Sensors & Transducers
Published by IFSA Publishing, S. L., 2020
http://www.sensorsportal.com

R-testbench: a journey in open source programming for remote instrumentation with Python

[1, *] Alexandre QUENON, [2] Evelyne DAUBIE, [1] Véronique MOEYAERT, [1] Fortunato DUALIBE

[1] University of Mons, Electrical Engineering Department, 31 Boulevard Dolez, 7000 Mons, Belgium
[2] University of Mons, Physics Department, 6 Avenue du Champ de Mars, 7000 Mons, Belgium
[1] Tel.: +32 65 37 42 33
[*] E-mail: alexandre.quenon@umons.ac.be

Received: 23/10/2020 Accepted: Published:

Abstract: Nowadays, instrumentation involves remote control and test benches automation, because of the required accuracy or to guarantee the safety of the operators. In parallel, open source has known a fast expansion, and has helped scientific communities in industry and research to build strong collaborations to design effective and user-friendly software tools. This paper presents R-testbench, an open source Python library that offers high-level programming capabilities for instrument remote control and test bench automation. Comparatively to well-known off-the-shelf software solutions, it has been designed to be open source, reliable, adaptable and user-friendly. It relies on the Virtual Instrumentation Software Architecture (VISA) standard, and enables high-level features, including automatic instrument recognition, thanks to the object-oriented paradigm. It works with PyVISA, a Python front end that calls the VISA libraries, and uses the popular NumPy and pandas packages to offer an optimized data management that is efficient regarding the execution time and the memory use. The proposed library has been validated thanks to continuous integration, performance characterization, and alpha tests in the frame of experiments with nuclear material. It has also been released publicly on the GitHub platform, under the Open Software License 3.0, to be shared with and reviewed by the community.

Keywords: automation, instrumentation, NumPy, pandas, Python, remote control, software, test bench, VISA.

1. Introduction

Over the past decades, instrumentation has become increasingly critical for both research and industry. It is a major task, necessary to be able to push the science forward by acquiring reliable experimental results or to characterize a device to gain market shares.

Three aspects must be considered: the accuracy of the results, the physical and health safeties of the operators, and the rise of the open-source software community.

Regarding the precision, it has become essential to develop an automated control of the instrumentation. This might be due to several factors, such as the size of the equipment [1], the number and the speed of specific test sequences that come one after another [2], or the period of time of the events to record, which can be either very long or very short [3].

Considering safety, some experiments require to operate in extreme, possibly hazardous conditions. For instance, the characterization of radiation-hardened electronics, which necessitates a generation of ionizing radiation in order to measure the aging of the circuits. This issue can be addressed by controlling the instrumentation remotely.

Finally, open source software tools are becoming more and more popular in both industry and academia. This can be explained by two main facts: on the one hand, the price of the proprietary tools; on the other hand, the benefits of faster development, expansion, and adoption generally observed when open source licensing is used [4].

This paper presents R-testbench [5], an open source Python library for instruments remote control and test bench automation. Section 2 provides an overview of the specifications, defined according to the current needs in instrumentation, and motivates the design of a new tool. Section 3 explains the adopted software architecture design that enables the desired features. Section 4 offers perspectives on the data management, alongside the instrumentation process. Section 5 demonstrates the chosen software validation strategy. Section 6 presents a discussion of the results, whereas section 7 draws the conclusions.

2. An overview of the specifications

The development of R-testbench has been driven by four main requirements: open source, reliability, adaptability, and user-friendliness.

This section presents the reasons for which the aforementioned requirements are important, and why Python has been chosen to design the software.

2.1. Open source

The open source status has been obtained by licensing the software to the public with the Open Software License 3.0 (OSL-3.0) [6]. Moreover, the Python library related to the project is publicly available on the popular GitHub platform [7]. Also, it has to be noted that the source code of the Python language itself is open source [8].

2.2. Reliability

The reliability of the software is a necessary condition for the reliability of the data measured from an experiment. In this respect, two important features are required: a standard, that rules the communication between the computer and the instruments, and an error management system. This is deployed to check that the process works as expected and, if not, raises an error that can be caught to fix the issue on the software level or to warn the user.

A standard for instruments communication has been existing since 1995, known as Virtual Instrument Software Architecture (VISA). It offers an interface-independent framework to control instruments by software [9], as shown in Fig. 1. There are several implementations of VISA, provided by the main manufacturers of measurement instruments [10]—[12].

