Mem. S.A.It. Suppl. Vol. 3, 372

Memorie della
Supplementi

A new generation control system for astrophysical instruments

R. Cirami[1], M. Comari[1], C. Corte[1], P. Di Marcantonio[1], M. Pucillo[1], P. Santin[1] and C. Vuerli[1]

INAF - Osservatorio Astronomico di Trieste, via G. B. Tiepolo, 11, 34131 Trieste, Italy

Abstract. A new generation control system for telescopes and astrophysical instruments, both by the software and hardware point of view, has been developed and tested at the laboratories of INAF-Astronomical Observatory of Trieste.
In this paper we present a working prototype of such a system: a lightweight, portable, adaptive system, based on the most diffuse standards; such a prototype can be used as a general purpose building block in the design of new instruments.
The software environment is based on Linux, Java and CORBA for the communications among the components of the system.
The hardware has been chosen among COTS components; in particular the prototype presented here runs on a PC104+ platform.

Key words. Control Software – Linux – Real-time – CORBA

1. Introduction

The fast development of control system technology and the need to operate large scale reductions on the resources employed in the design, implementation and maintenance of such systems, strongly pushes towards a new generation of control systems for astrophysical instruments, both by the software and hardware points of view.

The general trend, as can be learned by realizations carried out in most recent years, clearly heads for drives to more cost-effective solutions: wide-spread, stable standards in the software field (programming languages, communications, etc.), COTS (commercial off-the-shelf) components and industry standards in the hardware field.

The spreading of robust, free, open-source software, such as the Linux OS (even the real-time versions), Java and the so-called middleware solutions, like CORBA, greatly enhances the possibility to define clean and effective architectures for those systems, eventually approaching the goal of a solution which adopts a common environment from the high level user interfaces down to the embedded device control. Besides, the availability of faster networks, even of the wireless type, allows the progressive abandoning of mixed, and com-

Send offprint requests to: R. Cirami
Correspondence to: via G.B. Tiepolo 11, 34131 Trieste

plex, communication solutions in favour, again, of a common environment.

All these considerations naturally lead towards a lightweight, portable, adaptive system, based on the most diffuse standards, which shall be used as a general purpose building block in the design of new instruments, or even when refurbishing old ones.

2. Main objectives

Among the main features of the control system we are setting up, the following are of particular relevance:

- the choice of open source technologies allows the control system to be adapted to different hardware platforms
- the self-configuration of the control system requires that each component, whenever added or modified, must behave like “objects”, i.e.:
  - declare its attributes (telemetry parameters, error codes, ...)
  - declare its methods (accepted commands, ...)
  - if required, declare also its operating graphical interface.

The control system in turn must be able to reconfigure itself in order to integrate the newly added capabilities.

This leads to a hierarchical tree-like structure, the leaves beeing the low-level hardware controllers and the root the main control system manager (see Fig. 1).

All information about operations, parameters type, return values and exception declaration of the system components are retrieved at run-time by means of the Dynamic Invocation Interface (DII). This allows the support of arbitrary interfaces, dynamically discovered at run-time without needing to rebuild the whole system.

Information about the configuration of the system components is table-driven. When a new component has to be added all configuration information characterizing it is manually inserted in a disk-resident repository using XML format. As already stated above, such configuration information include telemetry parameters, error codes, accepted commands and operating graphical interfaces. Once the configuration information is added in such repository it is automatically disseminated throughout the whole control system so that any other subsystem is immediately informed about the presence of the new component. In the same way modifications to pre-existing components are automatically notified to the whole control system.

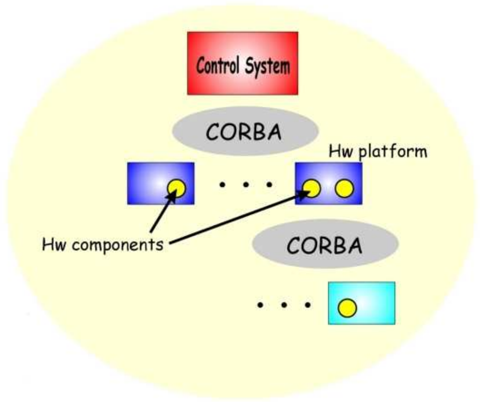

Fig. 1. General Architecture of the Control System.

3. Software solutions

The choice of free, open source software, according to current state-of-the-art technologies has been adopted.

The OS is an embedded Linux, with real-time extensions (RTAI). Such OS has been set up through the following procedure:

- The embedded Linux OS has been created as a restricted version of Linux RedHat 7.3 by copying only those parts of the complete system that are strictly necessary to make it fully operative (device drivers, commands, static and shared libraries, system configuration files and so on).

- The kernel of the embedded system has been updated to version 2.4.19. The kernel sources have been downloaded and compiled directly on the target machine running the full version of Linux RedHat.
- The Real-Time extensions (RTAI) have been successfully applied to the embedded system: the appropriate RTAI patch has been applied to the original kernel sources, the patched kernel has been compiled and installed, finally the RTAI sources have been compiled and installed in turn as a kernel module.

