MYST: a comprehensive high-level AO control tool for GeMS

F. Rigaut[a], B. Neichel[a], M. Bec[a], and A.Garcia-Rissmann[b]

[a]Gemini Observatory, c/o AURA, Casilla 603, La Serena, Chile;
[b]Steward Observatory, University of Arizona. 933 N Cherry Ave, Tucson, AZ 85721, USA

ABSTRACT

Myst is the Gemini MCAO System (GeMS) high level control GUI. It is written in yorick, python and C. In this paper, we review the software architecture of Myst and its primary purposes, which are many: Real-time display, high level diagnostics, calibrations, and executor/sequencer of high level actions (closing the loop, coordinating dithers, etc).

Keywords: Multi Conjugated Adaptive Optics, Software architecture, Calibration, Diagnostics, Automation

1. INTRODUCTION

GeMS is the MCAO system for Gemini South, currently undergoing the final stages of integration. It has been described in some details in other publications[1–6] to which we refer the reader for more details. In this paper, we will concentrate on the high level software tools for the control, optimization and diagnostics of the GeMS AO components.

We will first expose and justify our architecture choice, then come to describe this architecture with more details, focusing on three of the main high level (AO) components: the Myst **server**, **client** and **AO simulator**.

There are many more high level software components in GeMS: Modifications have been done at Gemini or by Gemini contractors of the Observing preparation tools (PIT & OT), the Telescope Control System (TCS & TCC), the sequence executor, etc... *We will not describe these in here*; instead we will concentrate, as the title indicate, exclusively on AO related high level software.

2. GEMS AO SOFTWARE ARCHITECTURE

Among the many functions required from a modern Adaptive Optics system, we can clearly isolate two main groups:

1. Real-Time Control, which main requirements are:
   - Fast: High I/O, High processing power
   - Repetitive, time critical tasks
   - Reliable
2. Diagnostics, Smart control, Calibration/Sequences, Initialization, in a word, a high level supervisor. Here, the main requirements are:
   - Flexibility (AO is still a new technique and new tools/algorithms are still emerging)
   - Fast development (simple language accessible by physicists)
   - Good graphical capabilities (for diagnostics)
   - Easy GUI creation (this will be the main point of entry for system operation)
   - No real need for speed (at least no real time speed)

Send correspondence to frigaut@gemini.edu or bneichel@gemini.edu

Obviously, those two sets of requirements are rather disconnected; in fact almost opposite. Thus, will call for different solutions and software components. At Gemini, we have adopted an architecture with:

- The Real-Time Controller (RTC), implemented with dedicated hardware (DSP).[7] Build for High I/O throughput, high computing power & low latency, it does its job very well. However, it is rather dumb, in the sense that it does not compute control matrices, does not take intelligent decision, etc... This is not a shortcoming but a voluntary design choice. It is coded in C++, C and assembly language.

- Myst (Mcao Yorick Smart Tools), the high level supervisor subject of this paper. It is coded almost exclusively in interpreted languages (yorick & python), with some C plugins when heavy processing is needed. All the graphics and AO-specific functions are handled by yorick and yao, an AO library and simulator developed at Gemini since 2001. The GUI capability is provided by PyGTK. Note that for various reasons, we have elected to use only Free/Open Source Software: No licensing restrictions, ability to hack the source code of upstream packages (which we have done many times), reduced impact of upstream possibly dropping support. Note also that all Myst software runs primarily on Linux, but we have been successful in building the Myst clients on OsX (the Myst server should also run on OsX, but as the primary infrastructure at Gemini is Linux-based, this is irrelevant).

This architectural choice is broadly similar to the one adopted by ESO for their AO platform (SPARTA + High level Linux-based supervisor).[8]

Communication between two components is also critical. Data need to be sent from the RTC to the supervisor at high rate. In GeMS, this was implemented as an afterthought -the RTC was specified very early in the project before we finalized the architecture exposed here-, and is not optimized; we pass data RTC $\rightarrow$ Myst (WFS pixels, DM actuator error & commands, TT signal and TTM command, loop gains & leaks, centroid gains, etc) using a socket interface at about 30 to 40 frames/s. This is sufficient for all spatial diagnostics, but does not provide data packets with continuous time sampling necessary for any temporal analysis (this is handled though the saving of circular buffer, a more "heavy" artillery).