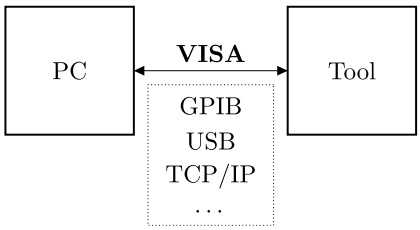

Fig. 1. Illustration of the VISA standard overseeing the communications between a computer and a tool, connected with one of the interfaces among GPIB, USB, and others.

Considering the error management, it is implemented in many object-oriented programming languages, including C++ [13] and Python [14]. Therefore, any of them is suitable to design an instrument remote control software.

In addition to the error management system, a programming language can also offer some standardization. A reliable programming language includes at least a robust standard library, which is the case of both Python and C++. A standard library provides a set of data containers, streams managers (such as terminal or file streams), and specific functions (e.g., mathematics or multithreading).

C++ has been standardized by the International Organization for Standardization as ISO/IEC 14882:2017 [15], for the most recent version. Python, in contrast, is standardized de facto by the developers through the Python Enhancement Proposals (PEPs) [16], which constitute a set of documents describing and tracking the design changes in the language.

2.3. Adaptability

The adaptability must take three different aspects into account: (1) the platforms on which the software is run, (2) the brand and model of the controlled instruments, and (3) the transmission protocols that can be used to communicate with the instruments.

The last two aspects are managed thanks to the VISA standard. Consequently, the only prerequisite is to be able to use a VISA implementation, as discussed in section 2.2.

The cross-platform ability is something slightly more complex. Firstly, the distinction between compiled languages and interpreted languages has to be done. A compiled language requires to be "converted" to machine-language instructions. Consequently, the compilation process must consider the target deployment platform, which can request stronger developer's efforts when a cross-platform software is desired. Conversely, an interpreted language is run immediately, without compilation, by an interpreter, which makes deployment easier, but can, however, generate bugs that will have to be fixed late in the development process.

The discussion about cross-platform leads to a small win of the interpreted languages, which is in fact the nature of Python, whereas C++ is a compiled language.

2.4. User-friendliness

The last requirement for the design of R-testbench is the user-friendliness. Indeed, as VISA libraries already exist, there would be no interest to provide a tool that requires very low-level programming, difficult to use and demanding high programming skills.

Regarding this feature, Python is currently the most suitable programming language for this project. Firstly, it allows to use the object-oriented paradigm, which offers the possibility to encapsulate operating details into objects, easy to implement and to use. Secondly, Python provides great scripting capabilities, making it very convenient to use in laboratories and test facilities. Finally, Python has been granted with powerful scientific computation libraries, such as the popular NumPy and pandas libraries. This allows the user to acquire and process data with only one programming language.

2.5. Motivation of developing a new tool

To the best of the authors' knowledge, there are currently no tools for remote instrumentation that fit the four main specifications discussed herein above. Hence, the design of a new tool, R-testbench.

LabVIEW [17] and MATLAB [18], [19] are the leaders in proprietary solutions for instrumentation. Graphical programming is possible, but generally not sufficient to control complex test benches. In such cases, the proprietary scripting languages that must be used, i.e., G and MATLAB, respectively, are specific to these tools. Consequently, the development communities and the number of related field experts are less important. In addition, no open source libraries are available. However, both companies offer a technical support for their customers.

On the open source side, one can find two solutions based on scripting languages for numerical computing, i.e., Octave [20], [21] and Scilab [22], [23], and one Python package, namely PyMeasure [24]. Both Octave and Scilab require (very) low-level programming skills, and are built with their own scripting language. The PyMeasure Python package is the closest solution found that meets almost all specifications. It offers high-level programming and is based on Python, whose features have been discussed previously. However, it misses the support of common files formats to save the results, and other characteristics provided by R-testbench, such as the automatic instrument recognition (see section 3.2).

3. A software architecture driven by the requirements

The objective of R-testbench is to allow the user to control and automate a test bench, that involves several instruments used for data acquisition. This is done from a unique control interface, a computer, for example. Certainly, the connections between the control interface and the instruments that can be of different types, have to be considered.

The emphasis is made on the user-friendliness. As described in section 2.4, this is partly achieved by using Python, but two major features had to be added. The first one consists in hiding the type of connection (e.g., GPIB or USB) used to control an instrument, from the software side. This is necessary to avoid script modification if the type of connection must change between two runs of experiments. The second feature is an automatic identification of the connected instrument, including the brand and the model. This is mandatory to offer simple and meaningful commands in Python, that will be automatically "translated" by R-testbench into the right instrument-compatible commands.

