Pyrame, a rapid-prototyping framework for online systems

2015 J. Phys.: Conf. Ser. 664 082028

(http://iopscience.iop.org/1742-6596/664/8/082028)

Journal of Physics: Conference Series **664** (2015) 082028 doi:10.1088/1742-6596/664/8/082028

Pyrame, a rapid-prototyping framework for online systems

Frédéric Magniette, Miguel Rubio-Roy, Floris Thiant

Laboratoire Leprince-Ringuet (LLR), École Polytechnique, CNRS/IN2P3, F-91128 Palaiseau, France

E-mail: frederic.magniette@llr.in2p3.fr,rubio-roy@llr.in2p3.fr,thiant@llr.in2p3.fr

Abstract. The present work reports on the software Pyrame, an open-source online framework designed with high-energy physics applications in mind and providing a light-weight, distributed, stable, performant and easy-to-deploy solution. Pyrame is a data-acquisition chain, a data-exchange network protocol and a wide set of drivers allowing the control of hardware components. Data-acquisition throughput is on the order of 4 Gb/s for memory to memory acquisitions and Pyrame protocol overhead is about 50 μs per command/response using the stock tools.

1. Introduction

High-energy physics experiments produce huge amounts of data that need to be processed and stored for further analysis and eventually treated in real time for triggering and monitoring purposes. In addition, more and more often these requirements are also being found on other fields such as on-line video processing [1][2], proteomics [3][4] and astronomical facilities [5].

The complexity of such experiments usually involves long development cycles on which simple test-bench prototypes evolve and eventually grow over time and require increasingly performant software solutions for both data acquisition and control systems. Flexibility and wide application range are, therefore, important conditions in order to keep a single software solution that can evolve over the duration of the development period.

Existing solutions such as LabView [6] provide proprietary solutions for control applications and small scale data acquisition, but fail on scaling to high-throughput experiments, add an important performance penalty and become difficult to maintain when the complexity of the software flow increases. Other lower-level solutions such as LabWindows/C, Matlab [7] or Tango [8] allow to build more complex systems but still fail on providing an integrated and close to turn-key solution for the specific needs of prototypes evolving from bench-based tests to distributed environments with high data-throughput needs. We designed the Pyrame framework to specifically address these requirements.

2. Architecture

The framework emulates a hierarchical architecture of unitary modules. Each of these modules implements a set of functions corresponding to some basic task: handling a specific hardware device, creating an abstraction layer, offering a global service to other modules or piloting the

entirety of a test-bench through other modules. They can interact together to provide sequences or complex tasks. Figure 1 gives a representation of such a system.

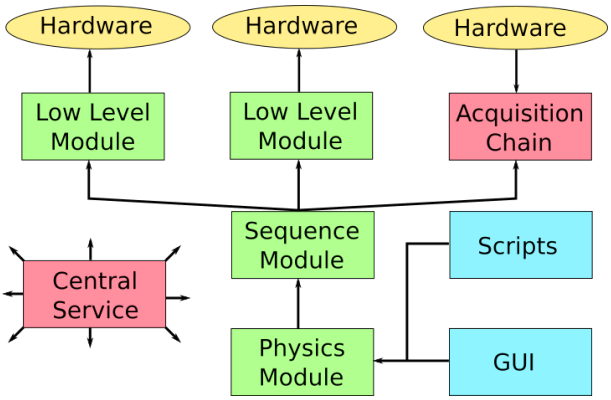

Figure 1. The Pyrame architecture

The emulation of such an architecture is granted by a peer-to-peer asynchronous network. This network is presently implemented by intermittent TCP/IP connections, naturally providing scalability over any TCP/IP network. A local or global directory provides the function-name/IP-address/TCP-ports tuples to initialize the connections. Modules can rely on each others through this network by using a dedicated protocol based on XML. Any module is able to get the API of any other module and send queries or commands to it. The hierarchical architecture guarantees a deadlock-free execution path.

2.1. Command module

The command module is the keystone of the framework. It is the program that handles the network and protocol aspects of a module. It embeds an interpretor to execute the functions in their native or compiled language. Figure 2 describes the internal of the command module.

