6th International Conference on Sensors Engineering and Electronics Instrumentation Advances (SEIA' 2020), 23–25 September 2020, Porto, Portugal

Type of Presentation: *Oral*

Topic: Mathematical Tools for Measurements

R-testbench: a Python library for instruments remote control and electronic test bench automation

A. Quenon [1], E. Daubie [2], V. Moeyaert [3] and F. C. Dualibe [1]
[1] University of Mons, Electronics and Microelectronics Unit, 31 Boulevard Dolez, 7000 Mons, Belgium
E-mail: alexandre.quenon@umons.ac.be, fortunato.dualibe@umons.ac.be
[2] University of Mons, Nuclear and Subnuclear Physics Unit, 6 Av. du Champ de Mars, 7000 Mons, Belgium
[3] University of Mons, Telecom. and Electromagnetism Unit, 31 Boulevard Dolez, 7000 Mons, Belgium

Summary: Numerous applications require instruments remote control and electronic test bench automation. Software solutions exist but are either proprietary or low-level programming. In this paper, a high-level open-source software solution, R-testbench, is presented. It is a Python package that allows to manage and control instruments remotely and to fetch data for saving and post-processing. The communication with instruments is built on the well-known and reliable Virtual Instrument Software Architecture (VISA) standard. The main advantages of the proposed solution are the high-level interface and the automatic instrument recognition, which facilitate the user experience. The Python package has been validated by performance characterization, unit tests, and by application to a real use case involving precise current measurements in the framework of a feasibility study in the nuclear domain.

Keywords: Remote control, test bench, instrumentation, automation, software, python, VISA.

1. Introduction

Critical applications, such as experimental research involving nuclear material or precise characterization of integrated circuits, require remote control of measurement instruments and electronic test bench automation for safety and/or accuracy reasons.

In this respect, the most popular proprietary tools that provide instruments control and automation are LabVIEW [1] and MATLAB/Simulink [2]. Both offer (partial) graphical programming, but scripting is based on their own programming languages, which are less known and practiced than generic purpose languages such as C or Python. On the open-source side, Octave and Scilab, which are MATLAB-like programs, have both a toolbox dedicated to instruments control [3], [4]. The main drawback is the low-level programming, which requires experience from the user. Finally, a Python package, PyMeasure, is an existing solution that tries to offer high-level programming [5], but lacks of properties such as automatic instrument recognition.

When comparing the main features of the existing software solutions for instruments remote control and test bench automation, one can observe that the major constraints are (1) the price, (2) the low-level programming, and (3) the need to learn a proprietary language. One solution to solve these problems consists in developing a Python-based automation tool. Indeed, Python has experienced a quick development over the past years, especially for numerical computation, data mining and artificial intelligence [6], [7]. Hence, it can be considered for both academic and industrial technical solutions.

Considering the needs for a high-level open-source tool offering automatic instrument recognition, we have developed R-testbench, an open-source Python package that allows to create a software remote test bench to control distant instruments from a computer and automate a complete electronic test bench. The proposed library relies on the Virtual Instrument Software Architecture (VISA) standard [8], which enables to communicate with instruments by software through many types of interfaces (e.g., GPIB, USB), and that is now implemented on most operating systems (OS). Our approach aims to reach at least the same performance and universality as the aforementioned solutions, while offering a high-level interface as well as improved automation capabilities.

This paper is organized as follows: section 2 describes the software architecture of the proposed Python package, section 3 presents results that validate the package for real use case, whereas section 4 summarizes the key points.

2. Software architecture

The object-oriented paradigm has been used to develop a package which offers a unique manager class, a generic instrument interface, and automatic instrument recognition.

In addition, to make the proposed package robust and reliable, the main Application Programming Interfaces (APIs) available for scientific computation have been used.

2.1. VISA communication API and automatic instrument recognition implementation

The manager class is named RTestBenchManager. Its main objectives are (1) to embed the VISA protocol detecting and connecting instruments to the software testbench, and (2) to manage all connected resources regarding communication channel access, memory allocation and tasks queuing. The VISA libraries are accessed through the PyVISA Python package [9].

