Hindawi Publishing Corporation
Advances in Astronomy
Volume 2010, Article ID 560378, 9 pages
doi:10.1155/2010/560378

Research Article

PiMan: System Manager of the "Pi of the Sky" Experiment

Krzysztof Nawrocki,[1] Maria Ptasińska,[2] Marcin Sokołowski,[3] Janusz Użycki,[3] and Marcin Zaremba[2]

[1] *Department of High Energy Physics, The Andrzej Soltan Institute for Nuclear Studies, Hoza 69, 00-681 Warsaw, Poland*
[2] *Faculty of Physics, Warsaw University of Technology, Koszykowa 75, 00-662 Warsaw, Poland*
[3] *Laboratory of Astrophysical Instrumentation, The Andrzej Soltan Institute for Nuclear Studies, Hoza 69, 00-681 Warsaw, Poland*

Correspondence should be addressed to Krzysztof Nawrocki, krzysztof.nawrocki@fuw.edu.pl

Received 30 June 2009; Accepted 20 December 2009

Academic Editor: Taro Kotani

This paper describes PiMan, a CORBA-based system manager of the "Pi of the Sky" experiment, whose prototype consisting of two cameras on parallactic mount is running at Las Campanas Observatory in Chile. We discuss in detail all software components of the "Pi of the Sky" system, interactions between them, and the architecture of the PiMan system. PiMan extension and other system software that will control and monitor final system being under construction, consisting of two sites with 16 CCD cameras installed on three mounts per site, is also described in this paper.

1. Introduction

The prototype of the "Pi of the Sky" [1] experiment, which was custom designed and made, located at Las Campanas Observatory in Chile, searches for rapidly changing optical objects such as Gamma Ray Bursts (GRBs). The prototype system is composed of two CCD cameras located on parallactic mount and operated by a PC equipped with the dedicated and highly specialized software which is divided into modules that correspond to hardware or logical components. The software system controls all aspects of data collection and online data analysis. The PiMan assures communication and interoperation between all modules, coordinates their behaviour, and gives possibility to operate the system automatically and to control it remotely over low bandwidth and/or unstable link. The implemented solution, based on CORBA, is described in the presented paper.

2. Overview of the "Pi of the Sky" Experiment

Perhaps the most powerful cosmic processes ever observed are gamma ray bursts (GRBs) which are 0.1–100 seconds short pulses emitted by extragalactic sources. The energy of the typical burst is estimated to be of the order of 10^{51} erg. The intensity of the burst is often higher than the total background from all other sources in the sky in gamma rays.

In order to proceed with understanding the physics of GRB one needs to observe them in wavelengths different than gamma rays. It is natural to expect that GRB should be accompanied by optical flashes. Systematic study of optical flashes accompanying GRB can impose important limits for theories explaining burst mechanisms and their energy engines. Perhaps not every GRB is accompanied by a bright optical flash.

We propose to build a system consisting of a number of CCD cameras covering as wide field of view as possible. The cameras continuously monitor the sky by taking relatively short (5–10 seconds) exposures. The data are analyzed on-line, in search for optical transients.

The design assumes that a large part of the sky is observed continuously. This is to be achieved by two sets of 16 CCD cameras per site (in the starting phase of the final stage of the experiment, each site will be equipped with 12 cameras. One additional mount per site will be added later), with each camera covering $20^{o} \times 20^{o}$ field of view (FOV).

Each set of 16 cameras will be organized in such a way that 4 cameras will be installed on single mount and each site will have 4 mounts. Cameras in separate sites will be

paired and each pair will observe the same field of the sky, a coincidence of cosmic events in both cameras will be required. Sites will be separated by distance of ~100 km which will allow to reject near Earth objects up to 30000 km orbit thanks to the parallax.

The apparatus is to be controlled and the data are to be analysed by a cluster of PCs. The system will be fully autonomous, but also fully controllable via Internet. During the normal operation the system will run autonomously according to the preprogrammed schedule. Dedicated script language has been developed to make the schedule programming easy and flexible. Every evening script is automatically generated according to satellite pointing information available on WWW. For most of the time the cameras follow the field of view of the SWIFT [2] satellite. When it is not possible to observe any of the satellites, another location in the sky is chosen. The system is also listening to GCN alerts. Should an alert arrive that is located outside the current FOV arrive, the mount automatically moves towards the target and exposures are being taken. Twice a night all sky scanning is performed.