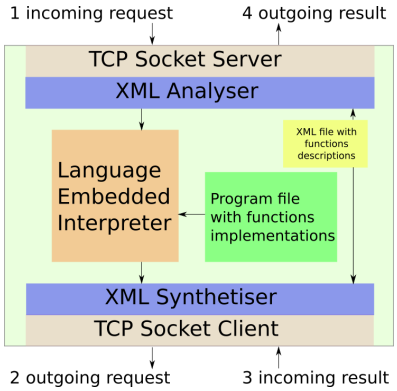

Figure 2. The command module

Language	μs per operation
C	43
C++	47
Lua	55
Python	68
Bash	117

Table 1. Performances

Multiple languages are currently supported. C and C++ can be used by dynamically loading functions in a shared library (dlopen). Python and Lua can be used by a direct embedding of their virtual machine. Bash can also be used as function language, where, the bash process is launched and connected through standard input/output. This serves as a proof of concept that the program could be extended to any command-line interpreter (Scilab, Octave, ROOT, R...). Table 1 presents the different performances that can be obtained depending on the function's implementation language. These performances are obtained with an open socket and without any code payload. It is just a measurement of the system overhead. If the socket is not already opened, a penalty of 70 μs must be added.

2.2. The peer-to-peer asynchronous network
Modules are connected through an asynchronous network. Whenever a module needs to execute a function of another module, it checks the availability of this function in its command module. If the function is found, a TCP socket is established (or an old one is reused) to connect to the corresponding module wherever it physically runs. Then, the addressed module executes the corresponding action and returns the result through the same socket. The socket can then be closed or kept open for another execution. This mechanism creates an intermittent peer-to-peer network connecting the modules when they need it.

2.3. The protocol
The protocol is very simple and is based on XML. A simple command structure is sent to request the execution of a function.

```
<cmd name="set_random_gen"><param>256</param><param>2</param><param>35</param></cmd>
```

The parameter name is the name of the function as it is defined in the command module. The parameters follow unnamed, so their order has to match the function's implementation. Once the execution of the function is finished, the queried module emits the answer including a numerical return code and a description string protected by a CDATA tag:

```
<res retcode="1"><![CDATA[Random generator set]]</res>
```

The return code is 1 for success and 0 for failure (other values can be used for internal control purposes). The string message can be useful in case of failure to determine its cause, and in case of success, to return the requested information.

3. Available Modules
As the framework has been designed for multiple test-benches, we can offer a wide range of modules corresponding to every hardware or every use-case necessary for these test-benches. They can be divided in three categories.

3.1. Hardware modules
We have implemented support for a wide range of hardware devices, summarized in Table 2. The support for more devices is easy to implement (and in some cases straightforward) thanks to the wide range of supported buses: RS232, USB, GPIB, Ethernet, TCP, UDP, VICP, LXI, SCPI.

Device type	Maker
Power supplies	Agilent, CAEN , Hameg , Keithley, TDK-Lambda
Pattern Generators	Agilent
Motion controllers	Newport, Thorlabs, Signatone
Digital storage oscilloscopes	LeCroy
Gaussmeter	LakeShore
Multimeter	Keithley, Agilent
Embedded electronics	Arduino

Table 2. Supported hardware

For most hardware classes, we have implemented convenient abstraction layers, in order to interact with similar devices in a similar way (i.e.: same API). In some cases, the abstraction layers take charge of emulating by software, features that some devices do not implement by hardware. For example, not all power supplies support ramps on power-on or voltage change

(known as slew rate), but this functionality is implemented by software on their abstraction layer with a single and hardware-independant API.

We also have a collection of modules for handling particle detectors read-out chips from the Omega group: Skiroc, Hardroc, Maroc, Easiroc, and we plan to support the complete catalog soon.

3.2. Acquisition Chain

The acquisition chain, described in Figure 3, is a software hub collecting data with multiple missions:

- Acquiring data from different media (raw-ethernet, TCP, UDP, USB, files...) through dedicated acquisition plugins.
- Uncapping data: removing headers and trailers, converting between data formats, verifying data integrity (e.g.: CRC), checking receive order (with sequence numbers) and reordering if necessary, splitting between data and control packets, injecting a reference clock in the data. All these tasks are performed by dedicated uncap plugins.
- Splitting data into separate streams.
- Storing data to files (one file per stream plus a trash). It has also the possibility to store the raw data to replay it later for debugging purposes.
- Dispatching data through TCP sockets. Any program which connects to the broadcasting port will get all the data from the corresponding stream (one TCP port per stream).
- Dispatching data through shared memories (one shared space per stream).