This section presents the software architecture that has been designed to implement the aforementioned features and the specifications described in section 2.

3.1. A front end to call the VISA libraries

To ensure the reliability of the communication with the instruments, the VISA standard is used. As explained in section 2.2, several implementations of VISA are proposed by the different manufacturers [10]—[12]. In addition, the VISA libraries are operating systems (OS) dependent, which can quickly turn the creation of a generic call method into something hard to handle.

Fortunately, PyVISA [25] is available. It is a Python package that works as a front end able to connect to multiple back ends implementing the VISA standard. This package starts to be recognized by instrumentation engineers and companies as a reliable way to control instruments, as proved by the donation of Keysight to the PyVISA's developers for test purpose.

PyVISA offers a hidden mechanism that makes the connection transparent from the software's perspective. The user has to pass the instrument's address, which will be parsed in order to identify the communication protocol. Then, the corresponding *Resource* object will be created, allowing the user to send commands to the instrument. The process is managed by a **ResourceManager**, responsible for calling the VISA libraries and opening the communication channels to the Resources.

Of course, the future of PyVISA cannot be predicted. The library could be discontinued or not be maintained anymore. In such cases, R-testbench should be able to find another solution, almost transparent for the user. To enable this, R-testbench works in parallel with PyVISA, as shown in Fig. 2. Using the object-oriented paradigm, an encapsulated instance of **ResourceManager** is created and used to build a communication channel with an instrument. Therefore, no direct manipulation of PyVISA has to be done by the user.

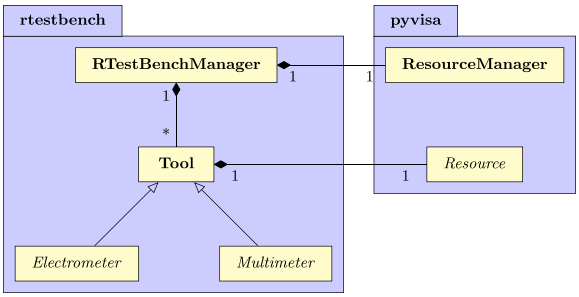

Fig. 2. R-testbench Unified Modeling Language (UML) class diagram: manager, tool interfaces mechanism, and interactions with PyVISA [5].

3.2. One way to recognize them all

The major difficulties in software instrument control are the many ways used by the manufacturers to design their instruments and how they operate. It forces the user to spend a lot of time in reading the documentation to understand the internal behavior of the tools, which is not necessarily relevant to carry out the experiments.

A first attempt of generalization was performed with the Standard Commands for Programmable Instruments (SCPI) [26]. This standard offers a common syntax to program all instruments in a unified style. However, it does not prevent the user from dealing with the specific features of each instrument, such as the data format.

To fix this issue and enhance the user experience, the R-testbench software architecture takes advantage of the object-oriented paradigm to encapsulate the SCPI commands and specific features of a given model of instrument into a dedicated class. These classes inherit from generic abstract classes that will define the interface common to a family of instruments, e.g., oscilloscopes, multimeters, or electrometers. These abstract classes also inherit from an even more generic class, named Tool, which defines the attributes and methods common to all tools that can be controlled by software using the VISA and SCPI standards (Fig. 2).

The proposed architecture offers three main advantages. Firstly, the development related to one specific model of instrument is fast, as it consists in embedding the SCPI commands into class methods and defining the invariants as class attributes. Secondly, a factory mechanism can be used to automatically identify the instrument and build the related object, by sending the "*IDN?" request. This method also ensures that the instrument answers properly. Finally, polymorphism can be used to make the real type of object transparent to the user and offer a unique interface per family of instrument (Fig. 3). Of course, the specific object can always be directly used, if necessary.

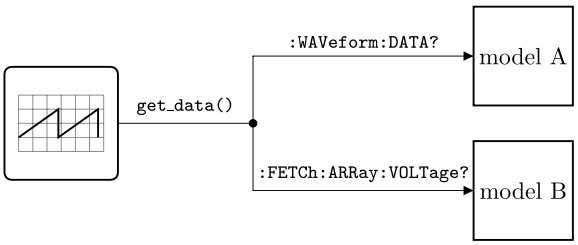

Fig. 3. Example of unique interface taking advantage of the polymorphism and encapsulation mechanisms to hide the specific SCPI commands into a single Python function.