3. "Pi of the Sky" as a Software System

As mentioned earlier the "Pi of the Sky" software consists of several modules that control main hardware components such as cameras and mount. The main modules are the following.

(i) Data AQuisition (DAQ) module: responsible for controlling the cameras, reading data streams and on-line data analysis.

(ii) Mount module: responsible for controlling the paralactic mount.

(iii) GRB Coordinates Network (GCN) module: a module that listens for messages from the GCN network, interprets them, and requests other "Pi of the Sky" modules to react, if necessary.

(iv) Satellite FOV module: this module is responsible for reading information about the current field of view (FOV) of satellites and providing this information to the system.

(v) PiMan module: module responsible for communication and coordination between all other modules.

A more detailed description of PiMan is presented in the following sections.

The overview of the "Pi of the Sky" software system and its interactions is presented in Figure 1.

As can be seen in the Figure 1, PiMan accepts several methods for controlling the system by the system operator:

(i) the PiMan execution shell (Pishell),

(ii) the cron to execute periodic commands,

(iii) scripts for the execution of several of commands (e.g., describing observation program for a given period),

(iv) runscript for execution of scripts from the operating system level (Unix).

PiMan supports a two-way communication with other modules: PiMan can send commands to be executed by modules and modules can send messages to be interpreted by PiMan. The system to perform observations continuously has to be monitored at all levels, starting from low level hardware and ending on all software components. The complexity of the full system consisting of two sites, each one built of 16 CCD cameras installed on 4 paralactic mounts, equipped in many dedicated electronic and software systems, and operated by a cluster of PC computers will grow substantially with respect to the prototype experiment that is operated in Las Campanas Observatory and consisting of two cameras in one mount.

Therefore a new approach to system monitoring fulfilling the following goals:

(i) modularity and scalability,

(ii) operational reliability,

(iii) easy configuration,

(iv) intuitive and interactive Graphical User Interface

had to be worked out.

The system, based on very well-established standards such as CORBA [3], embedded Linux systems, Django [4], AJAX or Nagios [5] and able to fulfill all goals described above, is being presented in the following chapters.

4. PiMan Architecture

Several assumptions had to be taken into account when designing PiMan.

(i) The system had to operate automatically and be controlled remotely over low bandwidth and possibly unstable link.

(ii) The fact that the system consisted of several modules designed by different developers required clearly defined interfaces.

(iii) Because the system serves as a prototype of the bigger system, the commands and messages had to be easy to modify and add.

(iv) In addition, a two-way communication with modules, "special" operation modes (e.g., GCN alert mode) and thread safety was required.

As a consequence of the above conditions well-tested solutions were chosen as system components:

(i) CORBA (MICO [6] implementation) for intermodule communication

(ii) Boost C++ libraries [7] and STL [8] as base for critical components,

(ii) Readline [9] was used as the library for Pishell, the PiMan interactive shell.

The internal structure of PiMan is presented in Figure 2.

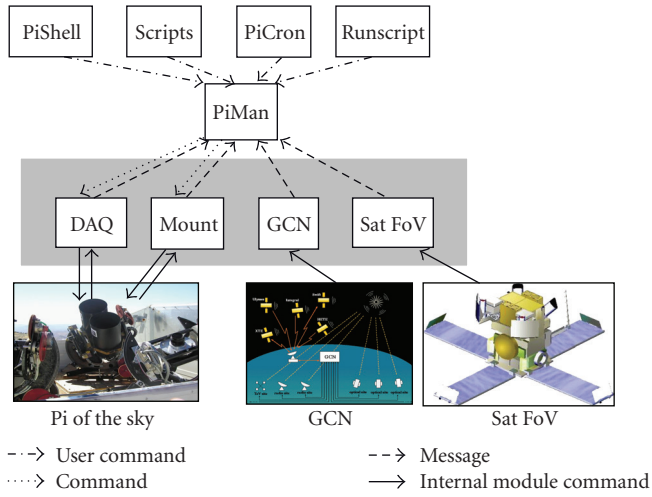

Figure 1: Overview of the "Pi of the Sky" software system.

