PYTHON IMPLEMENTATION OF IEEE C37.118 COMMUNICATION PROTOCOL

Stevan Šandi[*], Tomo Popović[**] , Božo Krstajić[***]

***Keywords**: Synchrophasor, IEEE C37.118, Phasor Measurement Unit, Phasor Data Concentrator*

Abstract: The analysis of tools for supporting the measurement of synchrophasors revealed the need for open, customizable, and platform independent tools. One such software tool is the open implementation of the IEEE C37.118 communication protocol for data transfer. This paper describes an implementation process of the protocol in a form of a Python library module. The discussion illustrates its use and validation using third-party tools. Finally, the paper outlines future work on the improvements and possible practical applications.

1. INTRODUCTION

Synchrophasors are time-synchronized phasor measurements. Satellite synchronized clocks allow time-stamped measurements that enable data to be aligned on reference time base. This provides an accurate and comprehensive view of a power system [1]. Synchrophasor support tools could play an important role in the development of Wide Area Monitoring, Protection, and Control (WAMPAC) solutions, because they enable validation of Phasor Data Concentrators (PDCs), communication networks and resources, phasor estimation algorithms and end-to-end tests [2][3][4][5]. The review of existing synchrophasor support tools revealed opportunities for the development of IEEE C37.118 support tools suite independent of the platform and operating system [6]. Furthermore, such tools are necessary to support synchrophasor application development and their evaluation [7]. An example is Phasor Measurement Unit (PMU) simulator capable of sending realistic PMU data streams, which may include phasor data corresponding to various test scenarios (i.e. pre-defined measurement, power system faults, synchrophasor commands). As for networking tests, it is important to evaluate latencies, errors, and

* Stevan Šandi is with Faculty of Electrical Engineering, University of Montenegro, Montenegro (e-mail: stevan.sandi@gmail.com).

** Tomo Popović is with Faculty of Electrical Engineering, University of Montenegro, Montenegro (e-mail: tp0x45@gmail.com).

*** Božo Krstajić is with Faculty of Electrical Engineering, University of Montenegro, Montenegro (e-mail: bozok@ac.me).

throughput during the transmission of synchrophasor data. The evaluation tools need to be easy to use and install with minimal dependency on third party libraries.

The requirements for PMU simulator and implementation of C37.118 communication protocol are given in Section 2.

Section 3 discusses the reasons why the Python programming language was selected and describes the implementation of the communication standard as Python module. The usage of the module is demonstrated with simple, yet powerful code examples.

Section 4 covers the validation of the implementation using external third-party tools. In addition, Section 4 discusses practical scenarios for the use of the newly developed module in real life situations.

The Python implementation of the synchrophasor communication protocol is only the first step in making a complete suite for synchrophasor tools for simulation, communication, and evaluation tools. Final discussions and future work are given in the Conclusion section.

2. REQUIREMENTS

In order to correctly capture the dynamics of electric power system in real time, PMU devices report phasor data up to 240 times per second. In addition, phasor data have to be properly time stamped. Global Positioning System (GPS) satellite clocks are used as time references for that purpose. PDC devices receive the phasor data and utilize time stamp information to properly sort and align measurements obtained from several PMUs. The time synchronization requirements affect physical implementations of PMU and PDC devices, but also their software simulator implementations. The timing in [8] is defined as Time = SOC + FRASEC / TIME_BASE allows time resolution of 59 ns. Synchrophasor measurements need to be synchronized with Coordinated Universal Time (UTC) with an accuracy that does not create phase angle error bigger than 0.01 radian. This requires a clock accuracy that corresponds to timing error of 26 μs at nominal frequency of 60 Hz or 31 μs at nominal frequency of 50 Hz. When talking about software implementations, achieving such timing accuracy depends on selected hardware and operating system platforms [9]. For example, Windows 7 operating system provides a function called *QueryPerformanceCounter*, which offers a maximal timing resolution better than 1 μs [10]. However, timing information obtained this way depends on CPU clock and are not necessarily accurate. Linux and Unix systems offer accuracy of system time in ns resolution. The requirements for PMU simulation may be achieved of the hardware clock is synchronized to the UTC time using Network Time Protocol (NTP). It is desirable for the simulation tools to offer time synchronization using an interface to GPS time reference [11][12].