As shown in Fig. 1, the base class to control an instrument is Tool, which defines the basic functions required for all instruments, such as send() for configuration or query() to get data. Then, one interface per type of instrument, defining the main functions, is created as an abstract class (a class from which no object can be instantiated), inheriting from Tool. This architecture allows to implement the automatic recognition of instruments by using polymorphism, i.e., an access to a specific class through the generic mother class. The user experience is then facilitated as the interface is always the same, whatever the brand and the model of the instrument.

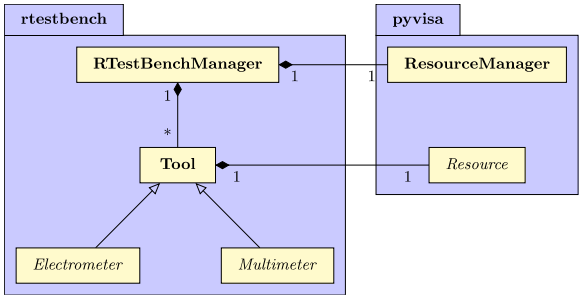

Fig. 1. R-testbench Unified Modeling Language (UML) class diagram: manager, tool interfaces mechanism, and interactions with PyVISA.

2.2. Data containers APIs and data management implementation

All data gathered by R-testbench can be stored by using Python built-in classes, such as lists, or by using scientific computations libraries such as NumPy and pandas. This provides flexibility and possibility to use any data mining or processing framework.

For efficiency reasons, the default data containers used to store data coming from any Tool is the well-known NumPy's N-dimensional array: ndarray [10]. It is the current best compromise between the memory and speed efficiencies of C/C++ and the high-level and user-friendly data manipulation of Python.

For its ease of use and its rich interface, the pandas API [11] is used to gather data all together and to save them to a specific type of file. R-testbench can save data either as human-readable or binary files. Currently, human-readable files types comprise the largely used CSV, which stands for comma-separate values. On the other hand, supported binary files formats are the common Python pickle, as well as HDF5, that is extensively used for big data and artificial intelligence. Other files types are supported by the pandas API, such as the more and more popular JSON, can be used to extend R-testbench in the future.

3. Validation

The R-testbench package has been used to carry out measurements in the frame of experiments that required real time monitoring of semiconductor devices irradiated by β-rays, as described in [12]. Taking advantage of these experiments, the software has been intensively tested together with the B2985A electrometer from Keysight [13]. In addition, automated unit tests have been implemented by using the unittest and PyTest modules. Finally, the package has been made publicly available on GitHub (https://github.com/Arkh42/rtestbench), in order to be shared with and reviewed by the open-source community.

All of these constitute the validation tests of the proposed R-testbench package. Details are presented further in this section.

3.1. Performance measurements

The main measurements to characterize the performance of the proposed software are: 1) the time required to fetch data from the instrument to the computer, 2) the time required to save data locally to a file, and 3) the size of the file. All measurements have been done on a 64-bit computer, Windows 10 Pro OS, 16-GB RAM, Intel Core i7-8550U CPU.

The Keysight B2895A can be controlled by USB and LAN (TCP/IP), among others. The maximum data transfer speed of the USB interface is 488 Mbit/s, whereas it is 100 Mbit/s for the LAN interface (Ethernet cable) [13]. Each datum can be stored either on 32 or 64 bits.

As expected, the transfer through USB is faster than by TCP/IP (Fig. 2). This is, of course, explained by the limitations due to the interfaces of the electrometer. Moreover, data transferred by TCP/IP are embedded with several addresses (MAC, IP), which increase the number of bits to be sent. Data stored on 32 bits are also fetched faster than data stored on 64 bits, which is expected for the transfer speed but could be jeopardized by the fact that the computer operates in 64 bits.

In addition, the fetch time increases linearly with *N*, the number of data. The linear regression performed on the four sets of measured data (Fig. 2) provided the following equations:

$$t_{\text{fetch,USB 32}} = 0.0204\,N + 2.1, \quad (1)$$

$$t_{\text{fetch,USB 64}} = 0.0281\,N + 4.8, \quad (2)$$

$$t_{\text{fetch,LAN 32}} = 0.044\ N + 4.6, \quad (3)$$
$$t_{\text{fetch,LAN 64}} = 0.064\ N + 6.7, \quad (4)$$

times given in milliseconds.