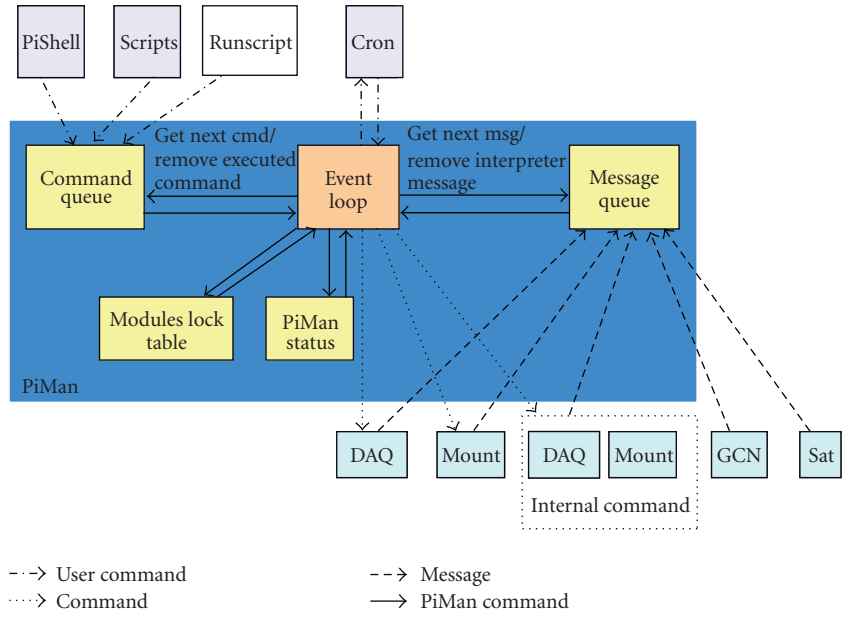

Figure 2: The internal structure of PiMan.

PiMan consists of the following.

(i) A command Queue containing commands to be send to modules for execution.

The syntax for a PiMan command is the following:

`[module_name] [execution_time] [command]`

`([parameters]),`

where: `execution_time` format is `MMddhhmm` and `execution_time` less or equal 0 means "now." Lower (more negative) values are used for higher priority.

(ii) Message Queue containing messages received from modules to be interpreted by PiMan.

(iii) Modules' Locks object containing matrix of modules dependencies (e.g., a module's command locks itself, *internal* command locks DAQ and mount, etc.).

(iv) PiMan Status object that controls PiMan behavior (e.g., enforces mount to automatically follow the SWIFT satellite FOV, enforces PiMan to execute cron, automatically correct mount position from astrometry, sets PiMan to "alert mode", etc.).

(v) Event Loop responsible for reading Command and Messages Queues (in two threads) and sending commands that are ready for execution to respective modules.

There are several special PiMan features. Some of them are the following.

(i) Locking mechanism to avoid sending commands to busy modules.

(ii) "Internal module" used in case a command uses several modules in one PiMan command (e.g., DAQ and Mount). This module locks all modules concerned, until all commands of the internal module are finished.

(iii) Alert Mode that blocks execution of ordinary commands until a command tagged as "forced" arrives. This mode is activated for example when a GCN alert arrives.

(iv) Scripts can be executed in asynchronous mode, when several commands can be executed in parallel, or in synchronous mode, where all commands from a script wait for previous command to finish. For this feature to work a thread_id is assigned to each command. Commands with the same thread_id are executed synchronously.

The Command Queue is implemented as STL's priority queue. Commands sent to PiMan are first checked for syntax validity and then put in the Command Queue. If the execution_time is expressed as MMddhhmm or hhmm (in this case the current night's date is assumed) then it is converted to the corresponding unix_time value. The lower the value the higher the priority assigned to the command. The Event Loop reads a command with the highest priority from the Command Queue and if the command's unix_time is equal or greater than the current unix_time, the current PiMan Status object state does not prohibit its execution. If the requested command does not use a module that currently executes another command then the command is sent for execution.

