PiMan - system manager for "Pi of the Sky" experiment

Mikolaj Cwiok\textsuperscript{a}, Lech Mankiewicz\textsuperscript{b},
Krzysztof Nawrocki\textsuperscript{c}, Marcin Sokolowski\textsuperscript{c},
Grzegorz Wrochna\textsuperscript{c}

\textsuperscript{a}Institute of Experimental Physics, Warsaw University;
\textsuperscript{b}Center for Theoretical Physics PAS, Warsaw;
\textsuperscript{c}Soltan Institute for Nuclear Studies, Warsaw;

ABSTRACT

This paper describes PiMan, a CORBA based system manager of the “Pi of the Sky” experiment. The “Pi of the Sky” experiment, located at Las Campanas Observatory in Chile searches for rapidly changing optical objects such as Gamma Ray Bursts (GRB). The system is composed of two CCD cameras located on paralactic mount and operated by a PC, equipped with the dedicated software. The software, divided into modules that correspond to hardware or logical components, controls all aspects of data collection and on-line data analysis. The PiMan assures communication between modules, coordinates their behavior and gives possibility to operate the system automatically and to control it remotely over low bandwidth and unstable link.

Keywords: Gamma Ray Bursts (GRB), astronomy, CORBA, robotic telescopes

1. INTRODUCTION

The apparatus used by “Pi of the Sky” experiment \cite{1} to search for short optical transients, which was custom designed and made, demands dedicated and highly specialized software. One of the required components was the system manager which aim was to ensure communication and interoperation between all other parts and to enable the use of the system as the robotic telescope, i.e. without direct human intervention, over low bandwidth and possibly unstable link. The implemented solution, based on CORBA, is described in the presented paper.

Section 2 of this article describes software components of the “Pi of the Sky” system and interactions between them. In Section 3 the architecture of the PiMan has been described and in Section 4 some external tools are presented, which extend and ease PiMan’s usage. Finally Section 5 contains conclusions and plans for the future.

2. "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. Main modules are:

- Data AQuisition (DAQ) module: responsible for controlling the cameras, reading data streams and on-line data analysis
- Mount module: responsible for controlling the paralactic mount
- GRB Coordinates Network (GCN) module: a module that listens for messages from the GCN network, interprets them and request other “Pi of the Sky” modules to react, if necessary
- HETE module: this module is responsible for reading information about current field of view (FOV) of the HETE satellite and providing this information to the system. Basing on this information the system can be programmed to follow the HETE FOV.
- PiMan module: module responsible for communication and coordination between all other modules. 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, PiMan accepts several methods for controlling the system by the system operator:

- the PiMan execution shell (Pishell)
- the cron to execute periodic commands
- scripts for execution of several of commands (e.g. describing observation program for a given period)
- runscript for execution of scripts from the operating system level (Unix)

PiMan supports 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.

---

**Figure 1. Overview of the “Pi of the Sky” software system**
3. PIMAN ARCHITECTURE

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

- The system had to operate automatically and be controlled remotely over low bandwidth and possibly unstable link.
- The fact that the system consisted of several modules designed by different developers required clean and easy to modify modules’ interfaces.
- Because the system serves as a prototype of the bigger system, the commands and messages had to be easy to modify and add.
- In addition, the two-way communication with modules, ‘special’ operation modes (e.g. GCN alert mode) and thread safety were required.

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

- CORBA [2] (MICO implementation [3]) for inter-module communication
- Readline– [6] was used as the library for Pishell, the PiMan interactive shell

The internal structure of PiMan is presented in Figure 2.

PiMan consists of:

- 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.
• Message Queue containing messages received from modules to be interpreted by PiMan
• Modules’ Locks object containing matrix of modules dependencies (e.g. a module’s command locks itself, internal command locks DAQ and mount, …)
• PiMan Status object that controls PiMan behavior (e.g. enforce mount to automatically follow the HETE satellite, enforce PiMan to execute cron, automatically correct mount position from astrometry, set piman to ‘alert mode’, etc.)
• 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:
• locking mechanism to avoid sending commands to busy modules
• ‘internal module’ used in case when a command uses several modules in one PiMan command (e.g. DAQ and Mount). This module locks all modules concerned, until all respective internal module’s commands are executed
• 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
• scripts can be executed in asynchronous mode, where 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 send 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 priority is 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 then the current unix_time, the current PiMan Status object state doesn’t prohibit its execution. If the requested command doesn’t use a module that currently executes another command then the command is send for execution.

PiMan State in the current state of the system has the following flags:
• ignoreCmds – to run PiMan in the ‘GCN alert mode’ and suppress execution of standard commands,
• manualMode – to give an operator full control of the system by suppressing automatically executed commands,
• updateCron – to reread the PiMan cron file,
• updateConfig – to reread main PiMan config file,
• sunIsAboveHorizon and mountIsParked – self explaining,
• cronPointHete – to update mount position periodically according to HETE FOV,
• autoAG – to decide if the system is to run in the auto guiding mode, i.e. to correct mount movement by data from the astrometry procedure,
• controlCamIdx – to chose main camera for astrometry.