In addition to the Tool inheritance system, a class named RTestBenchManager has been created. Its role consists in calling a VISA resource manager, which is currently provided by the PyVISA library, and creating all resources necessary to provide interactions and other functionalities to the user, such as a logger. All instruments instances are attached to the Manager object, for proper life cycle management. Finally, this class provides facilities for saving data.

4. Data management considerations, from acquisition to post-processing

Scientific research experiments and characterization for industrial purposes share the same problem: the number of data that are retrieved and must effectively be managed. Efficiency is mandatory for two main reasons: the data quality and the computational load. This is monitored using three parameters: the data integrity, the memory usage, and the execution speed.

For this purpose, the data management has been divided in four steps: (1) fetch from instruments, (2) local temporary storage, at execution time, on the machine running R-testbench (e.g., a laptop), (3) permanent saving, i.e., dump data into a file, and (4) processing, for example to extract statistical information (Fig. 4).

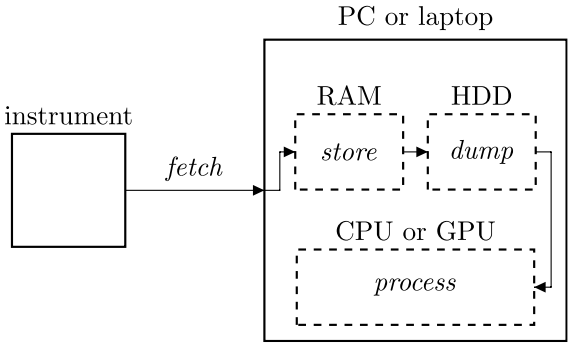

Fig. 4. Illustration of the four-step data management in the case of data acquired with one instrument controlled from a laptop or a personal computer.

This section reviews those four steps to highlight the strategies used for efficient data management.

4.1. From the instruments to the computer: the fetch operation

As data fetch is controlled by the PyVISA front end and the VISA libraries, there is no possibility to optimize the related data management. The only possible action consists in choosing the interface used to communicate with the instruments.

The chosen interface must be compatible with VISA in order to be controlled by R-testbench. Another solution consists in choosing any standard suited for instrumentation [27], and, if not VISA-compatible, to use a gateway whose communication protocol is supported, such as TCP/IP.

4.2. From acquisition to temporary saving: the local storage

Right after the fetch action, when data arrive to the hardware running R-testbench, a local storage in the RAM is necessary in order to process further the data or to save them in the permanent memory. According to the adaptability specification, the library offers the choice to store data in Python built-in containers, such as lists or dictionaries, as well as specialist objects provided by libraries optimized for scientific computations, e.g., NumPy or pandas.

Nevertheless, despite the flexibility given to the user, the default data container used for temporary local storage is the NumPy's N-dimensional array, ndarray [28]. It currently reaches the best compromise between memory management, execution speed, and user-friendly manipulation of huge arrays.

4.3. From temporary to permanent saving: the dump action

At this step, all acquired data are temporarily stored locally, for example in the RAM of a laptop. Afterwards, two actions are jointly performed: aggregation and permanent saving.

On the one hand, it is generally required to aggregate related data for saving. For instance, the vector of timestamps corresponding to the measurement of a time-dependent parameter, such as the voltage on an oscilloscope.

On the other hand, data must be permanently saved into files, for later processing or reading. Depending on the applications, the expected-file formats can be different. The common compromise comes on either the possibility to read immediately the data values by opening the file with a text editor, or an efficient storage that needs the minimum memory space for the maximum number of data, without jeopardizing the accuracy.

Regarding both requirements, the pandas API offers a rich and easy-to-use interface that fits the needs [29]. Currently, R-testbench takes benefits of pandas to offers the possibility to save data in one text format, CSV, and in three binary formats, pickle, Feather, and HDF5. CSV (Comma Separated Values) has the advantage to be human readable and to be supported by a plethora of data processing tools. Pickle is a Python-specific module for object serialization, able to convert an object into a byte stream to write the corresponding data into a binary file [30]. Feather is a format based on the Apache Arrow columnar memory specification for data representation, that is very performant [31], [32]. HDF5 (Hierarchical Data Format) is available in several programming languages, and very popular for data science and artificial intelligence [33].