It is assumed that the simulation tool will send pre-defined or random synchrophasor measurements. In order to make a simulation module appear as a realistic PMU or PDC device, it is necessary to have the correct implementation of IEEE C37.118 communication protocol [13]. As for the user of the simulation package, the protocol implementation needs to offer simple means for transfer of four types of messages – data frames: measurements, configurations, header, or command. The implementation needs a simple Application Programming Interface (API) that allows easy packing relevant information to and from

data frames according to the standard. At the receiving end, the implementation should offer functionality to check validity of data frames and then unpack all the relevant information so that it can be used and presented to the user. Finally, the implementation of the communication protocol needs to be validated using third-party tools provided by independent vendors [14][15][16]. Two test scenarios for evaluation of the communication protocol implementation are depicted in Fig. 1.

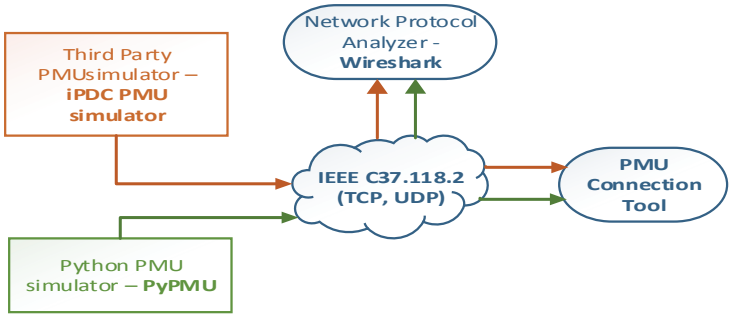

Fig. 1. Test scenarios for validation of the communication implementation.

The first test scenario uses a network protocol analyzer such as Wireshark, which is capable of recognizing C37.118 protocol messages [17]. The second test scenario assumes there is an exchange of messages between the module under test, PMU simulator, and third-party test tool PMU Connection Tool, which is designed to check correctness of the received messages [18]. In addition, the newly developed PMU simulator (PyPMU), implemented in Python programming language, needs to be compared to a third-party PMU simulator from iPDC open source suite.

Additional API functions may be needed to make it easier to create PMU/PDC simulator instances and configure the selection of transport protocol (TCP or UDP). For that reason, the implementation should be in Python library form following the Python standards and programming styles. The implementation should be cross-platform with minimal dependencies on external libraries so that the users have maximal freedom for its utilization and further development of synchrophasor simulation tools.

3. IMPLEMENTATION

Based on the requirements discussed in previous section, the Python programming language has been selected for the implementation of the communication module for the PMU simulator. The Python programming language emerged in 1990s and supports various programming paradigms: object oriented, imperative, functional, procedural, etc. It comes with a very rich standard library and great community support [19]. As an open source, Python is also very popular within the scientific community and there are several scientific tools based on Python (numpy, scipy, and others). Python is available for a variety of platforms and therefore it makes a great choice for cross-platform applications. It comes

preinstalled on Linux and Mac OS X systems, and is very easy to download and install on Windows [20]. Also, Python comes with `pip` packet manager, which allows easy installation and exchange of packages – useful library modules available in Python Package Index (PyPI) repository [21]. There is also the Python Enhancement Proposal (PEP) that includes recommendations for coding practices aimed at the creation of more readable and easy to maintain code. All of these Python characteristics made it a perfect choice for the implementation of the synchrophasor communication toolbox.

The implementation of the communication resulted in a Python module frame inside the package synchrophasor that can be a placeholder for addition of future modules. The easiness of it's use is illustrated in following code snippet:

```
from synchrophasor.frame import ConfigFrame2
from synchrophasor.frame import DataFrame
```

Fig. 2. Importing IEEE C37.118 communication classes from synchrophasor library.

The messages and data frames defined in the standard have been described using class abstraction, which allows users to utilize the module in object oriented fashion. One of the goals was easy access to the functionalities where the user needs to define a subset of required parameters while the rest of them may have default values. Python also allows for explicit specification of each of the parameters. The following code excerpt illustrates two ways of creating configuration data frames:

```
cfg1 = ConfigFrame2(1, "Station A", 991)
cfg2 = ConfigFrame2(1, "Station B", 992, fnom=50)
cfg3 = ConfigFrame2(1, "Station C", 993, data_rate=30)
df = DataFrame([(14635, 1230), (1092, 0)], 2500)
```

Fig. 3. Examples of creating of communication frame instances.

