GAS - A General Application Server

H. Delchini [a], J. Gilly [b], J.F. Huppert [a], P. Mora de Freitas [b], M. Punch [c], M. Rivoal [a]

[a] *L.P.N.H.E., Universits Paris VI et Paris VII, 75252 - Paris Cedex 05, France.*

[b] *L.P.N.H.E., Ecole Polytechnique, 91128 - Palaiseau, France.*

[c] *L.P.C., College de France, 75231 - Paris Cedex 05, France.*

When developing the CAT Data Acquisition and Control System, particular attention was paid to highly modular, open, and portable design. This led us to develop and implement the General Application Server (GAS), a portable and general architecture for HEP acquisition systems structured around GAS control and data clients.

Key words: Data acquisition systems; Control systems; Monitoring systems; Networking; Portability; Client-Server model

1 Introduction

The HEP community often deals with an heterogeneous distributed architecture when building data acquisition systems. Typically, a master central mainframe or workstation controls slave processors, which in turn control the various sub-detectors, all connected by the TCP/IP protocol.

The master should be able to start remote tasks on slaves and to send slow control parameters commands in sure way. Remote readout tasks collect data, keep it in local slave data storage or pipe it to the master. Monitoring tasks running anywhere else must access and display sampled collected data to control quality and detector conditions.

To manage this distributed environment, HEP experiments have usually developed proprietary message interprocess communication systems and buffer managers layers ([1], [2], etc). Unfortunately, these proprietary codes are usually too experiment-context embedded and too platform-dependent to meet portability and re-usability goals.

2 CAT's GAS Model

The Čerenkov Imaging Telescope CAT [1] is located in the French Pyrnes and is intended to provide a low threshold for detection of 200 GeV γ-rays with good energy resolution and source-location capability. The physics goals, the apparatus, and the general DAQ architecture of CAT are described in [3]. This experiment has been running successfully since Summer '96.

CAT's DAQ and Control system was build around the General Application Server (GAS) model. In this model, each sub-detector is controlled by a GAS Control client (GCC) and monitored by one or more GAS Data clients (GDC). These clients are connected together via a GAS server (GS) that launches and communicates with the sub-detector's Remote Control task (RCT). The GCC, the GDCs, the GS and the RCT can run anywhere on the net.

[1] "Čerenkov Array at Themis"

Preprint submitted to Elsevier Preprint *3 March 1997*

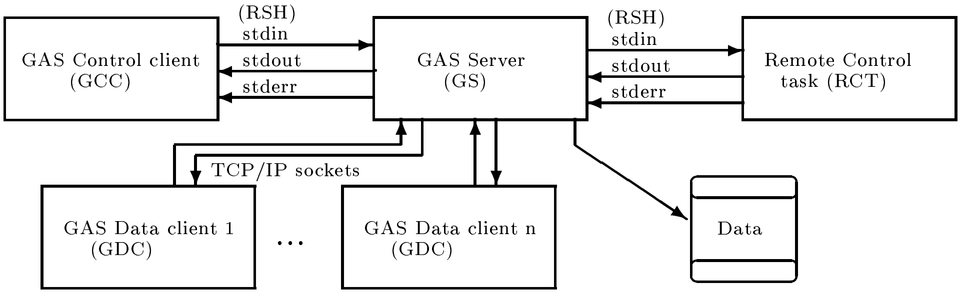

Figure 1: The GAS Model

The GCC on host A, dealing with a particular sub-detector, launches a GS on host B. The GCC takes control of this GS and commands the launch of a RCT on host C. Typically, host C is near the sub-detector hardware, and the RCT is able to control it and collect its data. However, hosts A, B, and C can even be the same machine.

The GCC commands the RCT by sending it statements via the GS. A statement is a printable free-format ASCII stream passed to the RCT, which sends back a free-format acknowledgment. The GS doesn't care about message contents, and simply passes them on. The RCT can also send asynchronous out-of-band messages to the GCC, typically alarms.