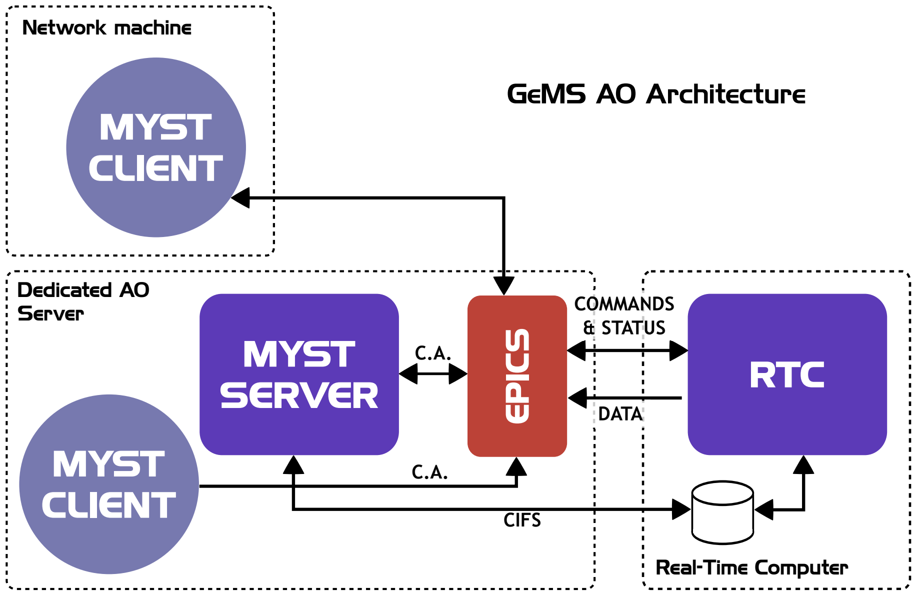


Figure 1. GeMS AO Architecture

Figure 1 present the overall GeMS S/W architecture, including exclusively the AO components. The RTC communicates with the outside world through an epics layer (command & status), a network socket interface for fast data feed and a hard drive located in the RTC (in fact in the windows interface machine) for slow I/O of large files (e.g. upload of control matrices to the RTC or download or circular buffers from the RTC). There is a single instance of the MYST server, and its main purpose is to load initialization matrices/files to the RTC, execute operation sequences (close the loops, etc) and provide high level diagnostics (r_0, Cn^2) and status (Anything not normal? Does the performance look good?).

There can be an arbitrary number of MYST clients, and they can run either locally on a network computer or be remotely displayed from another machine (e.g. our dedicated AO server). Their purpose is to provide a Real-Time display, simple local diagnostics, control (direct or through the MYST server) and relay complex diagnostics and status from the MYST server.

