First Light GREAT and CASIMIR software implementation

S. Stanko, J. Stutzki, January 2003, v0.2

1. Introduction

This document describes the implementation concepts for the GREAT/SOFIA AOS-data-acquisition software, which is most likely also going to be used for CASIMIR/SOFIA with its WASP and DCS backends. It presents the present status of the GREAT planning and development, closely linked to the KOSMA related software efforts. It also starts with discussing and comparing, to the level that we are able to figure out at present, the possibilities to link to the corresponding WASP efforts.

The observing software for **GREAT** (working name: "**KOSMA_control**") will run on Linux-based PCs, which will communicate with the hardware either directly via IO-Ports or via serial connection.

The software will be build up in a modular way with many independent tasks which will communicate and synchronize each other via the **KOSMA_file_io** system. This is a package for distributing information by writing and reading ASCII files. Also a synchronization facility is included.

2. Software and hardware

2.1. Structure of the program package

To ensure maximum flexibility and modularity the software is split into several tasks. Only the lowest layer of the software (**aos_server**) has to communicate with the backend hardware and therefor has to run on the PC which is connected to the backend. All other software layers communicate with this and the other layers using **KOSMA_file_io** over the network and may run on different PCs.

As shown in figure 1, **aos_server** is the only software task which communicates with the AOS hardware. Thus, changing to another backend hardware would mean to replace this small program by another task which is capable to address that particular hardware. In addition, other spectrometers, such as WASP, may require specific calibration task, that are presently not covered in this write-up.

In the layer above the **aos_server** there is the beam switch server (**bs_server**), the on-the-fly server (**otf_server**), the total power server (**tp_server**), and the servers to calibrate the spectrometer (in our example for the frequency calibration it produces combs, so there is a **comb_server**, and for the intensity calibration it has to measure a load, which is done by the **load_server**). All communication in this layer is done via **KOSMA_file_io**.

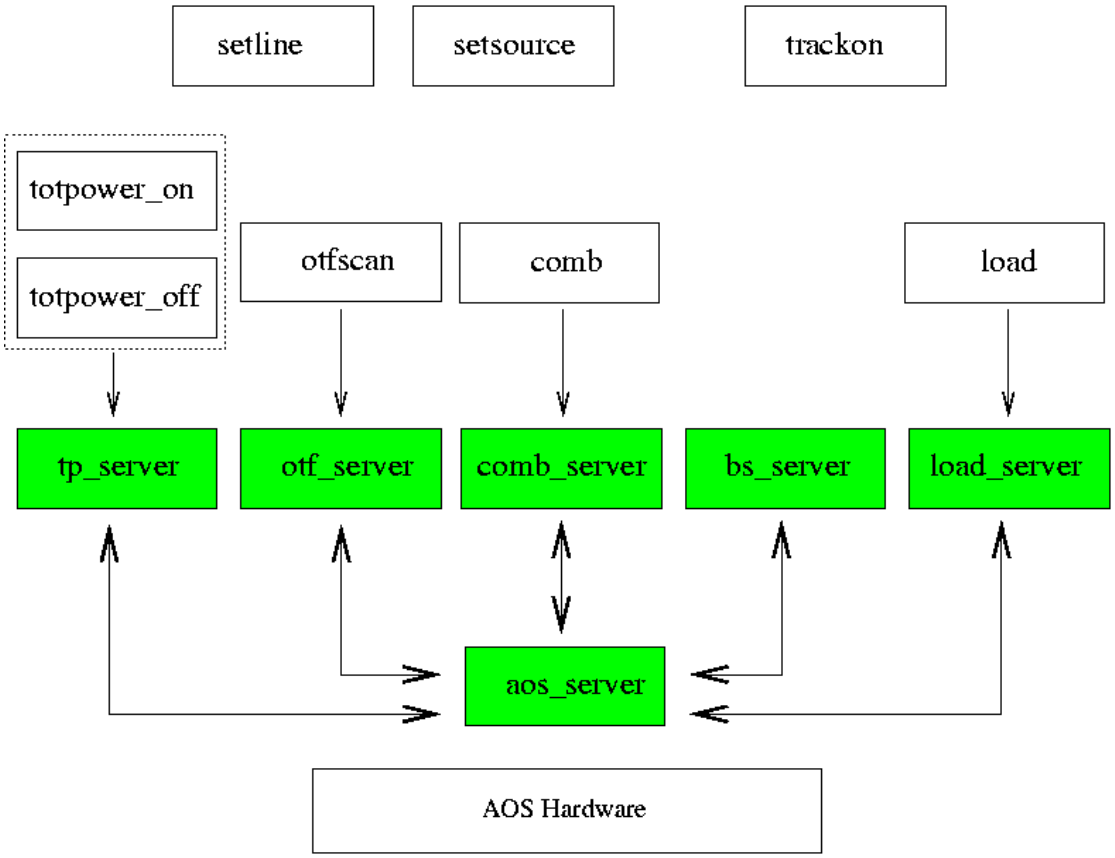

Figure 1: Schematic drawing of the kosma_control software. In this example an AOS is physically connected to the PC and the aos_server task communicates with it.