4.4. From raw data to refined information: the data (post-) processing

Data processing can be performed in two ways: in real time during the acquisition, or in post-acquisition. The former has the advantage of freeing memory as it is not necessary to store raw data permanently, the interesting information being already extracted. However, a simultaneous processing requires either a fast and efficient computation tool (hardware or software) or a huge buffer to accumulate the acquired data while they are being processed. The latter method, post-processing, does not constrain the computation capabilities, but requires the storage of the raw data, which can be a problem from the memory perspective.

Currently, R-testbench implements the post-processing mechanism, in order to allow the user to run the Python library without hard constraints on the hardware. Moreover, because of the various data files formats available, many data processing tools can be used, including open-source ones. This is compliant with the adaptability specification.

5. Software validation

This section presents the four means that have been implemented to validate the proposed software, namely, continuous integration (CI), performance characterization, alpha test and public release on an open-source platform.

5.1. Continuous integration and alpha tests

Continuous integration (CI) is a software development methodology that consists in (1) committing, i.e., sending the modified code to a unique repository, (2) versioning, which means tracking the modification of the code, and (3) building and testing the code each time it is modified to ensure that no anterior functionalities have been broken by the new integration [34], [35].

In the case of R-testbench, a public repository has been created on GitHub (see section 5.4). The versioning tool used for modification tracking is Git.

Automated testing has been implemented by using PyTest, to develop the unit tests, and GitHub Actions and Travis CI, to create the test scenarios including different Python versions (3.6, 3.7, and 3.8) and operating systems, presented in Table 1.

The R-testbench Python package has also been successfully tested by the developer in the so-called alpha tests. They were realized in the frame of experiments involving the characterization of semiconductors devices irradiated with ionizing radiation, e.g., β-rays [36]. This use case requires remote instrumentation, for safety reasons.

Table 1. List of the verified versions of operating systems distributions that are supported, and test methods that used to perform the verification. CI stands for continuous integration.

Distributions	Versions	Test methods
macOS	10.15 (Catalina)	CI (GitHub)
Ubuntu	16.04 (Xenial Xerus)	CI (GitHub, Travis)
	18.04 (Bionic Beaver)	Alpha tests, CI (GitHub)
Windows 10	N/A	Alpha tests
Windows Server	2019	CI (GitHub)

5.2. Performance characterization

All performance measurements were realized with the Keysight B2985A electrometer [37], the instrument which was involved in alpha tests (cf. section 5.3). All results were achieved by running the acquisitions and computing the properties seven times, and averaging the values. This has been determined empirically to minimize the standard deviation.

The time required to fetch data from an instrument and store locally in a NumPy ndarray object evolves linearly with the number of data, as demonstrated in the case of a transfer by USB and LAN (TCP/IP) in Fig. 5.

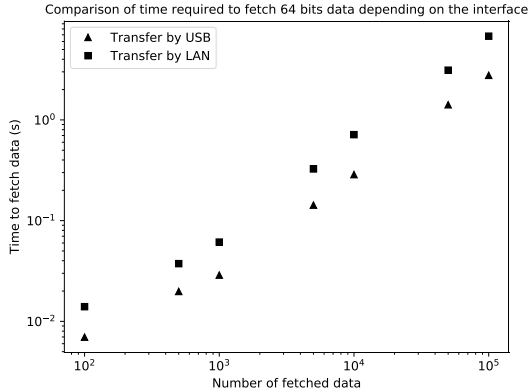

Fig. 5. Comparison of time required to fetch data from a Keysight B2985A electrometer to the computer [5]. Data are stored locally in a numpy. ndarray container.

To evaluate the overhead time, the theoretical transfer time, t_{transf}, must be computed. It depends on the theoretical transfer speed of the related interface, v_{transf}, the number of transmitted data, N, and the number of bits used to code one datum, n_{bit}, as shown in eq. (1):

$$t_{\text{transf}} = \frac{N \cdot n_{\text{bit}}}{v_{\text{transf}}}. \quad (1)$$

The instrument used for the test is equipped with a USB 2.0 interface, which means that the theoretical speed limit is 480 Mbit/s. If the protocol overhead for high-speed bulk transactions is taken into account, it leads to a speed limit of useful data of 425.984 Mbit/s [38]. Hence, using (1), a thousand 64-bit data should require

$$t_{\text{transf}} = \frac{1000 \cdot 64}{425.984 \times 10^6}, \quad (2)$$

leading to a minimum transfer time of 150.24 µs.