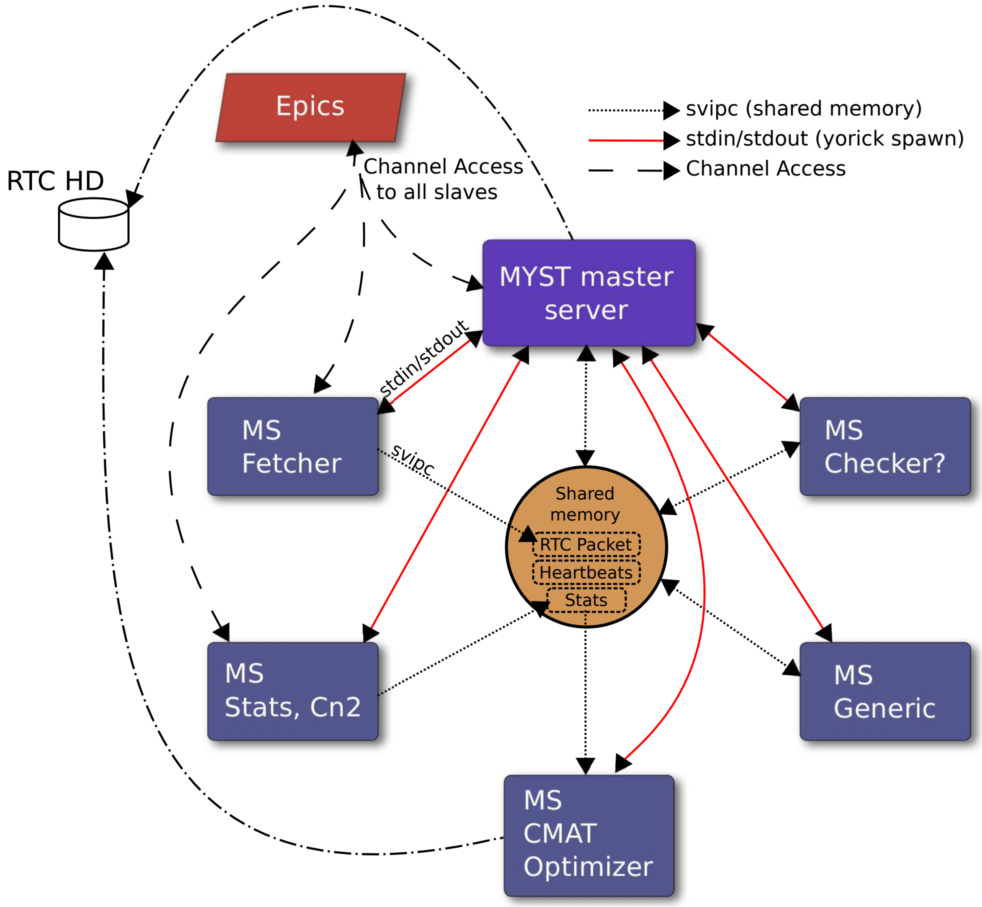


Figure 2. MYST server architecture

3. MYST SERVER

3.1 Requirements

The top level functionality we need from the MYST server are:

- Sequence loop operations
- Gather statistics, publish results
- Compute and apply optimized LGS CMAT
- Compute and apply optimized NGS CMAT (Tip-Tilt & Tilt-anisoplanatism)

- NGS loop & centroid gains optimization
- Smart control of offloads
- Check LUTs
- Smart checks of system health
- Expose only a reduced, simple, foolproof set of APIs to simplify the handling/operation of GeMS by external systems.

3.2 Architecture

Given that the Myst server should stay reactive at all times, and that it has to carry a number of tasks that demand heavy CPU load (e.g. CMAT optimization) or network load (fetching data from the RTC), we had to adopt a multi-threaded, or multi-process architecture. Yorick is not (safely) threadable, but has a convenient spawn capability, which provides easy command and message passing between the parent and children processes (through stdin/stdout for the children and callback functions for the parent). This has the side advantage that it deals naturally with life and death of the various Myst children.

The architecture we selected is presented in Fig. 2. The Myst server is essentially made of one main process (the master/parent server), that takes care –within others– of the main communication with the outside world, and children dedicated to specialized tasks. The parent and the children communicate through several channels: command/status passing is done through stdin/stdout of the child process, which have been DUP'ed to allow the passing of command through a yorick call to spawn(), while a child stdout is connected to a callback function of the parent process. To pass data, we use memory shared between all myst server process. This makes use of a shared memory yorick plugin developed by one of us (MB), based on the systemV IPC APIs.

- The **main** Myst server process (also called parent or master) is the primary point of contact of Myst with the outside world. It also de facto synchronize all jobs, and takes care of relatively light tasks (in the sense that they do not block the main process for long periods of time, being all event based): executing sequences, doing health checks, etc...
- The **fetcher** child process has a single functionality: it fetches data packets from the RTC (through EPICS): 22kB packets at about 30 frames/s, i.e. some 650kB/s.
- The **stats** child process is exclusively dedicated to computing all kind of statistics over the data fetched by the fetcher. It computes the slopes, commands, Zernike projection first and second order statistics, C^2, r_0, etc...
- The **CMAT optimizer** is dedicated to continuously produce optimized LGS and NGS control matrices based on the current conditions. It can take up to one minute to invert a matrix, hence the choice to dedicate one process to this task.

The Myst server is running on our dedicated GeMS server, a 2008 8 cores Linux machine.

3.3 Sequences

The following table lists a subset of the Myst exposed commands. Clients are essentially the TCS/TCC (Telescope Control System) and the sequence executor (centralized component to coordinate science exposure, telescope dither, AO system, and some data saving).

EPICS command	What it does
myst:lgsLoopCtrl	START, STOP, PAUSE, RESUME LGS loop
myst:ttLoopCtrl	Same for NGS TT loop
myst:taLoopCtrl	Same for NGS T.Anisoplanatism loop
myst:ngsConfig	Initialize NGS configuration (coordinates & magnitudes)
myst:slewReset	Reset configuration following telescope slew

There are only 18 commands in total. Other includes enable/disable of M1, M2 and Cassegrain rotator offloads, focus and flexure loops, centroid gains optimization and control matrices optimization. In the following subsection, we give, as illustration, an example of sequence: the sequence executed to close the high order LGS loop. This is allegedly the most complex one.

To prevent operator error and integrate automation in the operation flow, we also implemented some basic logic for sequence execution: All sequences have *dependencies* and *control* attributes.

