NEUTRON SCATTERING INSTRUMENT CONTROL SYSTEM MODERNIZATION - FRONT-END HARDWARE AND SOFTWARE ADAPTION PROBLEMS

M. Drochner, L. Fleischhauer-Fuss, H. Kleines, M. Wagener, S. v. Waasen
FZ Jülich / ZEA-2, Jülich, Germany

Abstract

When the FRM-2 neutron source went into operation (2002) and many instruments were moved from the closed-down Juelich reactor to the new facility, it was agreed on a choice of front-end hardware and the TACO middleware from ESRF. To keep up with software standards, it was decided recently to switch to TACO's successor - the TANGO control software. For a unified "user experience", new graphical user interface software "NICOS-2" is being developed by the software group at FRM2.

While general semantics of TACO and TANGO don't look very different at a first glance, and adaption of device servers seemed to be straightforward at first, various problems in practical operation were found, due to difference in state handling, timing behavior and error reporting. These problems, and the changes that had to be made to ensure reliable operation again, will be described.

INTRODUCTION

After some different developments turned out to be rather short-lived, partly due to political reasons, partly due to manpower issues, the "NICOS2" instrument control and user interface framework got administrative backing and developer personnel to be further maintained and adopted by neutron scattering instruments at FRM-2. (See [1] and [2] for some historic information.)

For device access, it was decided to abandon TACO and rewrite device servers to use the more modern TANGO framework. Since the basic functionality of TANGO device servers, the remote, synchronous, execution of commands, is quite similar to that of TACO, an easy migration was expected. It was assumed more or less that the old implementation of device server commands just needed to be adapted to a slightly different API syntax.

BOUNDARY CONDITIONS, DEVICE SERVER ARCHITECTURE

A number of device servers, in particular those which handle detectors, have a structure as shown in Fig. 1.

It is often a matter of discussion whether detector data which come as individual events should be histogrammed immediately during data acquisition. It is not subject of this paper to discuss this design decision – it should just be considered that real-time histogramming needs to be done anyway, because instrument operators want a live picture of measured data. If single event data provide no additional advantage, the cost (both in terms of runtime overhead and

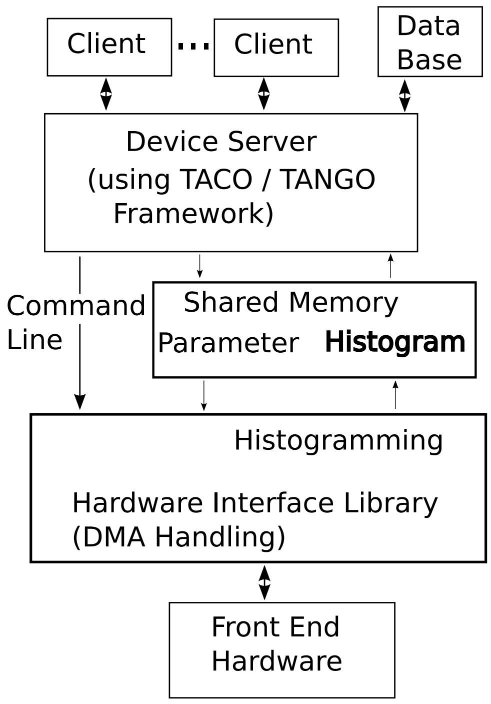


Figure 1: Structure of a detector device server which does real-time histogramming of incoming data.

storage media) can be saved. Up to now, no compelling reason for single event data was found.

Since TACO is not multithreaded from the beginning, and very likely not prepared to run with multiple threads active (actually, it was seen some years ago that even the underlying RPC library was not thread-safe), detector access was put into a separate process which runs asynchronously to TACO command execution. Another advantage of this was that the detector code could be run from a standalone process, for tests when the TACO manager and database services were not available.