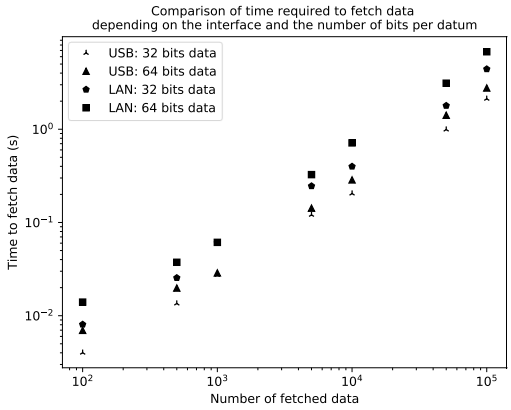

Fig. 2. Comparison of time required to fetch data from the Keysight B2985A electrometer to the computer. Data are stored locally in a numpy.ndarray container.

As explained in section 2, the pandas API is used to save data to files, locally. The time required to save data (Fig. 3), as well as the file size (Fig. 4), have been studied with an increasing number of data.

As expected, CSV is the worst format, considering both speed and memory, especially when the number of data increases. However, if the number of data to save is lower than 1,000, the HDF5 is surprisingly less efficient. A possible explanation might be that this file format has been created to target big data applications and could suffer from a lack of optimization for a small amount of data.

Regarding the binary files, the pickle format seems to have better performance than the HDF5 format. This assertion is true on the tested range of data number. Considering the speed, it seems that the gap tends to reduce for large sets of data (Fig. 3). Conversely, the evolution of the size seems to be linear, except for small datasets.

3.2. Continuous integration automated testing

Reliability is of utmost importance in instrumentation. This is the reason for which the proposed R-testbench package has been built on top of industrial protocols, such as VISA. In order to achieve the same objective, continuous integration and automated tests have been deployed together with the package. Both rely on the well-known PyTest library and GitHub workflows.

The functions of R-testbench are tested extensively (different conditions, success and fail cases) by creating unit tests. Each test ensures that a function behaves as expected for a specific condition. Once a test passes, it should never be modified, so that the user application is guaranteed to continue working even if the development of the package goes further.

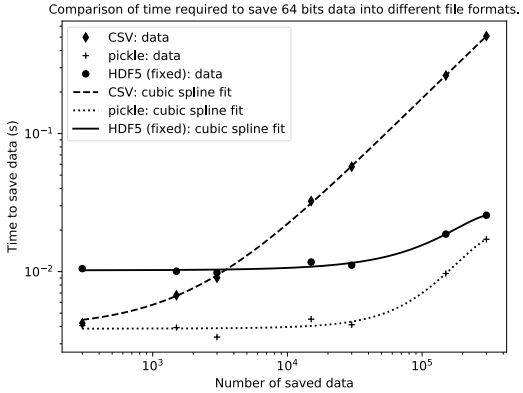

Fig. 3. Comparison of the evolution of the time required to save data into CSV, pickle and HDF5 files formats with the number of data. Data are saved to files using pandas.

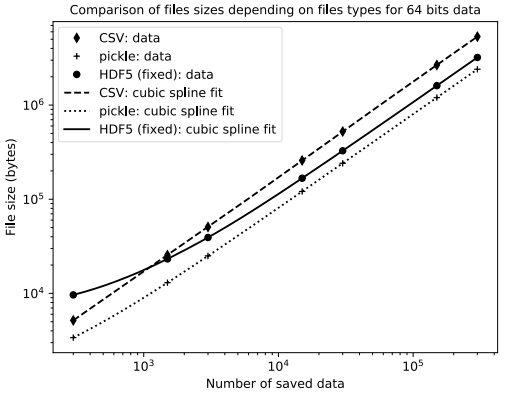

Fig. 4. Comparison of the evolution of sizes of CSV, pickle and HDF5 files with the number of data. Data are saved to files using the pandas API.