PiMan State in the current state of the system has the following flags:

(i) ignoreCmds—to run PiMan in the "GCN alert mode" and suppress execution of standard commands,

(ii) manualMode—to give an operator full control of the system by suppressing automatically executed commands,

(iii) updateCron—to reread the PiMan cron file,

(iv) updateConfig—to reread main PiMan config file,

(v) sunIsAboveHorizon and mountIsParked—self-explaining,

(vi) cronPointHete—to update mount position periodically according to HETE FOV,

(vii) autoAG—to decide if the system is to run in the auto-guiding mode, that is, to correct mount movement by data from the astrometry procedure,

(viii) controlCamIdx—to chose main camera for astrometry.

Commands sent for execution are analyzed by the PiMan's command interpreter and are sent to respective modules by calling modules' CORBA methods. Executed commands are removed from the Command Queue.

As can be seen in Figure 2, the cron can be used to execute periodic commands. The Event Loop reads the cron file that contains commands for execution and their repeating interval. The cron execution can be suppressed by setting the PiMan state manualMode on.

Communication from modules to PiMan is implemented as STL's queue. The syntax of a message is the following: [module_name] [message] ([parameters]).

Messages, in contrast to commands, are interpreted instantly. For example, a GCN module can send a message:

gcn trigger (type_of_trigger, trigger_id, trigger_sub_id, tigger_unix_time, trigger_ra, trigger_dec, trigger_likelihood, moving_decision).

Such message is put in the Message Queue, read by the PiMan Event Loop and interpreted. In the case of sending the above message, the system reacts by putting an internal command gcn_trigger in the Command Queue for immediate execution.

PiMan, from the very beginning, was designed to be fully scalable, so migration from the prototype to the final system is straightforward. Every mount with four cameras of the final system will be controlled by one, very well tested in the prototype in Las Campanas Observatory, PiMan instance. As seen in Figure 3, the only new component needed is an additional layer controlling all PiMan instances at one site called PimanMaster (each Piman communicates with PimanMaster using CORBA protocol). This module is responsible for receiving operator's commands, interpreting GRB alerts, and generating scripts for all PiMan modules present at one site.

The only case when communication between DAQ processes bypasses PimanMaster is the checkFlash function (remotely called by CORBA) that is used for comparing flash information with data from other site for satellite and cosmic muon background rejection. Direct communication is used here for performance reasons.

5. External Tools

There are several external tools of the "Pi of the Sky" system which enhance PiMan's functionality:

(i) the generator used to create scripts for a given time period (e.g., one night) to program "Pi of the Sky" system behaviour,

(ii) script checker that can be used to check scripts for syntax errors. The checker dynamically learns the PiMan syntax based on the PiMan sources,

(iii) during the operation of the "Pi of the Sky" a watchdog checks if the whole system works properly. In case of problems a human operator is informed by email or SMS

(iv) the new integrated monitoring system is being developed for the final system.

Below we describe in detail the script generator, which is the system component crucial for the autonomous operation of "Pi of the Sky" experiment and the new monitoring system.

5.1. Script Generator. The script generator is a component responsible for generation of PiMan scripts for a given night. The example of such script is presented at the address http://grb.fuw.edu.pl/pi/papers/piman_script_malaga09.txt. The script generator is launched an hour before the system starts its night work. It retrieves pointing information of INTEGRAL, HETE, and SWIFT satellites available on the Web and programmes mount movements and data collection according to it. The generator calculates times of sunset and sunrise for choosing the time of system start-up, shutdown, and the time for the evening and morning scan of the whole sky.

The generator adds commands to the scripts to perform the following tasks.

(i) Initialize modules—make mount calibration, start cameras cooling immediately after the system startup.

(ii) Start dark frames collection—15 minutes before the Sun sets 10^{o} below horizon.

(iii) Perform evening and morning scan of the sky when the Sun is 15^{o} below horizon. Subscripts scan_evening.pish and scan_morning.pish are generated for this purpose and executed at the proper time in the main script.