Compared with the results shown in Fig. 5, the minimum transfer time due to the interface is two orders of magnitude lower. Consequently, the overhead time necessary to (1) call the VISA libraries that allow to communicate with the instruments and (2) to store the data in a NumPy ndarray object, is a hundred times longer than the transfer time.

Further measurements are necessary to determine the bottleneck of the fetch operation. This requires the possibility to monitor the low-level mechanisms of the VISA libraries, and to test several implementations. Another problem could arise from the data storage with NumPy, which is managed by the PyVISA package.

The speed and size performances of the permanent storage have been characterized, for three different files formats: CSV, pickle, and HDF5. The corresponding results are presented in Fig. 6 and Fig. 7, for the saving time and the size of the generated files, respectively. The best performance is expected to come from binary formats, for both size and speed aspects. However, for a small amount of data, the CSV format generates smaller files, and generate files faster, than the HDF5 format. This can be explained by the fact that the latter has been designed for big data, which assumes the manipulation of huge datasets. On the studied range, which corresponds to a maximum of 300,000 data, the pickle format exhibits the best behavior. This is confirmed by other studies [39], [40]. Nevertheless, the range of data should be extended to draw definitive conclusions, as observed HDF5 and pickle formats tends to have the same performance for bigger datasets, as observed in Fig. 6.

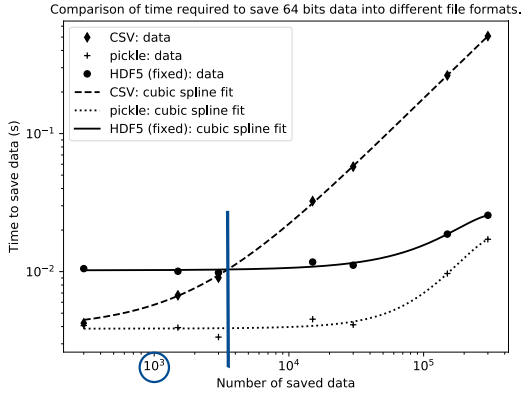

Fig. 6. Comparison of the evolution of the time required to save data into CSV, pickle and HDF5 files formats with the number of data [5]. Data are saved to files using the pandas I/O API.

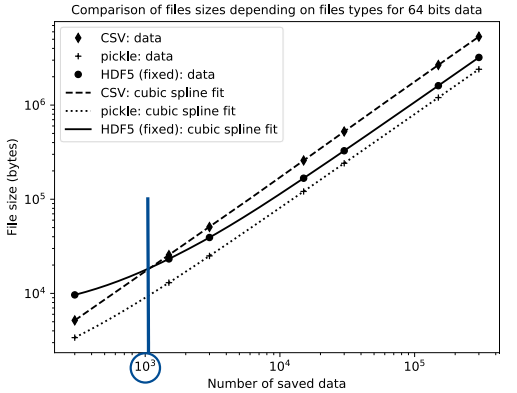

Fig. 7. Comparison of the evolution of sizes of CSV, pickle and HDF5 files with the number of data. [5]. Data are saved to files using the pandas I/O API.

5.3. Public release

R-testbench has been made publicly available on GitHub [7], an open-source platform widely used by researchers and programmers, from both industry and academia, to share and contribute to open source projects. The project is licensed under the OSL-3.0.

There is currently one contributor and maintainer: A. Quenon. As the package has been made publicly available for the SEIA' 2020 conference [5], and because of the contributions to PyVISA due to the strong dependency, the interest for this project is expected to grow up.

6. Results discussion

The proposed library has been clearly characterized for the execution speed and the memory space required to save results. To the best of the authors' knowledge, there is no available studies that offer the same type of characterization for the other available software solutions for remote instruments control.

Nevertheless, R-testbench has several advantages, as shown in Table 2. Specifically, it is open-source, offers high-level programming capabilities, performs automatic instrument recognition and supports several types of files formats to save results. Currently, the main drawbacks are the few number of specific instruments that have been implemented and the small size of the development team. Of course, this is a matter of time.

Table 2. Comparison of the main characteristics of software solutions for measurement instruments remote control and automation [5], [17]—[24].