Currently, more than 80% of the code is covered by tests. The remaining 20% is mainly the part specific to an instrument, for which automated testing is difficult to implement. Nevertheless, this part has been covered by the performance measurements exposed in section 3.1, as well as during the experiments conducted in [12]. Hence a high confidence level in the proposed tool.

3.3. Open-source community

R-testbench is publicly available on GitHub, an open-source platform used by researchers and developers from industry and academia to share and contribute to open-source projects. It is licensed under the Open Software License v. 3.0, a copyleft license that emphasizes the possibility to share the code.

There is currently one contributor and maintainer: A. Quenon. As the package has been made publicly available for the SEIA' 2020 conference, the interest for this project is expected to grow up.

4. Conclusions

R-testbench is a Python package that allows to control instruments remotely and to automate test benches. Its main advantages over other available solutions are the open-source share and the automatic instrument recognition (Tab 1). It does not support communications by User Datagram Protocol (UDP), but this is not really important for instrumentation as UDP provides a connection less communication model and does not guarantee the packet delivery, so the quality of the data transfer. However, it minimizes the latency during the data transmission.

The library has been tested and validated on a real critical use case, i.e., experiments involving nuclear materials. Several parameters have been measured to test the performance of data transfer speed and data saving speed and memory. Other performance parameters, such as the memory size necessary during execution time, can be monitored in later studies.

The project is now awaiting for reviews by the open-source community to be improved.

Acknowledgements

The authors would like to thank D. Binon and J. Hanton for advices and fruitful discussions about instrumentation.

This work was supported by the Fonds de la Recherche Scientifique — FNRS under Grant n° 33678493.

References

[1]. National Instruments, What is LabVIEW? (https://www.ni.com/en-us/shop/labview.html).
[2]. MathWorks, Instrument Control Toolbox (https://www.mathworks.com/products/instrument.html).
[3]. Octave Wiki contributors, Instrument control package (https://wiki.octave.org/Instrument_control_package).
[4]. Scilab team, Signal acquisition & instrument control (https://www.scilab.org/software/atoms/signal-acquisition-and-instrument-control).
[5]. C. Jermain and G. Rowlands, PyMeasure scientific package (https://pymeasure.readthedocs.io/en/latest/).
[6]. TIOBE, TIOBE Index for April 2020, 2 April 2020 (https://www.tiobe.com/tiobe-index).
[7]. D. Robinson, The Incredible Growth of Python (https://stackoverflow.blog/2017/09/06/incredible-growth-python).
[8]. National Instruments, NI-VISA™ User Manual, 2001.
[9]. G. Thalhammer, T. Bronger, F. Bauer, H. E. Grecco, and M. Dartiailh, PyVISA: Control your instruments with Python (https://pyvisa.readthedocs.io/en/latest/).
[10]. S. van der Walt, S. C. Colbert, G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, *Computing in Science & Engineering*, Vol. 13, Issue 2, 2011, pp. 22-30.
[11]. W. Mckinney, Data Structures for Statistical Computing in Python, in *Proceedings of the 9th Python in Science Conference (SciPy 2010)*, Austin, Texas, USA, 28-30 June 2010, pp. 56-61.
[12] A. Quenon, E. Daubie, V. Moeyaert and F. C. Dualibe, On the Possibility to Use Energy Harvesting on Beta Radiation in Nuclear Environments, in *Proceedings of the Nuclear & Space Radiation Effects Conference (NSREC 2020)'*, submitted.
[13]. Keysight Technologies, Keysight B2980A Series Data Sheet, 2019.

Tab. 1. Main characteristics of software solutions for measurement instruments remote control and automation.

Tool	**License**	**Programming**			**Compatible protocols**	**Automatic recognition**
		Level	**Language**	**Graphical**		
LabVIEW	Proprietary	all	MathScript	yes	all	no
MATLAB	Proprietary	low	Matlab	partial	all	no
Octave	GPL	low	Octave	no	no PXI	no
Scilab	GPL	low	Scilab	no	no UDP	no
PyMeasure	MIT	high	Python	no	no UDP	no
R-testbench	**OSL**	**high**	**Python**	**no**	**no UDP**	**yes**