RTAI, once applied to the linux kernel, acts as an intermediate layer on top of the hardware. Such intermediate layer filters all the requests coming from real-time tasks and from the linux kernel itself and destined to the hardware. Because each real-time task has associated a specific priority, requests coming from the various tasks are dispatched to the underlying hardware taking into account the associated task priority. The linux kernel itself is treated as a special real-time task whose priority is the lowest one with respect to those of all other real-time tasks.

High level user interfaces are based on Java. From the client point of view, the use of the Java component model (Java Beans) produces a high level, user friendly graphical interface speeding up development process.

The management of the communication layer between the different components of the system is performed by CORBA based facilities. CORBA middleware provides the necessary infrastructure in exchanging messages among distributed objects, in a platform and language-independent way and in an object oriented context. Through CORBA it is possible to automate many network programming tasks, such as registration, location and activation of objects.

An Ethernet-based point-to-point system allows the network link to be established between the different distributed components of the Control System.

Fig. 2. Control System hardware prototype.

4. Prototype hardware solutions

The choice for a low cost PC compatible hardware environment for embedded applications has been adopted: the PC/104-plus. While the PC, PC/AT and PCI architectures have become extremely popular in both general purpose (desktop) and dedicated (non-desktop) applications, their use in embedded microcomputer applications has been limited due to the large size of standard motherboards and expansion cards and power consumption. So a compact version of the PC, PC/AT and PCI bus has been designed for embedded system applications, the PC/104-plus bus with the following key differences:

- reduction of the form factor to 90 by 96 mm
- elimination of the need for backplanes or card cages, through its self-stacking bus
- minimization of components and power consumption (to typically 12 Watts per module), by reducing the required bus drive on most signals to 4 mA.

The PC/104-plus is backward compatible with the PC/104. The PC/104 specifies two module versions — 8bit and 16bit — which correspond to the PC and PC/AT bus implementations, respectively. Low power consumption means low heat to dissipate and consequently no fans (they usually have limited life), allowing to install the system inside the scientific instruments.

The present prototype is based on a “Digital Logic MICROSPACE PC/104 PLUS MSMP5SEN”, a miniaturized modular device incorporating the major elements of a PC/AT compatible computer:

- for the mass storage unit a 256Mb flash card disk drive connected via the AT-IDE interface (to avoid hard disk weekness such as vibrations and shocks)
- for the motion controller card we used a PMAC2A-PC/104 base board.

We achieved in this way a very powerful and robust real-time control system.

The PMAC2A-PC/104 motion controller is a compact, cheap version of the Delta Tau’s PMAC2 family of controllers. The base board provides four channels of either DAC 10V or pulse and direction command outputs. The PMAC2 is a realtime, multi-tasking computer with its own stored programs and may run as a standalone controller or may be commanded by the host computer, either over a serial port or over the PC/104 bus. Optional boards can be added in a stack configuration.

PMAC can hold up to 256 motion programs at one time. Any coordinate system can run any of these programs at any time, even if another coordinate system is already executing the same program. PMAC can run as many motion programs simultaneously as there are coordinate systems defined on the card (up to 8). The PMAC motion programming language is perhaps best described as a crossing between a high-level computer language like BASIC or Pascal, and ”GCode” (RS-274) machine tool language. These motion programs are not suitable at performing actions that are not directly coordinated with the sequence of motions. For these types of tasks, ”PLC programs” have to be used; they operate in a similar manner as Programmable Logic controllers, continuously scanning through their operations as fast as the processor clock allows. The PLC programs are very useful for any task that is asynchronous to the motion sequences.

5. The PMAC Linux Driver

The version of the PMAC driver available for linux is interrupt-driven. This means that such driver relies on hardware interrupts generated by the PMAC whenever new I/O going to/coming from the PMAC is available. In this case an I/O intermediate buffer is placed between the PMAC card and the user applications so that the PMAC I/O is temporarily kept in this buffer and read/write operations by the user applications interact with this intermediate buffer. Unfortunately the PMAC card model installed on our control system does not generate hardware interrupts (no IRQ available) so the interrupt-driven linux driver has been modified in a way that it is now polling-based. To interface this driver in a proper way user applications must be based on the polling mechanism to check whether new data can be written on the PMAC card and/or output data coming from PMAC are available to be read.

References

PC/104Plus Specification Version 1.2 August 2001 PC/104 Embedded Consortium 1060 North Fourth St. San Jose, CA 95112 Website http://www.pc104.org

PMAC2A-PC/104 Hardware Reference - August, 2002 Delta Tau Data System 21314 Lassen Street Chatsworth, CA 91311 Website http://www.deltatau.com

PMAC2A-PC/104 Installation Manual - January, 2003 Delta Tau Data System 21314 Lassen Street Chatsworth, CA 91311 Website http://www.deltatau.com