As seen in code snippet in Fig. 3, configuration frame cfg1 is created using parameters such as number of PMUs sending the data (integer value 1), name of the substation ("Station A"), and unique sender's identifier (integer value 991). In addition to the required parameters, configuration frames cfg2 and cfg3 illustrate the use of additional parameters such as nominal frequency (integer value `fnom` = 50 Hz) and reporting rate (integer value `data_rate` = 30 frames per second). The availability of the pre-defined default values allows the user of the library to quickly and easily set up working examples that can be tuned later. The last line of the code illustrates the creation of the data frame, which in this case contains two phasor measurements written in complex notation and deviation from the nominal frequency.

It can be seen that the creation of a software PMU simulator can be easily done with few API calls that create and utilize communication frames. One such implementation of the PMU simulator is provided in the synchrophasor package. Both TCP and UDP methods for data exchange are provided. In order to achieve better performance, the PMU simulator from the library uses processes for each communication link. PMU simulator instance listens to the port for a command to start sending data. An example of PMU simulator instance creation is given in the following code:

```
from synchrophasor.pmu import Pmu
from synchrophasor.frame import ConfigFrame2
pmu1 = Pmu()
pmu2 = Pmu(9991, method='udp')
cfg = ConfigFrame2(1, "Station X", date_rate=60)
pmu2.set_config(cfg)
```

Fig. 4. PMU simulator instance creation.

The example in Fig. 4 illustrates the creation of two PMU simulators. The first one listens on port 4712, which is the default port for TCP transmission method suggested by standard. The second one listens on port 9991 and uses UDP transmission method. Also, the second PMU overrides the pre-defined configuration frame and uses custom configuration setup – PMU Id is 1, station name is "Station X" and report rate is 60 measurements per second. When omitted, configuration parameters assume default values defined in the library. Once send command has been received, the simulators send fixed pre-defined value or random values from the pre-defined measurement interval.

Fig. 5 illustrates the creation of a fully functional PMU simulator with custom configuration parameters, custom header message and defined default measurements. It can be seen that the PMU simulator has been implemented with only a few lines of code.

```
from synchrophasor.pmu import Pmu
from synchrophasor.frame import ConfigFrame2
from synchrophasor.frame import HeaderFrame
from synchrophasor.frame import DataFrame

pmu = Pmu(9991, "127.0.0.1")
cfg = ConfigFrame2(7734, # PMU ID
1000000, # TIME_BASE
1, # Number of PMUs included in DataFrame
"Station A", # Station Name
7734, # Data Stream ID
(False, False, True, False), # Data Format @see docs
4, # Number of phasors
3, # Number of analog values
1, # Number of digital values
["VA", "VB", "VC", "I1", "ANALOG1", "ANALOG2", "ANALOG3", "BREAKER 1
STATUS", "BREAKER 2 STATUS", "BREAKER 3 STATUS", "BREAKER 4 STATUS",
"BREAKER 5 STATUS", "BREAKER 6 STATUS", "BREAKER 7 STATUS", "BREAKER 8
STATUS", "BREAKER 9 STATUS", "BREAKER A STATUS", "BREAKER B STATUS",
"BREAKER C STATUS", "BREAKER D STATUS","BREAKER E STATUS", "BREAKER F
STATUS", "BREAKER G STATUS"], # Channel Names
[(915527, 'v'), (915527, 'v'), (915527, 'v'), (45776, 'i')], # Phasor
conversion factor
[(1, 'pow'), (1, 'rms'), (1, 'peak')], # Analog #Conversion factor
[(0x0000, 0xffff)], # Mask for digital status words
60, # Nominal frequency
22, # Configuration change count
240 # Rate of phasor data transmission)
```

```
hf = HeaderFrame(7734, # PMU ID
"Hello! I'm little PMU!", # Header Message)

df = DataFrame(7734, # PMU ID
('ok', True, 'timestamp', False, False, False, 0, '<10', 0), # Stat Word
 [(14635, 0), (58218, 52860), (58218,12675), (1092,0)], # Phasor
measurements - rectangle #representation
2500, # Frequency deviation
0, # Rate of change of frequency
[100,1000,10000], # Analog values
[0x3c12], # Digital status word
 0x0004)

pmu.set_config(cfg)
pmu.set_header(hf)
pmu.set_default_measurement(df)

pmu.run()
```

Fig. 5. Customized PMU simulator

The example of a simple GUI is shown in Fig. 6 where you can see the simplified configuration panel for PMU. User defined values for PMU ID, TCP port, data format, nominal frequency and report rate will override default values and the PMU is able to send one, fixed data frame. A simulator like this one can be a very useful tool in synchrophasor testbeds to evaluate communication network and equipment, basic PDC functions and configuration, and implementation of end-to-end test scenarios.