(iv) Observe field of view of a satellite that searches for GRBs—SWIFT, HETE, or INTEGRAL, strategy of choosing satellite and position to observe is described in detail later.

(v) Shutdown of the system—5 minutes after the Sun sinks 10^{o} below the horizon.

The system observes only fields from a predefined list of $30^{o} \times 30^{o}$ fields overlapping by 15^{o}. Scan scripts are generated according to odd fields from the list on one day and according to even fields on next day. Normal observations obey FOV of one of the satellites. It is realized in the following way.

(i) The program chooses the celestial position, according to the following algorithm.

  (a) Satellites on the list (currently SWIFT, HETE, and INTEGRAL) are checked if the altitude of their FOV is at least $h_{min} = 30^{o}$ above the horizon and in such case the position is chosen.

  (b) Otherwise, the satellite with the biggest part of FOV above horizon is chosen, but it is preferable that the satellite FOV is rising rather than setting.

  (c) In case it is not possible to find any field according to the above rules the generator can choose an alternative object (like Large Magellanic Cloud (LMC) or Small Magellanic Cloud (SMC)) or choose a position above the horizon in place where soon one of the satellites will appear after rising above the horizon.

  (d) The chosen field is checked against the calculated position of the Moon and in case it is closer then the minimal distance allowed (depends on the Moon phase, for full Moon it is equal to 30^{o}) it is rejected and another object and position must be determined.

  (e) There are several options for observing any object one wanted. For example, it is possible to define favorite fields which will be followed always when above horizon, define favorite fields to follow in case they overlap with the FOV of the satellite selected to follow or define position and time at which it should be observed.

(ii) The program finds the closest field from the predefined list, adds a command to follow this field.

(iii) The program calculates the time when the last followed field will be lower then 30^{o} and a new position must be determined for this moment.

In case of any problems with script generation, for example due to unavailability of pointing information an email describing the problem is generated and sent to a human system operator.

5.2. Integrated Monitoring and Control System. This part of the paper describes the architecture of the monitoring and control system of all components of the "Pi of the Sky" experiment. The experiment is totally remotely controlled so all information about its state (both hardware and software components) has to be collected. One of the main goal of the monitoring system is to present the state of the experiment in user-friendly and intuitive way and to detect and solve emerging problems automatically using an artificial intelligence system if possible or alert operators otherwise.

5.2.1. Low Level Hardware Monitoring. The low level hardware monitoring plays a fundamental role in keeping the detector hardware in an operational state. Due to the fact that whole detector has been constructed as modular and scalable system, the low level hardware monitoring part should reflect the detector architecture. Therefore a modular hardware called "PiMon" for monitoring any physical quantity has been developed.

The core of the "PiMon" hardware called "PiMon-d" module is based on ARM microcontroller with embedded operational system.

Additional measurement modules for voltage ("PiMon-k" and "PiMon-m"), temperature ("PiMon-t") and cooling fan rotation speed ("PiMon-w") monitoring are connected to main digital modules; see Figure 4.