Tool	**License**	**Programming**			**VISA compatible protocols**	**Automatic recognition**	**Supported results files types**
		Level	**Language**	**Graphical**			
LabVIEW	Proprietary	high or low	MathScript	yes	yes, and more	no	many
MATLAB	Proprietary	low	Matlab	partial	yes, and more	no	many
Octave	GPL	low	Octave	no	no (PXI)	no	bin, CSV, HDF5, mat, txt
Scilab	GPL	low	Scilab	no	yes	no	SOD (HDF5)
PyMeasure	MIT	high	Python	no	yes	no	CSV
R-testbench	***OSL***	***high***	***Python***	***no***	***yes***	***yes***	***CSV, feather, HDF5, pkl***

7. Conclusions

This paper presents R-testbench, an open source Python library devoted to remote instrumentation and test bench automation. The library has been designed using the object-oriented paradigm, to encapsulate implementations details and offer user-friendly high-level programming. It has been devised to be reliable, which is achieved by using the VISA standard for the communications with the controlled instruments. The solution is adaptable to several types of communication interfaces, e.g., GPIB or TCP/IP, as well as to several types of data processing tools, because of the support of different files formats for saving results.

The library has been validated by implementing continuous integration, including code versioning and automated build and tests. It is proved to be cross-platform. Alpha tests were successfully carried out by controlling remotely an experiment involving the characterization of semiconductors devices irradiated with ionizing radiation. The source code is publicly available on GitHub and is becoming mature.

Future developments include the implementation of specific models of instruments, as well as the support of other files formats, such as the increasingly popular JSON text format. It would also be interesting to create bindings with database management tools, e.g., Mongo DB or MySQL. Finally, the possibility to run R-testbench on portable platforms, such as Raspberry Pi or FPGAs, together with hardware acceleration, can be explored. The latter is a clue for efficient real-time data processing.

Acknowledgements

The authors are grateful for D. Binon, for his advices on instruments and test benches, and his help with proprietary instrumentation software. They also thank J. Hanton, for his help for solving IT issues, as well as S. Devouge and F. Defontaines for their support and kindness.

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

References

[1] The CMS Collaboration *et al.*, The CMS experiment at the CERN LHC, *Journal of Instrumentation*, Vol. 3, Issue 08, 2008.

[2] Z. J. Deng, N. Yoshikawa, S. R. Whiteley, and T. Van Duzer, Data-driven self-timed RSFQ high-speed test system, *IEEE Transactions on Applied Superconductivity*, Vol. 7, Issue 4, 1997, pp. 3830–3833.

[3] V. de Miguel Soto, J. Jason, D. Kurtoğlu, M. Lopez-Amo, and M. Wuilpart, Spectral shadowing suppression technique in phase-OTDR sensing based on weak fiber Bragg grating array, *Optics Letters*, Vol. 44, Issue 3, 2019, pp. 526–529.

[4] L. E. Hecht, L. Clark, and The Linux Foundation, Survey: Open Source Programs Are a Best Practice Among Large Companies, *The New Stack*, 2018 (https://thenewstack.io/survey-open-source-programs-are-a-best-practice-among-large-companies/).

[5] A. Quenon, E. Daubie, V. Moeyaert, and F. C. Dualibe, R-testbench: a Python library for instruments remote control and electronic test bench automation, in *Sensors and Electronic Instrumentation Advances: Proceedings of the 6th International Conference on Sensors and Electronic Instrumentation Advances (SEIA' 2020) and the 2nd IFSA Frequency & Time Conference (IFTC' 2020)*, Porto, Portugal, 23–25 September 2020, pp. 47–50.

[6] Opensource.org, The Open Software License 3.0 (OSL-3.0) (https://opensource.org/licenses/OSL-3.0).

[7] A. Quenon, R-testbench, 2020 (https://github.com/Arkh42/rtestbench).

[8] Python Software Foundation, History and License, 2020 (https://docs.python.org/3/license.html).

[9] IVI Foundation, VPP-4.3: The VISA Library, 2018.

[10] National Instruments, NI-VISA Overview, 2020 (https://www.ni.com/en-rs/support/documentation/supplemental/06/ni-visa-overview.html).

[11] Rohde & Schwarz, R&S®VISA, 2020. (https://www.rohde-schwarz.com/dk/applications/r-s-visa-application-note_56280-148812.html).

[12] Keysight Technologies, Software I/O Layers - VISA, VISA COM, SICL, Keysight 488 - Technical Overview, 2009 (https://www.keysight.com/main/editorial.jspx?ckey=1461160&id=1461160&nid=-33002.0.00&lc=dut&cc=BE).