- *dependencies* are used to list what sequence states the desired sequence *depends on* for it to be allowed to start. A typical example is the NGS TT loop (ttLoopCtrl): Because we want to benefit from the concentration of the NGS flux, it needs the LGS loop closed. So the lgsLoopCtrl is listed as a dependency of the ttLoopCtrl sequence. In the same fashion, the TT M2 Offload sequence depends on the ttLoopCtrl sequence to be started. If a request to start a sequence arrives and the dependencies of this sequence are not verified, the sequence is not started and an error is raised.
- *control* is a list of sequences a given sequence can control. For instance, the lgsLoopCtrl sequence could control the M1 Offload and CMAT Optimization sequences. This goes together with an "auto" flag that can be set individually for each sequence. When a request to start (or stop, pause or resume) a sequence arrives, the sequencer look for all *controlled* subsequences. If the "auto" flag of one of this subsequences is set, then the action is triggered for this subsequence. During commissioning, this should turn out as a convenient feature, allowing us to readily automate the various sequences as hardware functionalities become available (or unavailable).

3.3.1 Sequence example

The following gives the sequence that is used to close the LGS loop:

1. The MYST server receives the "close LGS loop" request from the OCS or from the MYST GUI (EPICS request)
2. As usual, acknowledge the request
3. Switch the LGS WFS exposure time to a "safe" value (say 0.005s = 200Hz)
4. If flag "use internal LGS pointing model" is set:
   (a) Move the PM/CM to the last recorded value at this elevation position (fit of internal model)
   (b) Wait for BTO to be in position
5. Check (averaged over last 5 second) the LGS flux
6. Do we have a recent value ($\leq$2 hours) of the LGS flux?
   (a) If yes, set flux threshold = 0.25 last seen flux
   (b) If not, use a default flux value (100 counts/subapertures)
7. Measure the LGS WFS flux (averaged over 5 second). Is it larger than flux threshold?
   (a) If false, do the following:
      - Offset the LGS (far field, using the BTO PM/CM controls) at (X,Y)=(+2,0) (-2,0) (0,+2) (0,-2), so on a square pattern
      - At each position, record lgs WFS flux (averaged over 0.5 second)
      - At the end, go to the position for which flux is max (possible upgrade: use some kind of fit to extrapolated best position)
      - Go back to "measure the LGS WFS flux"