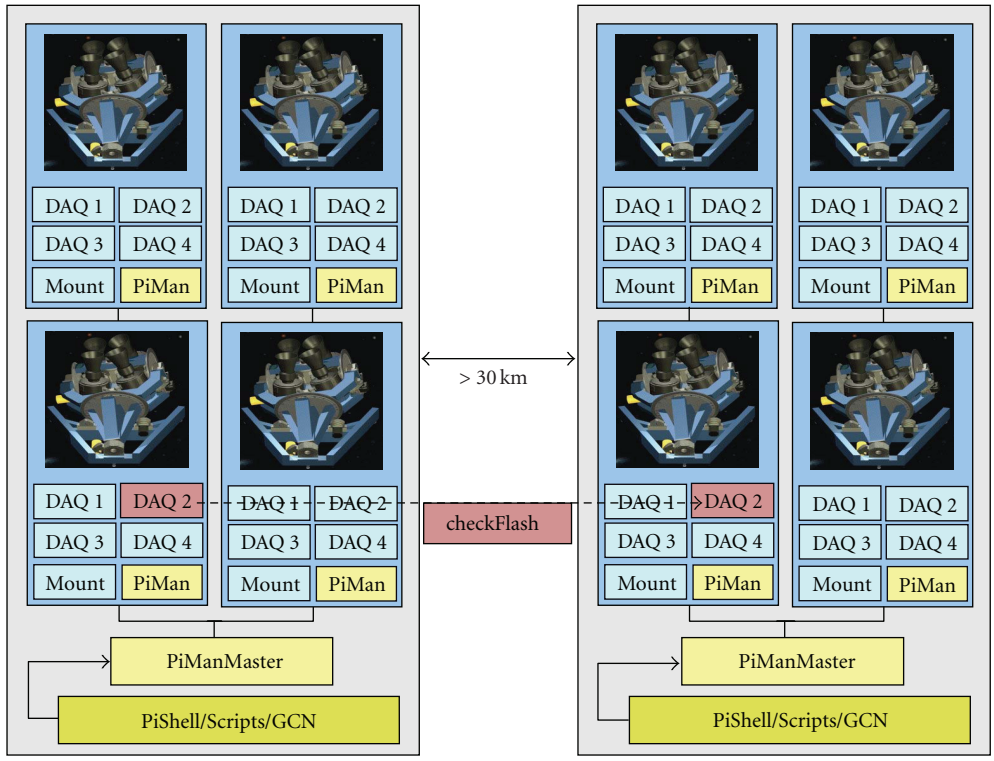

Figure 3: Software architecture of the final system. A new function, checkFlash, described in the text is shown in red.

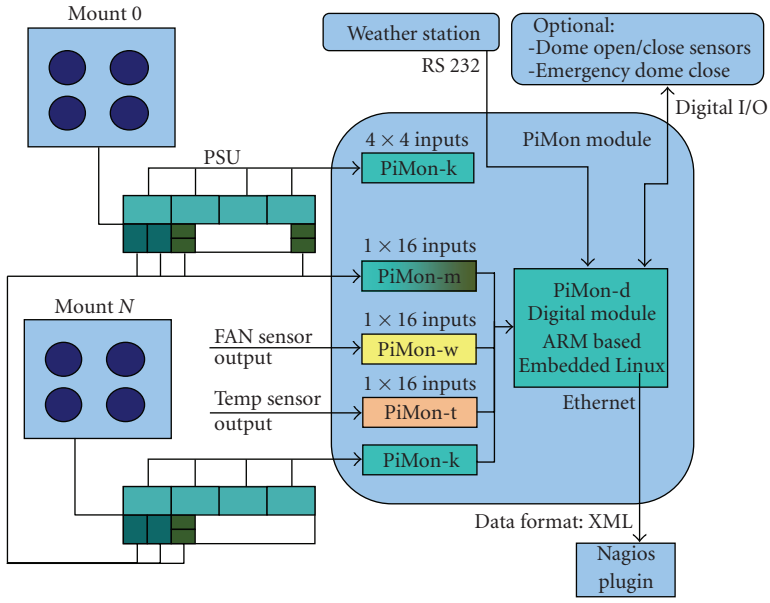

Figure 4: Low level hardware monitoring architecture—"PiMon" hardware.

As "PiMon-d" a common, the ready-to-use microprocessor system based on AT91SAM9260 micro-controller [10] and equipped with necessary operational memory (SDRAM) and data storage memory (NAND Flash) was used. Modularity was achieved by adapting the module to well-defined mechanical standard with an electrical interface. For this purpose VME mechanical standard was used. The communication with the central control system was implemented using an Ethernet interface (with possible additional communication through USB as backup). Additional two RS232 ports are used for debugging/maintenance purposes and for communication with the weather station.

Each measurement module sends data on demand to the master ("PiMon-d" module) through SPI bus. The maximal

number of modules is 255 irrespective of its type. The type of the measurement module defines the physical quantity being measured. In case of "Pi of the sky" hardware three types are needed to have full information about its status.
These are

(i) "PiMon-k" and "PiMon-m" for cameras' and mounts' power supplies monitoring,

(ii) "PiMon-t" for temperature monitoring in critical points,

(iii) "PiMon-w" for cooling fan rotation speed measurement.

