersome business. The solution is to add another wrapper to CORBA namely TANGO. This interface is rather easy to use and thus helps to speed up the development of very specific experimental control software.

Now after stating our motivation the working principles of TANGO will be shown. Moreover it will be presented how it can be used to build the experimental control software. Since we are talking about a distributed system we need a server client software. A server will control one kind of hardware and for each individual peace of hardware will provide a device that can be accessed by a client.

Let's look at a small example: Your sample is mounted on a 3 axis table. Each axis is adjusted by a single stepping motor, each of them belonging to the same type of motor. In order to position you sample in 3D you must move three motors. All these motors are controlled by a server and this server provides three TANGO-DEVICES each corresponding to a motor to be accessed by the client. We can already see a big advantage: First of all, if your experimental setup consists exclusively of motors of the same type you just need one server

to control each individual device. And second, the server is only developed once and then made available to all users. Everyone using this type of motor can use the provided servers thus eliminating unnecessary programming work.

So far, only the structuring of our software in servers providing devices and clients was concerned. But how does a device connect to a client and how does the communication work? I've already stated that we would like to control different types as hardware such as a motor and a detector in a common way. To do this every device provides us with commands we can send to it and is sporting attributes and properties. We will be mostly concerned with attributes and commands. An attribute can be the current position of a motor or the measurement data of the detector. How these attributes are generated is up to the server so the actual implementation doesn't bother the client. The same principle holds for commands. We could for instance move a motor with a MOVE command or start a camera with START_ACQUISITION command. The server has to provide the implementation to the TANGO-Interface.

2.2 Jive

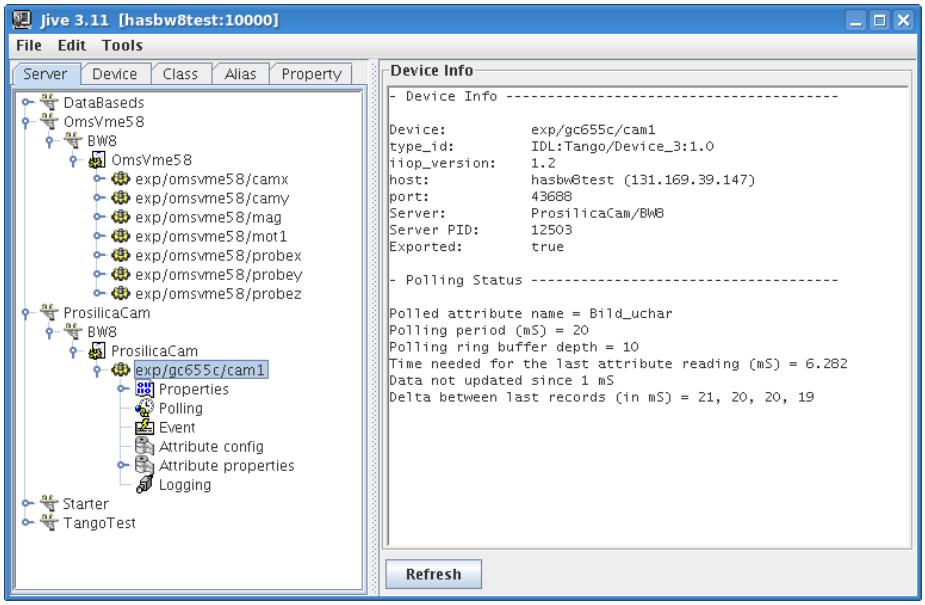

Figure 3: Main window of Jive

Jive is a monitoring tool for the TANGO servers. Each Device can be addressed individually and all its attributes and properties can be accessed by Jive. Via Jive commands can be sent to the devices. It can be used to control

an experimental setup directly or for debugging a client under development. Another main purpose is to configure all devices i.e. setting all needed attribute values e.g. the slew rate or acceleration for a motor or the pixel format for a camera.