   (b) If true, proceed

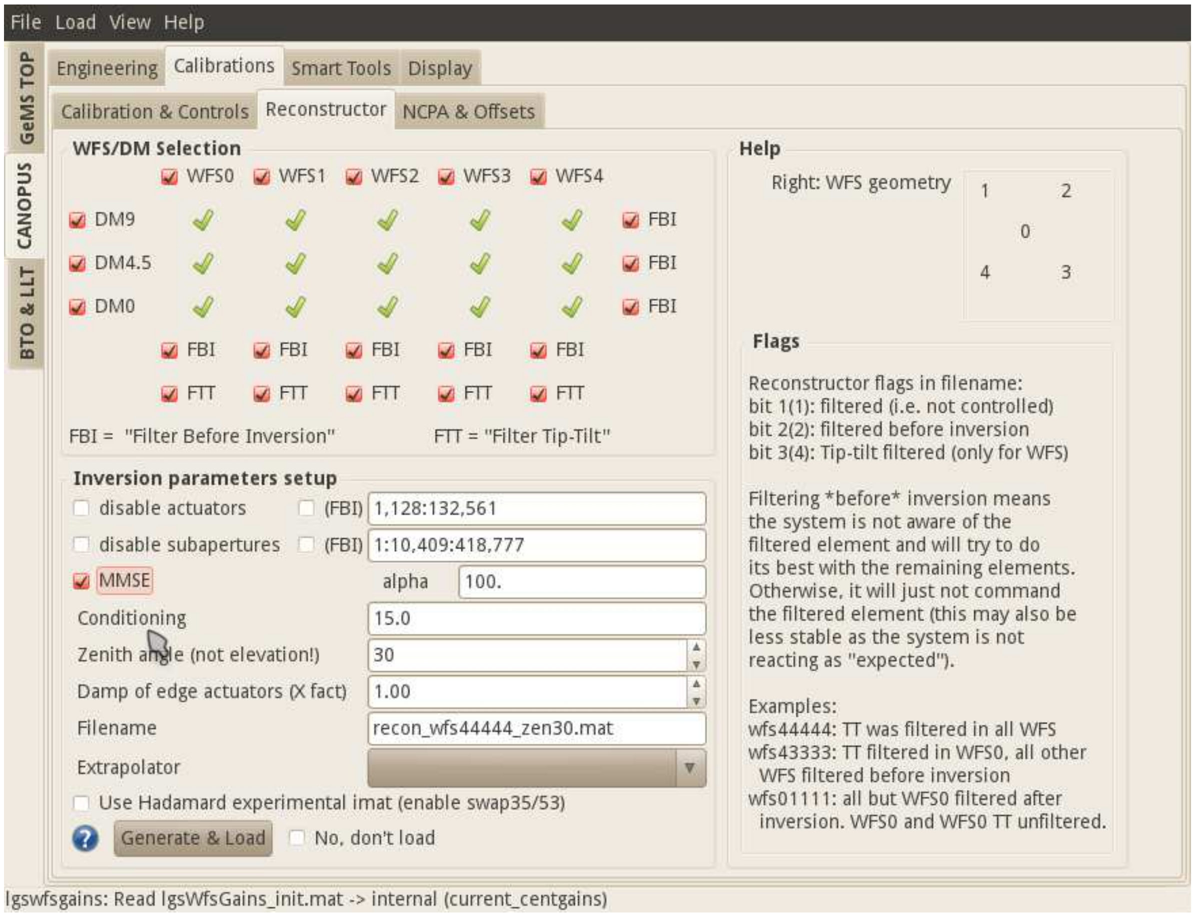
8. Set the LGS loop gain/leak for the initial setting (gain = typically 0.2, leak = not relevant for TTM control operation, typical low value 0.01 is fine)
9. Store state of M1 and M2 offload
10. Disable offloads to M1 and M2 (RTC internal feeds to TCS and reflective memory, not TCS settings)
11. Close the TTM using LGS information (have a cmat slot dedicated to that). This CMAT is a non-standard one with 0 elements everywhere except in the LGS/TTM section (which is normally 0):
    (a) Tell RTC to use CMAT in special slot
    (b) Close LGS loop
12. Start background process: check LGS WFS flux. If flux goes below a certain threshold for more than 1/2 second (parameter), then force the loops open.
13. Recenter TTM by doing N (parameter in myst.conf ,typically 3 to 5) iterations of the following:
    (a) Average TTM position over 1 second (parameter)
    (b) Offload this position to PM/CM through internal MYST model (to be calibrated), with some gain (parameter, typical 0.5)
    (c) During these N iteration, compute average LGS WFS flux (to be used in next step)
    (d) After N iterations, TTM should be roughly centered, proceed to next step.
14. While we did all this, the CMAT optimizer (MYST server slave) has had the time to compute at least one CMAT (=CMAT0) and loaded it to some slot in the RTC.
15. Set regular LGS loop gain/leak (typical gain=0.5/leak=0.01)
16. Make sure the LGS WFS $\rightarrow$ FSA offload is enabled (and FSA $\rightarrow$ PM/CM offload too), unless "auto" is deselected (MYST Smart tool panel)
17. Tell the RTC to use optimized CMAT (CMAT0). The LGS loop is closed, so we're passing from a state of LGS $\rightarrow$ TTM (and no DMs) to LGS $\rightarrow$ DMs (and no TTM)
18. Determine the best LGS WFS exptime from the LGS flux current value (as monitored by background process started above), apply new LGS WFS exposure time (this could be done in 2-3 steps if necessary to ramp up).
19. Enable offload to M1 and M2 (but nothing written as TTM at (0,0)) if it was requested
20. Give the LGS loop READY signal.
21. Report STATE CLOSED
22. Note that the background process continue running to check in real time the state of the LGS loop
23. Another background process is continuously analyzing the loop performance. This background process is the one that sets the performance flag (BAD—MARGINAL—GOOD—OPTIMAL)

This is a good illustration of how complex and intricate such a seemingly simple action as "close the LGS loop" can become if one wants reasonable automation, and it underlines the need for and benefits of a sequencer. Note that this sequence has not yet been implemented in its entirety and certainly not been tested on the sky, so its details are prone to change.

4. MYST CLIENT

The Myst client is intended to provide a GUI for:

1. Report of GeMS status
2. Semi-realtime graphic display of RTC data (see below)
3. Higher level diagnostics as r_0, projection on Zernike modes, etc...
4. Engineering control of the RTC main functionality: close loops, adjust gains, enable offload, enable disturbances, etc...
5. Front-end to the Myst server: Start/stop server features, generate MYST server high level command requests (sequences), etc...
6. Generation of initialization files: Reconstructors, slope offsets, ...