[13] B. Stroustrup, The C++ Programming Language, 4th ed., *Addison-Wesley*, 2013.

[14] B. Slatkin, Effective Python: 90 Specific Ways to Write Better Python, 2nd ed., *Addison-Wesley Professional*, 2020.

[15] International Organization for Standardization, ISO/IEC 14882:2017, 2017 (https://www.iso.org/standard/68564.html).

[16] Python Software Foundation, PEP 1—PEP Purpose and Guidelines, 2013 (https://www.python.org/dev/peps/pep-0001/).

[17] National Instruments, What is LabVIEW?" (https://www.ni.com/en-us/shop/labview.html).

[18] MathWorks, Instrument Control Toolbox (https://www.mathworks.com/products/instrument.html).

[19] MathWorks, Supported File Formats for Import and Export—MATLAB & Simulink, 2020. (https://www.mathworks.com/help/matlab/import_export/supported-file-formats.html).

[20] Octave Wiki contributors, Instrument control package, 2020 (https://wiki.octave.org/Instrument_control_package).

[21] J. W. Eaton and Octave developers, Simple File I/O, 2020 (https://octave.org/doc/v5.2.0/Simple-File-I_002fO.html).

[22] Scilab team, Signal acquisition & instrument control (https://www.scilab.org/software/atoms/signal-acquisition-and-instrument-control).

[23] Scilab team, save format—Format of files produced by 'save.' (https://help.scilab.org/docs/6.1.0/en_US/save_format.html).

[24] PyMeasure Developers, PyMeasure scientific package — PyMeasure 0.8.0 documentation, 2020 (https://pymeasure.readthedocs.io/en/latest/index.html).
[25] G. Thalhammer, T. Bronger, F. Bauer, H. E. Grecco, and M. Dartiailh, PyVISA: Control your instruments with Python, 2020 (https://github.com/pyvisa/pyvisa).
[26] SCPI Consortium, Standard Commands for Programmable Instruments (SCPI), 1999 (https://www.ivifoundation.org/specifications/default. aspx).
[27] J. Park, S. Mackay, and E. Wright, Practical Data Communications for Instrumentation and Control, *Newnes*, 2003.
[28] S. van der Walt, S. C. Colbert, and G. Varoquaux, The NumPy Array: A Structure for Efficient Numerical Computation, in *Computing in Science & Engineering*, Vol. 13, Issue 2, 2011, pp. 22–30.
[29] W. Mckinney, Data Structures for Statistical Computing in Python, in *Proceedings of the 9th Python in Science Conference*, Austin, Texas, 28 June-3 July 2010, pp. 56–61.
[30] Python Software Foundation, pickle — Python object serialization, 2020 (https://docs.python.org/3/library/pickle.html).
[31] W. McKinney, Feather: fast, interoperable data frame storage, 2019 (https://github.com/wesm/feather).
[32] The Apache Software Foundation, Apache Arrow Overview (https://arrow.apache.org/overview/).
[33] The HDF Group, HDF5 File Format Specification, 2019.
[34] T. Durieux, R. Abreu, M. Monperrus, T. F. Bissyandé, and L. Cruz, "An Analysis of 35+ Million Jobs of Travis CI," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Cleveland, OH, USA, USA, 29 September-4 October 2019, pp. 291–295.
[35] E. Laukkanen, J. Itkonen, and C. Lassenius, Problems, causes and solutions when adopting continuous delivery—A systematic literature review, *Information and Software Technology*, Vol. 82, 2017, pp. 55–79.
[36] 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 12th IEEE Latin American Symposium on Circuits and Systems (LASCAS2021)*, Arequipa, Peru, 20–24 February 2021, submitted.
[37] Keysight Technologies, Keysight B2980A Series Data Sheet, 2019 (https://www.keysight.com/en/pc-2444652/b2980a-series-femto-picoammeter-and-electrometer-high-resistance-meter).
[38] Compaq Computer Corporation *et al.*, Universal Serial Bus Specification, 2000.
[39] pandas development team, IO tools (text, CSV, HDF5, …) — pandas 1.1.3 documentation, 2020 (https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html).
[40] I. Zaitsev, The Best Format to Save Pandas Data, 2019 (https://towardsdatascience.com/the-best-format-to-save-pandas-data-414dca023e0d).

Published by International Frequency Sensor Association, IFSA Publishing, S. L., 2020 (http://www.sensorsportal.com).