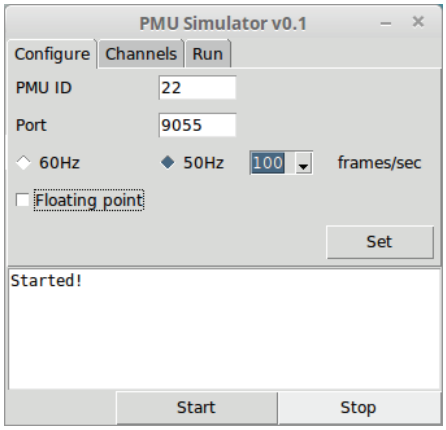

Fig. 6. GUI for PMU simulator.

4. TESTING AND VALIDATION

The implementation of the communication module has been evaluated using external tools developed by third party vendors that provide support for C37.118 standard. The validity of the implementation can be verified using tools for network protocol analysis as well as other PMU/PDC specific tools that were previously shown in Fig. 1. Both test scenarios have been implemented using communication between virtual machines (VirtualBox): a) Ubuntu 14.04 64-bit machine hosting PMU simulator that was sending data; b) Windows 7 64-bit machine used to receive the data. A network traffic analysis tool called Wireshark was utilized to validate the correctness of received data frames since it comes with the support for IEEE C37.118 communication protocol [16]. Testing was done by creating configuration and data frames using `synchrophasor.frame` module, sending of the frames over the network, and capturing and analysis of the frames using Wireshark at the receiving end. In the example shown in Fig. 3, the data frame was created using predefined values from Appendix D, table D.1 in the standard [12]. It is also desirable to send the same or similar data frames using a third party PMU simulator (i.e. iPDC PMU simulator) to perform a sanity check on the use of Wireshark tool.

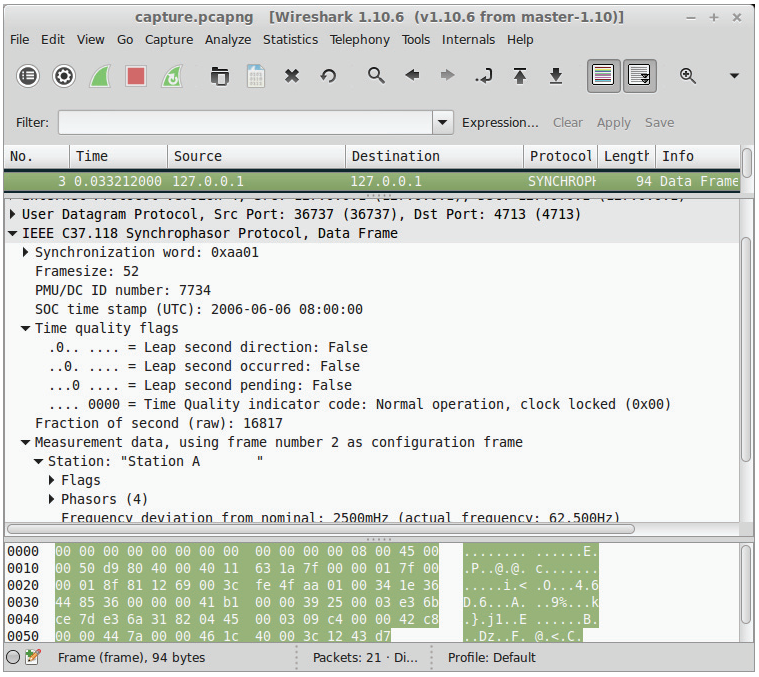

Fig. 7. Using Wireshark to capture a synchrophasor data frame.

It can be seen in Fig. 7 that Wireshark correctly recognized synchrophasor data frame and labeled measurements correctly. Also, the actual values match the values used to create the data frame in the PMU simulator code.