WFS/DM Selection

	✓ WFS0	✓ WFS1	✓ WFS2	✓ WFS3	✓ WFS4	
✓ DM9	✓	✓	✓	✓	✓	✓ FBI
✓ DM4.5	✓	✓	✓	✓	✓	✓ FBI
✓ DM0	✓	✓	✓	✓	✓	✓ FBI
	✓ FBI	✓ FBI	✓ FBI	✓ FBI	✓ FBI	
	✓ FTT	✓ FTT	✓ FTT	✓ FTT	✓ FTT	

FBI = "Filter Before Inversion" FTT = "Filter Tip-Tilt"

Inversion parameters setup

☐ disable actuators	☐ (FBI)	1,128:132,561
☐ disable subapertures	☐ (FBI)	1:10,409:418,777
✓ MMSE	alpha	100.
Conditioning		15.0
Zenith angle (not elevation!)		30
Damp of edge actuators (X fact)		1.00
Filename		recon_wfs44444_zen30.mat
Extrapolator		

☐ Use Hadamard experimental imat (enable swap35/53)

☐ No, don't load

Help

Right: WFS geometry

```
1     2
   0
4     3
```

Flags

Reconstructor flags in filename:
bit 1(1): filtered (i.e. not controlled)
bit 2(2): filtered before inversion
bit 3(4): Tip-tilt filtered (only for WFS)

Filtering *before* inversion means the system is not aware of the filtered element and will try to do its best with the remaining elements. Otherwise, it will just not command the filtered element (this may also be less stable as the system is not reacting as "expected").

Examples:
wfs44444: TT was filtered in all WFS
wfs43333: TT filtered in WFS0, all other WFS filtered before inversion
wfs01111: all but WFS0 filtered after inversion. WFS0 and WFS0 TT unfiltered.

lgswfsgains: Read lgsWfsGains_init.mat -> internal (current_centgains)

Figure 3. Myst client reconstructor panel

4.1 Real-Time Display

As already stated, the RTD achieves update rates of 30 frames/s, if on a good-enough network (data rate about 650kB/s). The RTD has 3 main graphic panels (WFS, DMs, misc). On each one of them, a pull-down menu allow to select the display of a particular quantity. In total, 41 different things can be displayed: WFS pixels, subaperture intensities, slopes (avg, rmsx, rmsy), WFS stats (noise, jitter, flux, r_0 per WFS), NGS WFS configuration, Slow Focus Sensor (our "truth sensor") in full frame or just region of interest, results of the CCD noise analysis, and results of the LGS centroid gains calibrations; DMs errors (slopes $\times$ control matrix), DMs integrated commands, avg and rms of these previous quantities and actuator gains. On the misc panel: NGS WFS APD raw values, LGS WFS slopes histogram, actuators histogram, Zernike coefficients variance and r_0 fit (from either the WFSs, the DMS, or combined), Strehl estimate, Zernike from the WFSs, Zernike from the DMs, r_0/seeing and Strehl ratio estimate history, Cn^2 profile, LFS WFS flux history, TT loop gain and centroid gain history, and various other minor variables. When displaying statistical quantities, one can select to work on a single LGS WFS or on the average of all of them. There are other facilities like for instance the ability to select a single data point in most of the 2D displays (e.g. one actuator or one CCD pixel) and display its value versus time. One can also stop the RTD and look back in time, frame by frame. Data can be saved and reloaded later for further analysis. In brief, what we believe to be a rather a large and complete set of features.

4.2 Calibrations

Most of the calibrations can also be done in an automated fashion from the MYST GUI: LGS WFS CCDs dark and noise, LGS and NGS WFS centroid gains, DM to lenslets registration, DM actuator gains and offset voltages. Every one of these items execute sequences that put the bench in the correct configuration (incl. closing the loop if needed), record and analyze the data and finally display and publish (save in a webpage/wiki) the results.[6]

4.3 Utilities & Control

Myst also counts with a reconstructor panel[5] (see Fig 3), in which the operator can easily create new reconstructors for various situations (mostly an engineering & commissioning tool), and a Non-Common Path Aberrations panel. All of the GUI commands are also directly available at the yorick session prompt (the GUI is accessible and stays reactive while the Myst client is running), so that more complex commands can be invoked/built/combined.

Several Myst use cases are developed in a companion paper.[9]

5. MYST AO SIMULATOR

We chose very early in the project to include none of the AO high level functionality in the RTC: the generation of the control matrices, the parameter optimizers (centroid gains, control law parameters) was to be handled by an outside component. As already stated, this is an obvious choice as it allows to fold in the latest AO theoretical developments, include more realistic effects and more generally to keep the software evolving as the need arises. Our choice naturally went to **yao**[10] (Yorick Adaptive Optics library), an AO simulator under development at Gemini (FR) since 2001. Yao, which is rather versatile and support a variety of WFS (SH, curvature, modal), DM (PZT, curvature, Zernike, Disk Harmonic, KL), control matrices generation (MMSE, TSVD), was forked in 2008 into `yao_mcao` to adapt yao to the GeMS needs and idiosyncrasies without polluting the yao trunk.