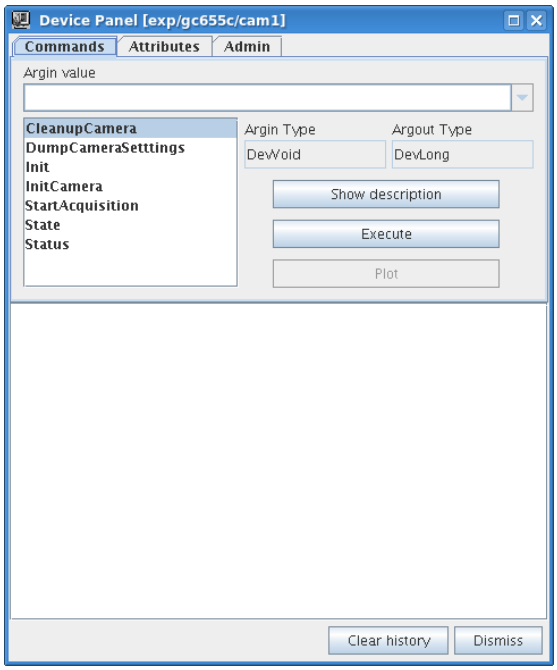

Figure 4: Window for controlling single device

Figure 3 shows the main window of Jive. In the left frame you can see a list of all Servers and the Devices they are providing.
Each device can be configured individually (see Fig. 4). Events can be set and attributes configured. A Device can also be tested. To do this Right click on the device name and choose "Test device". As you can see in Figure 4 this new window allows you to send commands to the device. You are also able to read write all device attributes.

2.3 Client-Examples

Since two summerstudents worked on this project the work was devided. Karen Dreyer was responsible for a general motorcontrolling and a user interface for the table and microscope motors (see chapters 2.3.1 and 2.3.2). Thomas Traub dealed with the gneration of a camera viewer and its imaging functions (see chapter 2.3.3).

2.3.1 Motorcontrolling

For better handling in following programs a motor class was implemented. If a new motor object is created it automatically has all the attributes which are important for a motor like e.g. its position, slew rate or state. These can be changed by the user through getter and setter functions.
Subsequent, the methods for reading and changing the acceleration of the motor are given as an example. The getter methods only reads the attribute "acceleration" and doesn't communicate with the device itself, but only the motor object. This reduces the communication over the network and with this makes the program run faster.

In contrast to this the setter method has to transfer the information x to the device to which value the acceleration should be change (this is done by write_attribute) and afterwards changes the attribute-value of the motor. By this the attribute always contains the actual value of the acceleration.
The ♯-sign declares the line as a command.

```
#Acceleration
    #getter
    def acceleration(self):
        return self.__Acceleration
    #setter
    def change_acceleration(self,x):
        self.__dev.write_attribute("Acceleration",x)
        self.__Acceleration=self.__dev read._attribute("Acceleration").value
    Acceleration=property(acceleration, change_acceleration)
```

2.3.2 Motor interface

In the userinterface of the motor controll the user can choose between either controlling the motors of the table or the microscope. Three switch-buttons serve to choose between up-down, left-right and focus. The motors can be moved by mouse clicks on arrow buttons or by using the arrow keys. An entry field offers the possibility to cross greater distances.
With the motorinterface it is now possible to control 2 motors by the same time. This was not possible in Online and by this time can be saved, especially at the beginning where the motors have to move great distances, because the sample might not be hit by the beam at all.

One problem in implementing the program was to make it possible to use either keys or buttons without disturbing each other. It was tried to use events which can be created by Tango whenever a value changes by a given amount. The server was not implemented without any errors which might be because Tango just starts to get used at DESY. That is why in the end the use of events was cancelled, to make the program usable without any exceptions although it takes some more CPU-time and there might be more elegant ways of programing

it. This can be achieved by further improvement by a programer who is more familier with Python, Tango and PyTango.

2.3.3 Camera viewer

The main goal of the project was the development of a camera viewer that will allow users to watch there sample in real time and adjust its position with respect to the beam position. In the following the different classes of the viewer program will be discussed briefly.