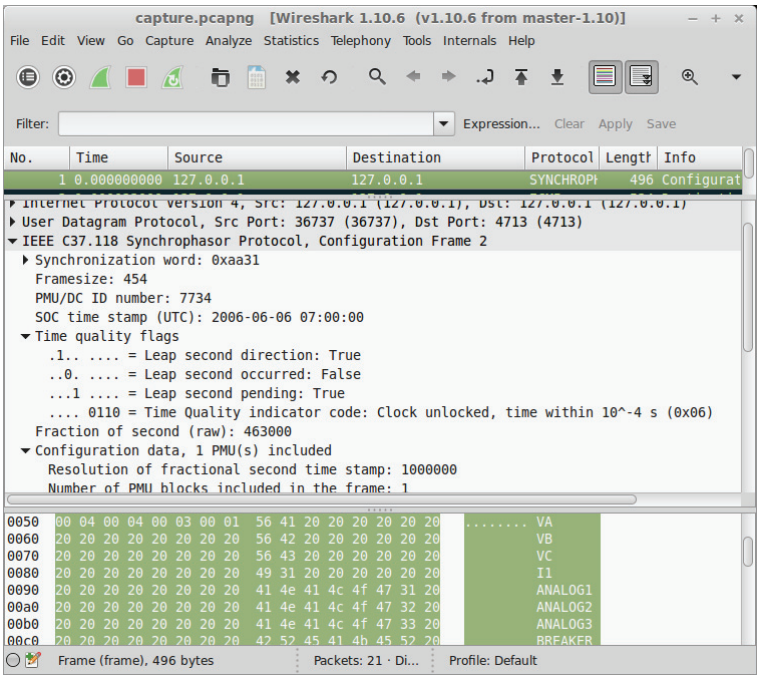

Fig. 8. Using Wireshark to capture a configuration frame.

Fig. 8 depicts the use of Wireshark to capture and identify a configuration frame. The values used to create the configuration frame have been checked against the values captured with Wireshark analysis tool. The same test scenario was exercised using iPDC PMU simulator and Wireshark.

Another approach for evaluation of the PMU simulator utilized a test scenario where the PMU Connection Tester was used to communicate with the simulator script. PMU Connection Tester offers visualization of the measurements extracted from the synchrophasor data stream. It supports both TCP and UDP and it is capable of sending commands to the PMU under test, which was in this case PMU simulator script. The simulator listens for the command frames at the preconfigured network port. The Connection Tester first sends a command to request the configuration, then after receiving the configuration frame from the simulator, it sends a command to tell the simulator to start sending synchrophasor data.

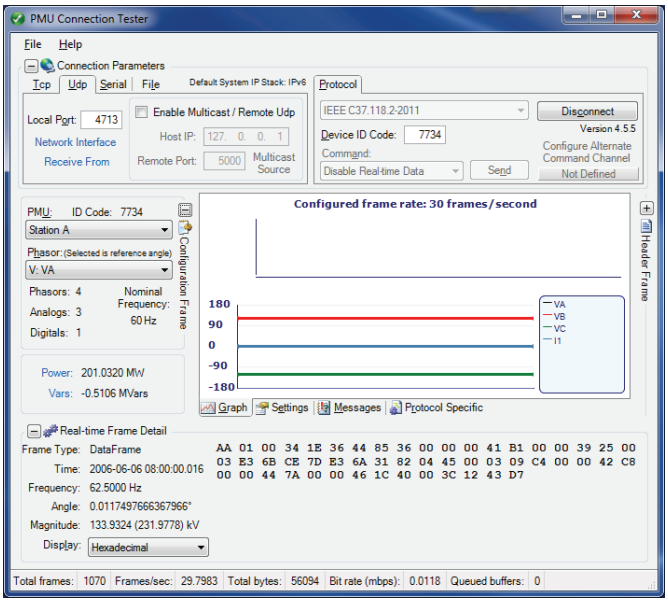

Fig. 9. Visualization of the received synchrophasor data using PMU Connection Tester.

Fig. 9 illustrates the visualization of the received synchrophasor data using PMU Connection Tester. The data frame contains information on four phasors – three voltages and one current. The phasor values in the figure do not change since the data fames contained fixed pre-defined values. The same results and behavior were observed when compared to the use of iPDC PMU simulator.