If the RCT on host C starts up successfully, the GCC stores in the GS a free-format configuration buffer to be propagated to later GDC connections. Via the GS, GCCs can also send "on-the-fly" messages to GDCs.

GDCs on hosts A, B, C, D... search through the running GSs to find the one which is connected to the desired service. When found, the GDC receives a copy of the configuration buffer from the GS; it is then able to ask this GS to send it a specific type of data samples.

Data records coming from the RCT are free-format binary data except that they must be preceded by a two-word header. The first word holds the record type number and the second the size in bytes of the remaining data to receive. This header allows the GS to receive the data record and to send copies to GDCs that requested the data type. The GS can also save the data on disk if requested to by the GCC.

3 The GAS implementation

To meet the portability, maintainability, and re-usability goals, CAT's GAS model is implemented using standard TCP/IP services and POSIX-compliant code.

Thus, the GAS Control client starts the GAS server using the Remote Shell TCP/IP service and communicates with it by its standard stdin, stdout, and stderr I/O channels. The same applies when the GAS server launches the Remote Control task. The GAS Data clients communicate with the GAS server using standard TCP/IP sockets.

The Remote Control tasks are yacc grammar-interpreters which execute the control statements coming from stdin and which output data to stdout and acknowledgments to stderr. Thus, the control statements are expressed in a readable and easy-to-understand language. This approach facilitates the RCT's stand-alone execution and debugging: the Control task can be launched locally, redirecting outputs to files or filters, and the control statements typed to the input to analyze hardware response and/or data outputs.

The readout tasks are RCT's POSIX-compliant threads. To enable readout, the Remote Control task launches these threads, which output data when triggered. The access to the output channel is a shared resource, so a mutual exclusion is implemented using standard mutex functions.

The VME drivers which have been written implement the POSIX-standard device interface (open, read, ioctl...). The Remote Control tasks access the hardware using standard input/output C-functions, and the particular hardware's parameterization is possible using the ioctl interface.

The GAS server itself is a small C program ($\simeq$1000 lines compiled under gcc) and a C-shell script. In this script the Remote Control task is launched using the standard rsh command, and its outputs are piped to the GAS server input.

An application interface (API) is available for LabVIEW users, providing an easy way to develop portable GAS Control and Data clients with graphic interfaces.

The MasterRC, an additional LabVIEW layer, manages CAT's configurations, starting and commanding the Control and Data clients, depending on the run type. The MasterRC is an automat-model implementation, completely data-driven. The run type and its parameters are described using a simple database system also written in LabVIEW; the MasterRC and its database structure are completely experiment-independent.

4 Conclusions and future

The CAT experiment has validated the GAS model, and this first GAS implementation meets the portability and re-usability goals. It is experiment-independent and two other experiments (Celeste[4] and Thmistocle[5]) are adopting the GAS and the MasterRC layers.

The TCP/IP protocol, remote shell service, standards UNIX tools (lex, yacc, pipes, threads...), and LabVIEW are the key choices to reduce the implementation's effort and to increase portability and maintainability.

The TCP/IP protocol and the gcc compiler should be sufficient GAS server portability to any platform. However the present application interface is limited by LabVIEW's portability, so we are considering a Java API to widen the GAS model.

References

[1] Aleph Message System - http://axaonl.cern.ch:8000/a_doc$prt/ams.ps

[2] J. Milewski, C. Youngman and A. Kotanski, The Zeus message-passing system, proc. CHEP'94, San Francisco (1994), pp.177-181.

[3] M. Punch, proc. International Workshop "Towards a Major Atmospheric Cherenkov Detector IV", Padua (1995), p.356.

[4] J. Quebert et al., proc. International Workshop "Towards a Major Atmospheric Cherenkov Detector IV", Padua (1995), p.148.

[5] A. Djannati-Ata, Mesure du spectre d'mission de la nbuleuse du Crabe en rayons gamma au-del du TeV, par la dtection en lumire Cherenkov des cascades atmosphriques. Doctoral thesis, Collge de France, Paris (1995).