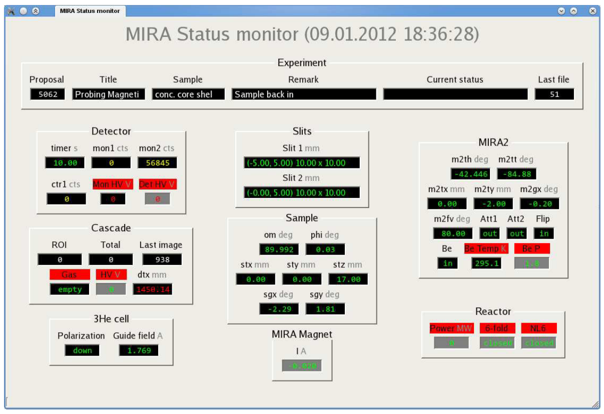
Communication between the server core and the detector access program uses shared memory — keeping a table telling detector parameters (e.g. TOF slot settings) in one direction and the detector data histogram in the other. The shared memory was implemented as a memory-mapped file. This was helpful for debugging because the histogram data could be looked at everytime just using standard UNIX command line tools like od(1)/hexdump(1).

NICOS is a comfortable user interface to control the instrument, define and run measurement scans, show a life display of detector data and support logging and event han-

ISBN 978-3-95450-148-9

dling. Figure 2 shows just a little part of it, a window which shows results of instrument state polling.

This comes at a cost – it scans the instrument state permanently and accesses device servers from multiple threads.



MIRA Status monitor

MIRA Status monitor (09.01.2012 18:36:28)

Experiment

Proposal	Title	Sample	Remark	Current status	Last file
5062	Probing Magneti	conc. core shel	Sample back in		51

Detector

timer s	mon1 cts	mon2 cts
10.00	0	56845
ctr1 cts	Mon HV V	Det HV V
0	0	0

Slits

Slit 1 mm: (-5.00, 5.00) 10.00 x 10.00

Slit 2 mm: (-0.00, 5.00) 10.00 x 10.00

MIRA2

m2th deg	m2tt deg	
-42.446	-84.88	
m2tx mm	m2ty mm	m2gx deg
0.00	-2.00	-0.20

m2fv deg	Att1	Att2	Flip
80.00	out	out	in
Be	Be Temp K	Be P	
in	295.1	1.8	

Cascade

ROI	Total	Last image
0	0	938
Gas	HV V	dtx mm
empty	0	1450.14

Sample

om deg	phi deg	
89.992	0.03	
stx mm	sty mm	stz mm
0.00	0.00	17.00
sgx deg	sgy deg	
-2.29	1.81	

3He cell

Polarization	Guide field A
down	1.769

MIRA Magnet

I A: -0.020

Reactor

Power MW	6-fold	NL6
0	closed	closed

Figure 2: Screenshot of a status display window of the NICOS UI.

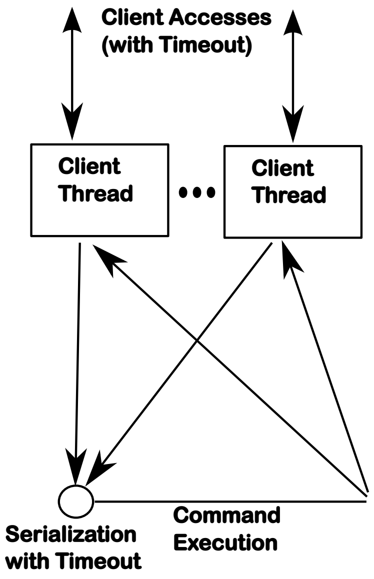


Figure 3: TANGO timeouts for transaction and serialization.

PROBLEMS SEEN, ANALYSIS AND SOLUTIONS

While the new TANGO device servers worked well in tests using just simple clients which do a single action at a time, strange errors occured when the full NICOS interface was used. Commands to start or stop a measurement failed due to timeout for no apparent reason (which is considered fatal), and the background task updating process variables for status display got timeouts as well (which is not fatal but leads to slow UI updates and logged error messages). Bumping the TANGO client timeout, even to unreasonably high values, did not help.