These two test scenarios illustrate the validation process of the implementation of the communication standard using Python. The PMU simulator allows for creation of invalid data frames as well. This feature is useful for using the simulator as a part of synchrophasor testbed to evaluate behavior of PDCs in the synchrophasor system.

5. CONCLUSIONS

This paper describes software implementation of PMU simulator as Python module. The implementation allows creation of various Python scripts that can be used as synchrophasor testbed support tools. The core module implements C37.118 communication protocol and provides API for easy creation and manipulation of message data frames. The

The main contributions of the paper can be summarized as follows:

- Python implementation of the C37.118 communication protocol, which is a cross-platform and simple to use in custom Python scripts.
- The library allows user to create Python scripts that can be used in various synchrophasor related test and evaluation scenarios. The paper discusses implementation of PMU simulator capable of sending realistic PMU data streams.

- The PMU simulator has been evaluated using third party tools Wireshark, PMU Connection Tester, and iPDC PMU simulator.
- Further research may include creation of multiple PMU simulator instances, time synchronization and interfacing to GPS clocks, implementation of PMU data stream splitter, as well as various PDC functionalities.

REFERENCES

[1] IIEEE Std C37.118.1, "*IEEE Standard for Synchrophasor Data Transfer for Power Systems*,", IEEE Power Engineering Society, 2011.
[2] IEEE Standard, *"Test Suite Specification: IEEE Synchrophasor Test Suite Specification - Version 2," 2015, pp. 1-43.*
[3] I. C. Decker, M. N. Agostini, A. S. e Silva, and D. Dotta, *"Monitoring of a large scale event in the Brazilian power system by WAMS,"* in Proc. VIII iREP, 2010, pp. 1–8.
[4] G. Heydt, M. Kezunovic, P. Sauer, A. Bose, J. McCalley, C. Singh, W. Jewell, D. Ray, and V. Vittal, *"Professional resources to implement the Smart Grid, in North American Power Symposium (NAPS),"* 2009, 2009, pp. 1–82011.
[5] P. Sauer, *"Educational needs for the Smart Grid workforce,"* in Power and Energy Society General Meeting, 2010 IEEE, 2010, pp. 1–3.
[6] S. Šandi, T. Popović, B. Krstajić, *"Alati za podršku mjerenju sinhrofazora,"* Žabljak, 2014.
[7] E. National Electric Sector Cybersecurity Organization Resource, *"Wide Area Monitoring, Protection, and Control Systems (WAMPAC),"* 2012.
[8] IEEE PowerEngineering Society*, IEEE Std C37.118.1, "IEEE Standard for Synchrophasor Measurements for Power Systems"*, 2011.
[9] IEEE Standard 1588-2002, *"IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems,"* 2002.
[10] https://msdn.microsoft.com/en-us/library/windows/desktop/dn553408(v=vs.85).aspx, last access: 5.10.2015.
[11] A. G. Phadke, *"Synchronized phasor measurements in power systems,"* IEEE Comp. Appl. Power, vol. 6, 1993, pp. 10–15.
[12] A. G. Phadke and J. S. Thorp, *"Synchronized phasor measurements and their applications,"* New York: Springer, 2008.
[13] IEEE Power Engineering Society, *IEEE Std C37.118.2, "IEEE Standard for Synchrophasor Data Transfer for Power Systems,"* 2011.
[14] A. J. Stadlin. (2013, Out) *"Gridtrak open-source synchrophasor PMU project," http://gridtrak.codeplex.com,* last access: 5.10.2015.
[15] D. Dotta, J. H. Chow, L. Vanfretti, M. S. Almas, and M. N. Agostini, *"A MATLAB-based PMU simulator*," IEEE PES General Meeting 2013, 2013.
[16] OpenPDC, http://openpdc.codeplex.com, last access: 5.10.2015.
[17] http://wiki.wireshark.org/IEEE%20C37.118, last access: 5.10.2015.
[18] https://pmuconnectiontester.codeplex.com/, last access: 5.10.2015.
[19] http://en.wikipedia.org/wiki/Python_%28programming_language%29, last access: 5.10.2015.
[20] https://www.python.org/downloads/windows/, last access: 5.10.2015.
[21] http://en.wikipedia.org/wiki/Python_Package_Index, last access: 5.10.2015.