"PiMon-k" and "PiMon-m" have 16 independent differential inputs for voltage measurement with low noise amplifier and low-pass filter at each input to avoid ground loops. Each module includes two 12 bit AD converters with SPI interface to communicate with the digital module. The measurement resolution and range of each input channel depend on the amplification of the input stage selected during assembly (e.g., from 3 mV for 0–12 V range to 7 mV for 0–28 V range). An additional rectifier for AC measurement is required. One "PiMon-k" module can measure voltage from 4 cameras PSU (4×4 inputs–4×1 DC and 4×3 AC) therefore one module per one mount is required. One "PiMon-m" module can measure all voltages from mount engines, encoders, and eth/can converters PSU (16 DC inputs) therefore one module per one "site" is required.

5.2.2. Software Monitoring and Data Exchange. The main software is based on OpenWRT Linux [11] distribution designed for embedded devices. Each measurement input module has a different communication set of messages sent through SPI bus being defined in the particular software modules. The main program reads out measured data from each module and stores them in memory. The refresh rate of data depends on the input modules count.

Stored data can be read out by the central system through Ethernet interface using TCP/IP protocol. The special module written for embedded Linux sends data on demand in XML format. Measurement data were grouped in logical blocks. Each element of the real system has its representation in the XML document which simplifies reading its state.

For example, one "Site" element (<xs:complexType name="Site">) includes information about itself and time of read out information:

```
<xs:attribute name="location" type="xs:string"/>
<xs:attribute name="date" type="xs:date"/>
<xs:attribute name="time" type="xs:time"/>
...
and further other elements such as
<xs:element name="Mount" minOccurs="1"
  maxOccurs="4" type="Mount"/>
<xs:element
  name="EthCANconverterPwrSupp"
  type="TwoOutputPwrSupply"
  minOccurs="1" maxOccurs="1"/>
....
```

Nagios Plugin described below reads out data every defined period of time and after parsing XML data stores them in central database where they can be used for further processing.

5.2.3. Central Monitoring and Error Recovery System. The most important assumptions taken into account when developing the monitoring system for the final setup were the following.

(i) Modularity and scalability—these are the most important issues for the system due to the fact that the experiment is still underway and it was essential to consider its dfuture evelopments.

(ii) Operational reliability—it goes without saying that system that controls another system should be more reliable than the latter. If it works fine all information will be collected in one place. It will not be necessary to look for information how the system works in a separate log file what would take a lot of time.

(iii) Easy system configuration—the system should be fully configurable at the interface level allowing administrators to easily define components being monitored and define the way the information is presented.

The general architecture of the monitoring system is presented in Figure 5.

It consists of the following logical modules.

(i) Nagios—main data collector which collects the data from all defined sources and write them to the MySQL [12] database using NDOutils library.

(ii) Nagios plugins—software utilities used to communicate with all monitored components, able to read their state and deliver it to Nagios. The following Nagios plugins classes are present:

  (a) electronic hardware plugin—able to read and parse information about PSUs, weather station, dome position, and so forth, delivered in XML format (already described in the previous chapter),

  (b) PC-s, routers, UPS-es plugins—to monitor state of computers, network equipment and UPS. Ready plugins obtainable with Nagios are being used,

  (c) software components such as PiMan, DAQ, Mount server, GCN server, that control various aspects of the system. Custom plugins that communicate with software modules using CORBA protocols are being used.

(iii) Recovery System—module to perform recovery actions in case Nagios triggers predefined system failure (such as mount slide, computer failure, software module problem, etc.) and/or notice system operators through email/sms.