The error messages did indicate a "serialization timeout" in the depth of TANGO. After asking the original developers (see [3]) this led to the explanation that not the command currently executing is to blame for being too slow, but a command started previously by some other client task.

Each TANGO client connection is handled by its own thread. As Fig. 3 illustrates, these threads need to synchronize before hardware is dealt with, or if data global to the device are accessed. The maximum time to acquire this synchronization mutex is fixed to 3 seconds within the TANGO server framework.

Since the status display part of TANGO polls the state of devices permanently, and measurement control is done from another thread, there is a certain likelihood that calls collide and need to be resolved by the synchronization mutex. With TACO (see [4]), there was just a single thread multiplexing all client connections (select(2) in UNIX). An incoming request did stay in the network buffer queue until the server got ready to handle it, without any timeout. As long as the TACO clients had set their transaction timeouts large enough to cover the worst case, no error was reported – just sluggish user interface behaviour was possibly noticed.

This means in effect that no command within the TANGO server can be allowed to take more than 3 seconds, because subsequent commands or data accesses might be issued by another client within that time window. All commands which operate on slow hardware, and which we could just easily deal with before by bumping the TACO timeout need to be split into two halves – one which initiates the action and one which polls for the result.

One might argue that the fixed 3-second timeout can be easily fixed and made adjustable, but, on the other hand, any limit might be too small if a certain number of client threads is in the waiting queue. So it would have been a possible solution to handle errors differently in the clients and allow retries if possible. This would lead to practical problems however because client code is written by different people (most notably the NICOS developers), and it is not always obvious which commands are idempotent or where side effects can occur.

So it was decided to make sure that all TANGO commands are executed within much less than 3 seconds. It has shown that even simple operations like starting (fork(2) in UNIX terms) or stopping the external histogramming process mentioned above can take longer, depending on usage history (whether the program and data needed are already in cache) and operating system background activity. It even happened that the act of zeroing the histogram data array (256MBytes – for 64 channels and 20 bits time resolution) was not done in time after a period of inactivity, obviously because the data were paged out to swap space. The machine these problems were observed on, and where the problems were tracked down, is a Core2Duo with 2GBytes of memory. This is certainly not a powerful system by today's measure, but absolutely appropriate for a front end computer which just has a single task to fulfill.

To be able to finish the TANGO calls in time, command semantics had to be redefined to be asynchronous. The command now just initiates the actions and returns immediately, actual work is done in a background thread. The TANGO server returns some "busy" state until the command is finished. The client needs to poll the state before it issues another command. This kind of side-steps the serialization mechanism described above, but it allows for more flexibility because clients can wait as long as appropriate for command completion.

Unfortunately, we could not add individual "busy" states telling all clients what the server is currently waiting for. The semantics of TANGO states is well-defined and custom values would cause confusion in other parts of the framework.

CONCLUSION

One can't assume that a TANGO installation works like "TACO, just using CORBA instead of Sun-RPC", even if just the subset of TANGO was used which is analogous to functions provided by TACO. The changes in timing behaviour are subtle, but require design changes towards asynchronous operation if any command can possibly take more than three seconds.

This three seconds limit must be met, even if the operating system (which is not aware of real-time requirements) has dedicated memory and CPU ressources to other tasks. On unfortunate occasions, even seemingly innocious acts like clearing some hundreds of megabytes of histogram data hits that time limit.

A standard desktop installation comes with many service tasks which potentially eat up system ressources and slow down intended uses, and there is a tendency that memory requirements grow on each software upgrade. Thus, hardware upgrades need to be done early enough.

REFERENCES

[1] T. Unruh, Instrument Control at the FRM-II using TACO and NICOS, Proceedings of the NOBUGS 2002 conference, 2002, arXiv cond-mat/021043

[2] M. Drochner et al., Adoption of the "PyFRID" Python Framework for Neutron Scattering Instruments, ICALEPCS 2013, San Francisco CA, 2013.

[3] TANGO control system, http://www.tango-controls.org/

[4] TACO control system, http://www.esrf.eu/

ISBN 978-3-95450-148-9