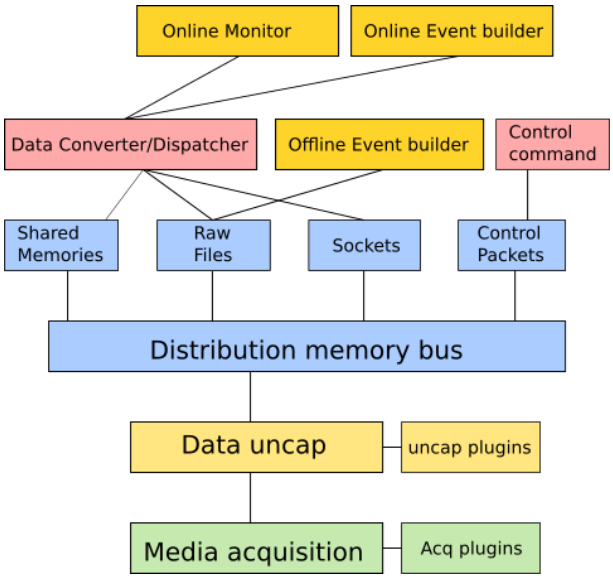

Figure 3. The acquisition chain

The acquisition chain is a real multi-media acquiring system: different data from different media can be acquired and synchronized at the same time via a plugin system. Data are dispatched in different streams corresponding to logical sources (not only different media). The acquisition chain also provides a synchronization mechanism that allows reconstruction of data even if the hardware does not include synchronization information (e.g.: timestamps). Some specific data can be extracted or computed from a stream and stored as a time tag. This, in turn, can be used by all the other acquisition plugins to integrate it into their data.

The different destinations of the data correspond to their different uses:

- Shared Memory: high performance online treatment (conversion)
- TCP Sockets: remote online treatment (online event building)
- Files: offline treatment
- Subsampling: online monitoring

The acquisition chain has been designed for both flexibility and high performance. Its maximum observed performance is around 4 Gb/s. This peformance has been obtained in a memory to memory test pattern, to measure the raw performance of the system (excluding disk or network access overheads).

3.3. Services modules

Pyrame includes a number of global services to allow all modules to work together.

The first one is a global variables module that allows to share values by their name. They are available for every module through the peer-to-peer network. This module allows to share both numeric or string variables and to perform operations on them like arithmetics or concatenation.

An automatic launcher module has also been implemented. This meta-module opens the sockets for all available modules and waits for incoming connections. As an incoming connection occurs, it launches the corresponding module. This mechanism allows to sparsely allocate the resources.

One of the most interesting service modules is the configuration one. An automatic configurator allows to spread a compact xml configuration file over all the modules to configure them. In this file, every module is described by values for all its parameters. In order to reduce the size of the file, a default values file if also provided. In the same way, there is a way to implicitly declare a set of hardware devices with common parameters. This system is generic enough to cope with eventual differences between devices: at run time, the APIs of the required functions are extracted and only the requested parameters are provided.

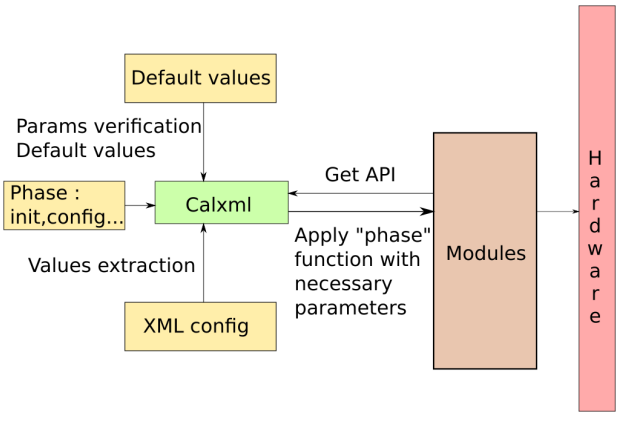

Figure 4. The configurator system

```
<?xml version="1.0" encoding="UTF-8"?>
<detector name="testbench">
  <domain name="localhost">
    <param name="domain_ip">127.0.0.1</param>
    <temp_probe name="tp1">
      <param name="unit">C</param>
    </temp_probe>
    <ps name="power_supply_1">
      <param name="ps_voltage">3</param>
    </ps>
  </domain>
</detector>
```

Example of XML configuration file

Conversely, all the modules can backup the values of their parameters on a configuration module, such that, at any time, an entire copy of the running configuration can be exported in the xml format. This is very convenient to fill a condition database for every run.

4. Interfacing Pyrame

A framework like Pyrame is useful only if it is easily connectible to other tools. Due to the extreme simplicity of the protocol, it is very easy to code bindings for a wide variety of languages. Pyrame includes some of the most popular.

The available bindings are for C, C++, Python and R. These bindings provide functions for simple interfacing: getting the particular port of a module, connecting to it, sending a command and getting back an answer. We have also implemented some experimental bindings for other online framework systems like Tango, OPC unified architecture and Xdaq.