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

As can be seen in Figure2 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’s flag manualMode.

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, trigger_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’s reacts by putting an internal command gcn_trigger in the Command Queue for immediate execution.

The summary of PiMan commands available at the moment is listed in the Appendix A and as an example of using PiMan, a real script interpreted by PiMan is presented in the Appendix B

**4. EXTERNAL TOOLS**

There are several external tools of the “Pi of the Sky” system which enhance PiMan’s functionality:

- the generator used to create scripts for a given time period (e.g. one night) to program “Pi of the Sky” system behavior
- script checker that can be used to check scripts for syntax errors. The checker dynamically learns the PiMan syntax basing on the PiMan sources
- 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.

Below we describe in detail the script generator, which is the system component crucial for the autonomous operation of “Pi of the Sky” experiment.

**4.1 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 in the appendix B. 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 startup, shutdown and the time for evening and morning scan of the whole sky.

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

- Initialize modules – make mount calibration, start cameras cooling immediately after the system startup.
- Start dark frames collection – 15 minutes before the Sun sets 10º below horizon.
- Perform evening and morning scan of the sky when the Sun is 15º below horizon. Subscripts `scan_evening.pish` and `scan_morning.pish` are generated for this purpose and executed at proper time in the main script.
- Observe field of view of a satellite that search for GRBs – HETE or INTEGRAL. strategy of choosing satellite and position to observe is described in details later
- Shutdown the system – 5 minutes after the Sun rises 10º below horizon

System observes only fields from a predefined list of 30º × 30º fields overlapping by 15º. 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 following FOV of one of the satellites. It is realized in the following way.

- The program chooses celestial position, according to the following algorithm:
  - Satellites on the list (currently INTEGRAL and HETE) are checked if altitude of their FOV is at least \( h_{\text{min}} = 30º \) above the horizon and in such case the position is chosen.
• Otherwise, the satellite with biggest part of FOV above horizon is chosen, but it is preferable that satellite FOV is rising rather than setting.
• 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 position above horizon in place where soon one of the satellites will come after rising above horizon.
• Chosen field is checked against calculated position of the Moon and in case it is closer then minimal distance allowed (depends on the Moon phase, for full Moon it is equal to 30°) it is rejected and other object and position must be determined.
• There are several options which allow to observe any object one wants 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 FOV of the satellite selected to follow or define position and time at which it should be observed.
• The program finds closest field from the predefined list, adds command to follow this field.
• The program calculates time when last followed field is lower then 30° 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. 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, when the number of cameras and mounts and the overall system complexity will grow significantly.

6. ACKNOWLEDGMENTS
The authors acknowledge the support of the Ministry of Science and Information Society Technologies under grant 1 P03B 064 29.

REFERENCES

APPENDIX A
The list below contains currently implemented PiMan commands.

• PiMan module commands

piman version – print PiMan version
piman exec_script – execute script from a file (asynchronous mode)
piman exec_script_synchro – execute script from a file (synchronous mode)
piman exec_script_if_dome_not_closed
piman exec_script_synchro_if_dome_not_closed
piman is_dome_opened - check if the “Pi of the Sky” dome is opened
piman clear_cmd_list – remove all commands from the Command Queue
piman del_cmd - delete command with a given id
piman del_thread – delete all commands of a given thread
piman print_cmd_list – print commands in the Command Queue
piman manual_mode_on – set the automatic mode off
piman manual_mode_off – set the automatic mode on
piman auto_ag_mode_on – switch on the auto guiding mode
piman auto_ag_mode_off – switch off the auto guiding mode
piman increment_ignore_cmds – increase the ‘alert mode’ counter
piman decrement_ignore_cmds – decrease the ‘alert mode’ counter
piman return_from_gcn_alert – return to normal operation mode
piman update_config
piman increment_ignore_cmds
piman update_cron
piman cron_point_hete_on – switch automatic following of the HETE FOV mode on
piman cron_point_hete_off – switch automatic following of the HETE FOV mode off
piman cron_send_pos_to_mount_on – switch automatic sending of position found by astrometry to mount on
piman cron_send_pos_to_mount_off – switch automatic sending of position found by astrometry to mount off
piman cron_send_fresh_pos_to_mount_on - switch automatic sending of fresh (calculated on demand) position found by astrometry to mount on
piman cron_send_fresh_pos_to_mount_off on - switch automatic sending of fresh (calculated on demand) position found by astrometry to mount off
piman sleep – suspend PiMan for a given period
piman date – print current date
piman stat – print PiMan status
piman quit – exit PiMan