Viewer This class mainly handles the GUI (**g**raphic **u**ser **interface**). One important thing to mention is that the GUI must be updated by the main thread, no multi-threading allowed. This is important since it imposes some restrictions to the implementation. All images read by a image reader thread have to be stored in a queue until the main thread can read them and update the canvas widget. Not complying to this rule may lead to unexpected and rather strange behavior of the whole program.

Legend and legend items This class creates the legend. The legend consist of a text item, an item displaying the current time and a scale. All items are derived from legend item which holds a image object. To create a new item the method draw must be implemented which uses a DrawImage object to create the items image which will then be displayed by the canvas.

Camera and ImageReader These classes handle the communication to the image server. The image reader is a thread continuously reading the camera image converting it and putting it on a image queue. This queue is then read by the main thread. The camera object is used to control the camera server.

3 Summary

In the end of the Summer Student Program 2008 at DESY a motor class was implemented to controll motor devices. Building on this two interfaces for the work with TANGO were implemented. One is for stearing the three motors of the table where the sample is put on and the three motors of the microscope (see Fig. 5). The second interface provides the user with a camera viewer and several functions concerning imaging like changing the magnification by using the motor class and saving pictures, including a scale next to the picture (see Fig. 6).

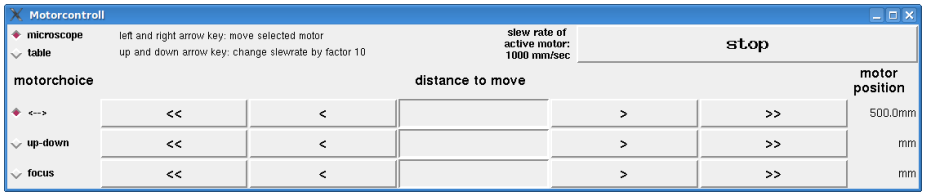

Figure 5: Screenshot of the motorpanel

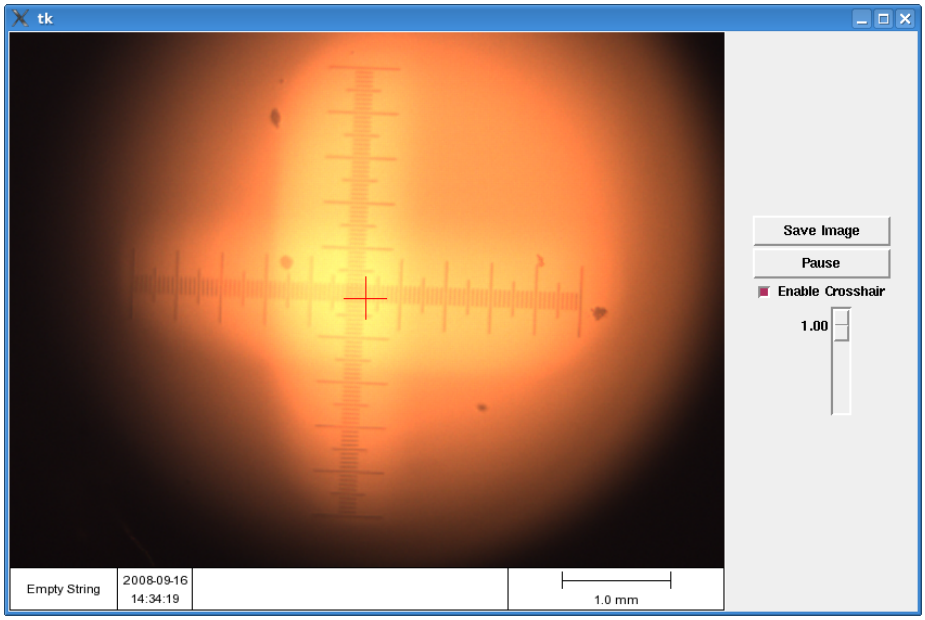

Figure 6: Screenshot of the camera viewer

References

[1] http://www.galileocomputing.de/openbook/python/

[2] www.hasylab.de

[3] www.wikipedia.com