A special module is also dedicated to interaction with users. This module allows sending messages to a human operator to warn him/her or ask questions eventually supported by data or graphics. The answer of the operator can then be taken into account by the system.

Another major need for an online project is to provides GUIs. To ease this task, we provide two specific bindings: LabView and JavaScript. The former allows easily migrating from LabView-only projects to hybrid ones in which LabView acts as a pure GUI and Pyrame manages all other tasks. This is an example of a soft migration scheme. The latter provides an easy way to call Pyrame functions through a web interface.

5. Pyrame on embedded systems

As Pyrame is by itself a very lightweight system, it is easy to use on minimal systems.

It can be used as-is on a Raspberry-Pi. On an up-to-date Raspbian distribution, we obtained a raw performance around 1 *ms* per operation, which is acceptable for a high level module or a slow hardware module. In this way, a Rapberry-Pi can be used to replace a PC as master on a test-bench as long as no data acquisition is involved.

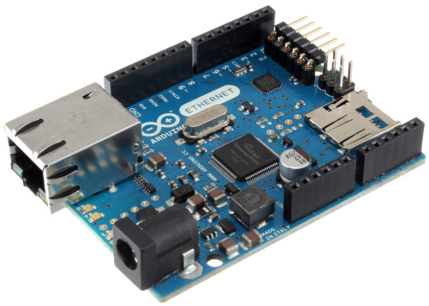

Figure 5. Arduino Ethernet

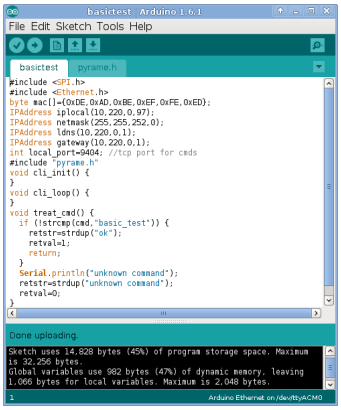

Figure 6. Simplest arduino Pyrame program

We also implemented a micro Pyrame library for Arduino Ethernet, the main difficulty being to implement an XML parser in such a small library. As a standard XML library like expat (which is used in Pyrame) is too big for the Arduino memory, we wrote an adhoc parser that only does the Pyrame parsing job. It is very light and allows to embed a Pyrame server on an Arduino. Its memory imprint represents 45% of code memory and 47% of global variables memory. This lets enough space to implement digital or analog I/O on the dedicated pins. The raw performance of such an implementation if around 7 *ms* per operations on a Gigabit network switch.

6. Applications

The Pyrame framework is used for all the test-benches at LLR. Thus we had to cope with a wide heterogeneity of hardware and procedures. Some of the most complete examples include: an optical single-photon detection bench, mechanical prototypes, or an automatic probe station. In this section, we develop two study cases: a candidate calorimeter for the future linear collider and a 3D magnetic field mapper.

6.1. ILC Ecal

The Silicium/Tungsten ECAL is a high granulometry electromagnetic calorimeter, designed for the future linear collider (ILC). It has been developed at LLR for many years and is now in the technical prototype phase. It is based on silicon wafers with PIN diodes read by Omega Skiroc chips. The readout and data acquisition is done by specific electronics and software described in [9] and [10]. The development of such system involves a lot of different test-benches which have all been managed by the Pyrame framework.

Figure 7. The SiW-Ecal in testbeam at DESY

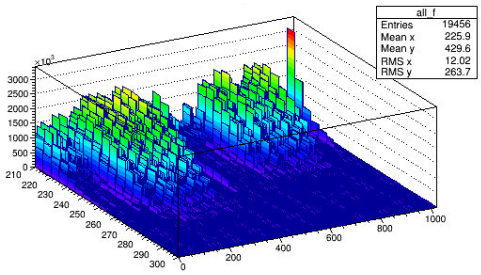

Figure 8. Scurves obtained with Pyrame high level scripts

Pyrame allows us to pilot the whole detector including detector chips and dedicated electronics but also peripheral devices (e.g.: power supplies and pattern generators). A top level module allows access to the whole detector as a black box with a comprehensive API. On top of it, a web graphical interface can be used to manually pilot the detector. A script system is also available to handle the complicated physics procedures like calibrations, data-driven parameter adjustments, construction of S-curves and repetitive data acquisition. The system has been fully tested during three test-beams in DESY between 2012 and 2014. It has demonstrated its stability during very long procedures involving more than 100k configurations and data acquisitions. Some acquisitions ran successfully for weeks.