• Internal module commands

internal prep_system -
internal calib_zenith – calibrate
internal send_fresh_pos_to_mount – send fresh (calculated on demand) position found by astrometry to mount
internal send_pos_to_mount_astro - send position found by astrometry to mount
internal send_pos_to_mount -
internal run_system – start all modules
internal gcn_alert – interpret an alert
internal send_pos_to_daq – send mount position to DAQ
internal fast_pos_to_daq
internal goto_ra_dec – go to a given position defined as (ra, dec)
internal goto_az_h – go to a given position defined as (az, h)
internal gotorel_ra_dec – change mount’s position by (ra, dec) wrt. to the current position
internal gotorel_az_h – change mount’s position by (az, h) wrt. the current position
internal check_mount_status
internal check_mount_position
internal point_hete – move the mount to the HETE FOV
internal pisys_shutdown – shutdown all modules

• HETE module commands

hete get_position – read the HETE FOV from the HETE web page

• Mount module commands

mount raw_cmd – execute an internal mount command
mount init – initialize mount
mount calib – calibrate mount
mount check_ra_dec – check if a position (ra,dec) is accessible by mount
mount check_az_h – check if a position (az,h) is accessible by mount
mount stat – print mount status
mount track – turn mount tracking on/off
mount park – park the mount in safe position
mount flats – turn on the flats making mode
mount site – print mount’s location data
mount quit – turn off mount server
mount pause – suspend mount operation for a given period

• DAQ module’s commands

daq start_analysis – start data taking mode
daq stop_analysis – stop data taking mode
daq start_daq – start DAQ
daq set_fits_key - set a given fits key
daq set_temp – set temperature of the CCD chip for a given camera
daq get_temp - get temperature of the CCD chip for a given camera
daq set_shutter_time – set exposition time
daq get_shutter_time – get exposition time
daq change_param – change a given parameter of a given camera
daq get_param– get a given parameter of a given camera
daq do_darks – turn on the dark taking mode
daq take_npictures – take n pictures
daq take_npictures_synchro - take n pictures
daq set_cooling
daq get_cooling
daq get_position – get mount position from astrometry
daq stat – print DAQ status
daq quit – turn off DAQ server

APPENDIX B

This appendix contains an example script interpreted by PiMan. Lines starting with a hash are treated as comment. For brevity the commands of this night script to be executed between 19:25 and 06:45 local time had been removed.

# night : 20050529
# PRIMARY SATELLITE = INTEGRAL
# SECONDARY SATELLITE = HETE
# SUN sets at 1840 LCO time, at (AZ,H)=(109.44,-9.91) [deg]
# SUN rises at 0640 LCO time
# SWIFT at 20050330_195600 is at (RA,DEC)=(205.06,45.00)
# HETE info file date : 20050529_154200
# HETE RA=247.81=16h31m14.40s DEC=-22.25
# HETE rises above horizont at 1751, sets at 548
# hete at h_hete >= 30.00 at 2015
# MOON RA=341.33=22h45m19.06s DEC=-11.53 illum = 50.59 %
# MOON rises at 20050530_003243, illum = 50.42 %
# INTEGRAL RA=183.38=12h13m32.16s DEC=-5.46
# INTEGRAL rises above horizont at 1414, sets at 237
piman 0 cron_point_hete_off
piman 1 exec_script_synchro(startup.pish)
daq 1825 start_daq
piman 1825 auto_ag_mode_on
piman 1830 manual_mode_off
# Following INTEGRAL at (RA,DEC)=(183.38,-5.46) (az,h)=(232.63,55.20)
# Closest field (RA,DEC)=(180.00,0.00), (az,h)=(222.19,53.20) => OBJECT=1200+00
# At 1920 field (RA,DEC)=(180.00,0.00) of object INTEGRAL is at (az,h)=(206.68,58.19)
# turning OFF cron
piman 1840 manual_mode_on
# do auto correction of position
daq 1840 stop_analysis
mount 1840 raw_cmd(goto ra=180.00 dec=0.00)
mount 1840 stat
daq 1840 set_fits_key(OBJECT,1200+00,-1)
internal 1840 fast_pos_to_daq
daq 1840 stat
daq 1840 change_param(CCD_ASTROMETRY_IN_TAKE_N_MODE,1,-1)
daq 1840 change_param(CCD_WAIT_FOR_ASTROMETRY_IN_SEC,120,-1)
daq 1840 take_npictures_synchro(1)
daq 1840 change_param(CCD_ASTROMETRY_IN_TAKE_N_MODE,0,-1)
daq 1840 change_param(CCD_WAIT_FOR_ASTROMETRY_IN_SEC,0,-1)
internal 1840 send_pos_to_mount
mount 1840 stat
piman 1845 cron_send_pos_to_mount_on
internal 1850 send_pos_to_mount
mount 1850 stat
# turning ON cron
piman 1850 manual_mode_off
# End of tracking (ra,dec)=(180.00,0.00) at 0010 , (az,h)=(106.24,26.77) deg
piman 1900 manual_mode_on
piman 1900 cron_point_hete_off
piman 1900 cron_send_pos_to_mount_off
piman 1905 exec_script_synchro(scan_evening.pish)
piman 1925 manual_mode_off
....
piman 0645 exec_script(shutdown.pish)