2.2. Observing modes

The **aos_server** implements the following observing modes:

- total power scans
- signal-reference-switched scans
- on-the-fly scans

All other modes may be implemented as higher level observing programs by using these low level scan programs.[1]

2.3. Other tasks the second layer software performs

The second layer of the software does some additional tasks besides triggering the **aos_server**. Tasks which do measurements on the sky (**totpower**, **otfscan**, **beamswitch**) check the state of the receiver and the sky/load mirrors. After performing the scan they check the tracking of the telescope.

Calibration tasks (**comb**, **load**) switch on and off the comb for calibrating the AOSes and the zero for AOS dark count measurements. In addition they control the load mirrors for temperature calibration.

1 For further details on observing modes see J. Stutzki: “First Light CASIMIR/GREAT observing modes and software implementation”.

In case the telescope system does not support one or more of these features, dummies can be started to simulate these or to prompt information to the user.

2.4. First generation user interface

The first generation user interface will supply information to the software by editing **KOSMA_file_io** variables. Some of these low level interfaces have yet been designed and can easily be connected to a graphical user interface (GUI).

To set the position and line name of the source the **setline** and **setsource** programs are included. To avoid synchronization problems, two different variable spaces are allocated, so the user may change variables for the next measurement which does not affect the measurement which is still running. The program **track** switches between these sets of variables and tracks the telescope. Then the user may start a total power scan with **totpower_on** and **totpower_off** and a on-the-fly scan with **otfscan**. Other programs to perform a **load** or **comb** measurement are also included.

2.5. Data format and online display

The raw data are written as FITS files by the **aos_server**. Thus, it is a good idea (but not necessary) to mount the data disk physically to the computer with the measurement hardware.

There is a quasi online display for the measured data as the data are converted and calibrated by a stand-alone program to class files, which then can be shown on all devices which are supported by the GILDAS software.

All actual system parameters are reachable in the KOSMA_file_io structure, which can be read out by all programs which can handle ASCII files.

2.6. Requirements

Because the software is designed with this high grade of modularity it runs on all Linux based PCs without any special requirements. Only the PC running the aos_server (or the server for another backend device) has to have the desired hardware (except for running in test mode). Even other operating systems should work, as far as they have a GNU gcc (or comparable) compiler and can mount NFS file systems as KOSMA_file_io depends on that.

2.7. Offline testing and debugging

Because all of the hardware communication is included into the **aos_server**, offline testing can be performed by changing the read out functions in this program. All other parts of the telescope system can be simulated by dummies, which communicate via **KOSMA_file_io**.

To implement debugging only the lowest layer of the software system has to be exchanged. Even to build a task which generates dummy data upon given system parameters (such as system temperature or atmospheric transmission) is possible.

2.8. Differences between KOSMA_control and WASP2

Kosma_control is highly modularized by using the possibilities of a modern operating

system such as Linux and its networking possibilities. That leads to many simple and small programs which can be debugged individually on any independent computer of your choice, and not only the observing hardware computer. All programs communicate via the NFS file system. If in later extensions a faster communication is needed, there is a very fast self configurable file distribution system available, compatible with KOSMA_file_io, which guarantees information exchange within 10ms.

Kosma_control will feature a complete test mode even with generation of faked data to verify calibration.

Kosma_control is prepared to perform on-the-fly scans.

To our limited understanding, WASP2 is a relatively monolithic program, which cannot be extended by exchanging small, modular tasks. In addition, the hardware interface is not that simply structured, so that it seems to be a relatively large task to implement another backend hardware into the system.

WASP2 is designed for one rest frequency only. With kosma_control each AOS channel is connected to a parameter defined local oscillator, which defines the measurement frequency.

3. KOSMA_file_io

The KOSMA file I/O library is intended to help exchange information between programs running on one or several computers. In the style of a database system, every variable is associated with its name and some more information about it. Exchange of information between programs is done by writing variables to files and reading them from files. A variable is found within a file exclusively through its name. It does not matter where the variable is located within the file. Thus, it is simple to, for example, edit or restructure the file without affecting the the way how information is exchanged. In addition, information exchange can be monitored by tracing the file contents.

In addition the ASCII database holds information about the time the variables where written the last time. So a reading process can determine whether to read the set of variables or not. That feature can be used to synchronize tasks.

If future developments show that the NFS based file communication is too slow, we already have a test version of a faster file exchange system running, that is fully upward compatible with KOSMA-file-io.

4. Conclusion

From our limited present understanding, which may be biased by not having been able to get the full information on WASP, we arrive at the following judgement, which should be discussed thoroughly between the groups involved (preferentially at a small workshop/meeting). To arrive at a definite conclusion about the GREAT/CASIMIR software implementation is crucial in order to have a well tested system up and operational for first light on SOFIA.

Kosma_control is not yet ready, but seems to be more flexible. Integration of WASP

hardware into kosma_control seems to be easier than integration of AOS-lib into WASP2, because in kosma_control there is only one central program to be changed.