Hence little by little, as I&T were progressing, we refined our numerical model of Canopus. We have been using synthetic control matrices from the very beginning of the integration.[4] A couple of years ago, it appeared that it would also be beneficial to actually plugin the numerical simulation tool within the Myst framework. Specifically, this means that we can use the *full functionality* of the Myst server and clients either with the real Canopus hardware or with the simulation tool. The incentives to do so are multifold:

- Provide a reference behavior, i.e. "can we reproduce what we see on the bench?" and give hints on why it is happening. The simulator provides a controlled environment, with which effects can be switched on and off. This allow for quick debugging/understanding of system behavior.
- Test and develop (new) Myst software components in a controlled environment (e.g. Cn2 profiler, etc...)
- Test potentially dangerous things without the risk of breaking any hardware ! In general, run any changes through the simulator first, which provides an additional layer of hardware safety.

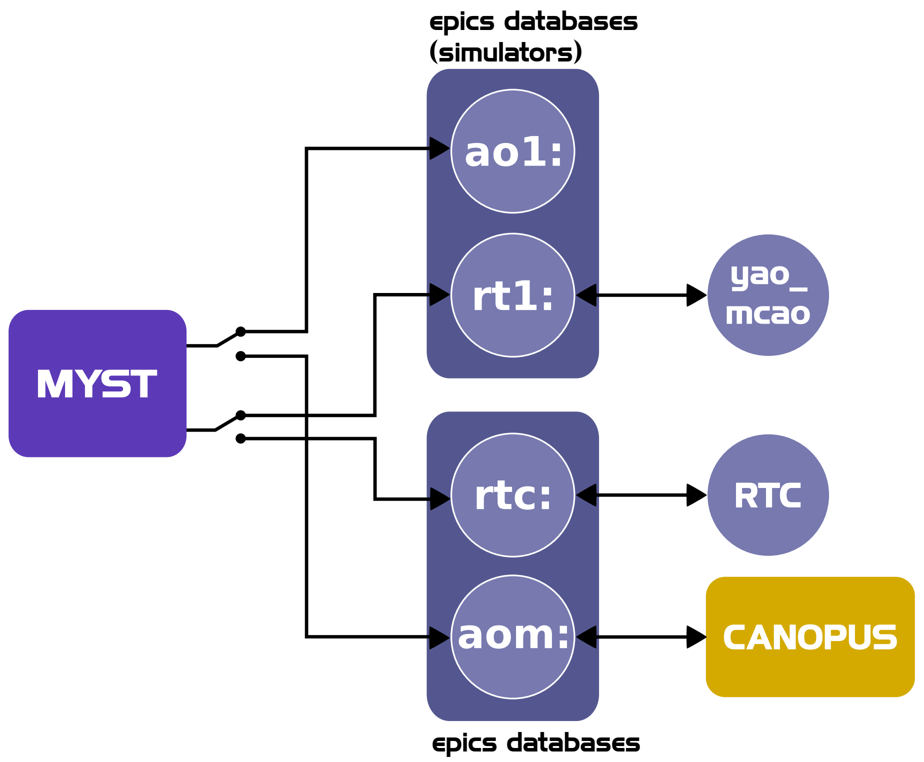


Figure 4. Myst AO simulator architecture

- Produce more reliable control matrices and other initialization entities for the system[4] as we can readily sync the simulation with the real system behavior.
- Last but certainly not least, provide a way to work offline, i.e. when the hardware is not available.

Figure 4 present a schematic of how the AO simulator is used within our Myst framework. The `rtc:` and `aom:` epics databases have been duplicated into `rt1:` and `ao1:`. `rt1:` communicates directly with the `yao_mcao` AO simulator (yorick has a epics channel access plugin), and the simulator write a data stream formatted as the one from the RTC back to `rt1:`. From there, it is just a matter of setting environment variables to have Myst point toward the correct epics databases. Additional simulators can be created by duplicating this scheme (epics databases & `yao_mcao`), and a self contained package can be ran on a single machine (we have one running on a laptop, albeit not at 35 iterations/s, see below).

Recently, we have invested significant effort to make our simulation tools faster. This has obvious generic advantages, but in particular, it allows to have close-to-real-time behavior, which is important when operating the system and looking at the Real-Time Display for system trends. `yao_mcao` was parallelized using the yorick svipc plugin (see above): The 5 WFS and the 3 DM shape computation now run in parallel to the rest of the close loop simulation, the control matrix / vector multiply can run on 2 processes, and the PSF estimation (performance metrics) are also running on a parallel process. All in all, we have been able to increase the speed –without sacrificing the simulation accuracy– by a factor 3.5, for a final number of 35 iteration/seconds. We are running the simulator on our dedicated AO server, which has 8 cores @ 3.33GHz (Xeons) and 8GB of RAM. The CPU usage sums up to a total of about 450% in the conditions reported above (100% means one CPU fully utilized), so that the parallelization overhead is about 30% (4.5× the CPU for a gain of 3.5× in speed).