6.2. 3D magnetic field mapper

The 3D magnetic field mapper is a test-bench designed to produce maps of magnetic fields and to characterize magnets. It is based on three motorized axis and a Hall-probe read by a gaussmeter. The probe being fragile, the control system has to provide security yet flexibility to scan the maximum space without risk of collision.

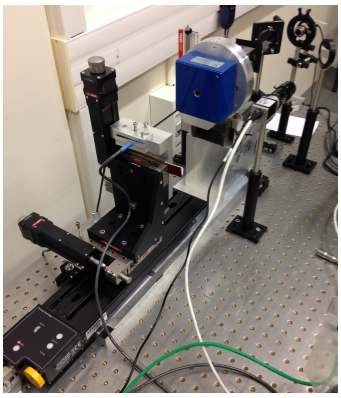

Figure 9. The Gauss Bench (the blue cube is the target magnet)

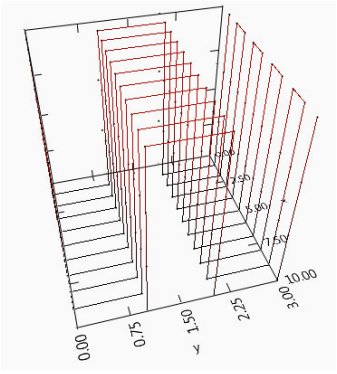

Figure 10. The probing mesh visualized in the Labview GUI

In order to reach these goals, the system implements a special module in charge of designing

the scan path. It takes as parameters the topology of the system, the density of the mesh (which can be different by zones) and the scan order. Once the mesh is generated, movement from one point to the next is possible. For each movement, a path avoiding possible collisions is calculated with a modified A* algorithm [11]. The gaussmeter is piloted by its dedicated module that acquires the data and store them in files. For convenience, a LabView GUI has been designed, allowing the modification of the mesh parameters and its visualization.

7. Community

The framework is released under the LGPLv3 license. You are thus welcome to download it, use it and modify it freely. It is available at the address:
http://llr.in2p3.fr/sites/pyrame
At the same address, you will also find the full online documentation. This documentation is also in the archive.

To share information and discuss technical issues, a mailing list is available at
https://listes.services.cnrs.fr/wws/info/pyrame
If you create a new module, please share it with us in order to enrich the framework from the experience of all users.

8. Conclusion and future work

Pyrame is a framework for quickly prototyping online systems. Very simple to install and use, it allows the online programmer to focus on online topics, abstracting network, protocol and system aspects. It has proven its stability and usability in real experiments as soon as high control rate are not involved.

Near-future improvements will come from the ongoing work on an online event builder for Ecal. This project should be generic enough to fit a lot of cases and provide a useful system for handling and monitoring the data in real-time.

References

[1] J.P. Cnossen, D. Dulin and N.H. Dekker, Review of Scientific Instruments 85 (2014) 103712
[2] A.S. Basu, Lab Chip 13 (2013) 1892-1901
[3] Y. Perez-Riverola, R. Wanga, H. Hermjakoba, M. Mllerc, V. Vesadab, J.A. Vizcanoa, *Biochimica et Biophysica Acta (BBA) - Proteins and Proteomics* 1844 1-A (2014) 63-76
[4] M-Y.K. Brusniak, S-T. Kwok, M. Christiansen, D. Campbell, L. Reiter, P. Picotti, U. Kusebauch, H. Ramos, E.W. Deutsch, J. Chen, R.L. Moritz and R. Aebersold, *BMC Bioinformatics* 12 (2011) 78
[5] D.A. Mitchell, L.J. Greenhill, R.B. Wayth, R.J. Sault, C.J. Lonsdale, R.J. Cappallo, M.F. Morales, S.M. Ord, *IEEE Journal of Selected Topics in Signal Processing*, 2 (5) (2008) 707-717
[6] Labview is a framework from National Instruments. All informations at http://http://www.ni.com/labview
[7] Matlab is a framework from MathWorks. All information at http://mathworks.com/products/matlab
[8] Tango is an opensource SCADA. All information at http://www.tango-controls.org
[9] R. Cornat, F. Gastaldi and F. Magniette, *Journal of Instrumentation* **9** (2014) C01030
[10] F. Gastaldi, R. Cornat, F. Magniette and V. Boudry, A scalable gigabit data acquisition system for calorimeters for linear collider, *Technology and Instrumentation in Particle Physics 2014*, Amsterdam, Netherlands
[11] P.E. Hart, N.J. Nilsson, B. Raphael, *IEEE Transactions on Systems Science and Cybernetics* 4 (2) (1968) 100-107