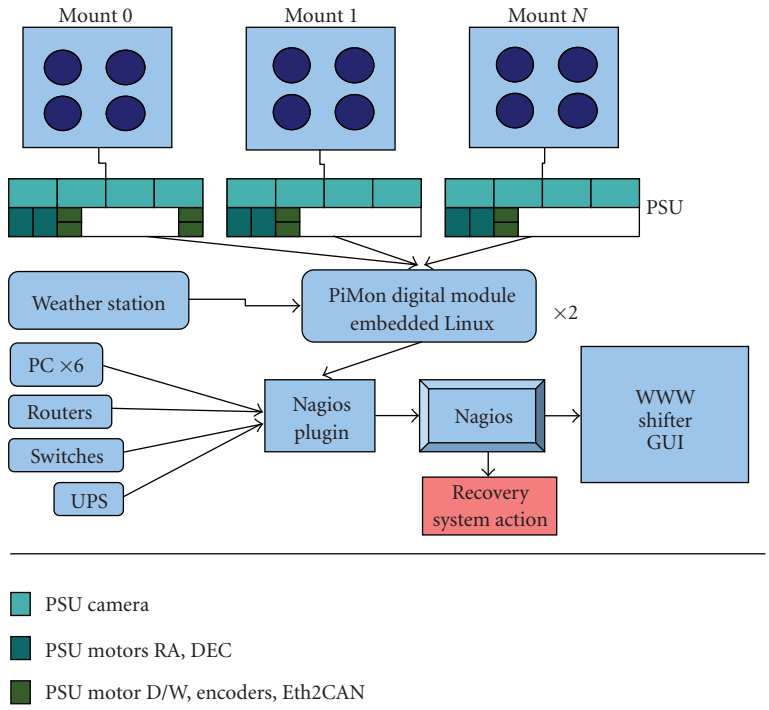

Figure 5: Overview of the data flow in the monitoring system.

(iv) Shifter GUI—user-friendly WWW interface for system operators (shifters), described later.

Details of the system GUI system are given below. The monitoring and controlling system is based on Apache server, Nagios, MySQL database, Django framework, Jquery JavaScripts library, SimpleJson, and AJAX.

Every observation post has its own Nagios server and its own MySQL database where information about the states of the devices is collected. All monitored components of the "Pi of the Sky" experiment have their own tables in database. Structure of table is described in model.py Django file.

This solution guarantees ease of maintenance of the tables. Ease of adding new components to the monitoring panel is an important element of system. It is of crucial importance due to the ongoing development of the project. Adding new parts to the system is performed by adding new interface elements, that is, by filling the relevant table in the database.

The user interface to control and monitor devices consists of a web site with fully dynamic content. The data (i.e., monitoring information) contained in the database are simultaneously displayed by the functions written with the use of JQuery libraries. The data are also simultaneously updated using AJAX technology. When monitored system failures are being detected the operator is immediately informed by alerts on the monitoring web page. Another important part of the monitoring system is the administration panel that enables graphical access to low level database monitoring data.

6. Conclusions

PiMan as the "Pi of the Sky" system manager tool turned out to be very stable, predictable, and easy to extend in many developers' environment. This was achieved by using mature and high level standards like CORBA. In the future the PiMan will be extended to be used in the next phase of the "Pi of the Sky" experiment, as the number of cameras and mounts and the overall system complexity will grow significantly.

Acknowledgments

The authors are very grateful to G. Pojmanski for access to the ASAS dome and sharing his experience with us. The authors would like to thank the staff of the Las Campanas Observatory for their help during the installation and maintenance of their detector. This work has been financed by the Polish Ministry of Science in 2005–2009 as a research project.

References

[1] A. Burd, M. Cwiok, H. Czyrkowski, et al., "Pi of the Sky—all-sky, real-time search for fast optical transients," *New Astronomy*, vol. 10, no. 5, pp. 409–416, 2005.
[2] SWIFT mission, http://heasarc.gsfc.nasa.gov/docs/swift/swiftsc.html.
[3] CORBA, http://www.corba.org/.
[4] Django project, http://www.djangoproject.com/.
[5] Nagios project , http://www.nagios.org/.

[6] MICO, http://www.mico.org/.
[7] Boost C++ libraries, http://www.boost.org/.
[8] STL, http://www.sgi.com/tech/stl.
[9] Readline—library, http://cholm.home.cern.ch/cholm/misc/rlmm.
[10] AT91SAM, http://www.atmel.com/dyn/products/product_card.asp?part_id=3870.
[11] OpenWRT project, http://openwrt.org/.
[12] MySQL database, http://www.mysql.com/.