6. CONCLUSION

Myst is a high level supervisor for GeMS that takes care of the control, diagnostics, calibration, initialization &

sequencing of the CANOPUS real-time computer. It is well decoupled from the pure Real-Time components. It has been in large part developed by AO physicists, for physicists (AO diagnostics/commissioning). It also present simple high level commands and automation for regular use by trained operators.

A relatively large amount of resources ($\approx$ 5 FTEs) have been invested in the development of this package, which we believe offers a flexible framework, while still retaining reliability and maintainability.

The AO simulator has turned out to be extremely useful, especially in the context of the long shutdown period we had recently (Nov2009-July2010).

Although re-use is rarely seen in our research community, we should also mention that even though the implementation is fairly MCAO-specific, Myst could still serve as base for other systems, being relatively easy to hack thanks to the use of interpreted languages.

ACKNOWLEDGMENTS

Part of this work has been founded by the Gemini Observatory. The Gemini Observatory is operated by the Association of Universities for Research in Astronomy, Inc., under a cooperative agreement with the NSF on behalf of the Gemini partnership: the National Science Foundation (United States), the Science and Technology Facilities Council (United Kingdom), the National Research Council (Canada), CONICYT (Chile), the Australian Research Council (Australia), Ministério da Ciência e Tecnologia (Brazil) and Ministerio de Ciencia, Tecnología e Innovación Productiva (Argentina).

REFERENCES

[1] Boccas, M. and et al., F. R., "Gems: Gemini mcao system, current status and commissioning plans," in [*Adaptive Optics Systems*], Hubin, N., Max, C., and Wizinowich, P., eds., *Proc. SPIE* **7015**, 70150X (2008).

[2] Bec, M. and et al., F. R., "The gemini mcao bench: system overview and lab integration," in [*Adaptive Optics Systems*], Hubin, N., Max, C., and Wizinowich, P., eds., *Proc. SPIE* **7015**, 701568 (2008).

[3] D'Orgeville, C. and et al., F. D., "The gemini south mcao laser guide star facility: getting ready for first light," in [*Adaptive Optics Systems*], Hubin, N., Max, C., and Wizinowich, P., eds., *Proc. SPIE* **7015**, 70152P (2008).

[4] Rigaut, F., Neichel, B., Bec, M., Boccas, M., Garcia-Rissmann, A., and Gratadour, D., "A sample of gems calibrations and control schemes," in [*1st AO4ELT conference - Adaptative Optics for Extremely Large Telescopes*], Clenet, Y., Conan, J.-M., Fusco, T., and Rousset, G., eds., *EDP Sciences* **1**, 08001 (2010).

[5] Neichel, B., Rigaut, F., Matthieu, B., and Aurea, G.-R., "Reconstruction strategies for gems," in [*1st AO4ELT conference - Adaptative Optics for Extremely Large Telescopes*], Clenet, Y., Conan, J.-M., Fusco, T., and Rousset, G., eds., *EDP Sciences* **1**, 02010 (2010).

[6] Garcia-Rissmann, A., Rigaut, F., Bec, M., Boccas, M., Galvez, R., Gausachs, G., Gratadour, D., and Neichel, B., "Calibration of the mcao canopus bench," in [*1st AO4ELT conference - Adaptative Optics for Extremely Large Telescopes*], Clenet, Y., Conan, J.-M., Fusco, T., and Rousset, G., eds., *EDP Sciences* **1**, 02012 (2010).

[7] S. L. Browne, R. H. Dueck, G. A. T., "A real-time controller for the multiconjugate adaptive optics system on gemini south," in [*Adaptive Optics Systems*], Hubin, N., Max, C., and Wizinowich, P., eds., *Proc. SPIE* **7015**, 7015–120 (2008).

[8] E.Fedrigo, B.Bauvir, and R.Donaldson, "Sparta roadmap and future challenges," in [*Adaptive Optics Systems II*], Ellerbroek, B. L., Wizinowich, P., Hart, M., and Hubin, N., eds., *Proc. SPIE* **7736**, 7736 (2010).

[9] Neichel, B., Rigaut, F., and et al., M., "The gemini mcao system gems: nearing the end of a lab-story," in [*Adaptive Optics Systems II*], Ellerbroek, B. L., Wizinowich, P., Hart, M., and Hubin, N., eds., *Proc. SPIE* **7736**, 7736 (2010).

[10] http://www.maumae.net/yao.