Linköping University | Department of Computer and Information Science
Bachelor's thesis, 16 ECTS | Software Development
2023 | LIU-IDA/LITH-EX-G--23/001--SE

Improvement of simulation software for test equipment used in radio design and development

Förbättring av simuleringsmjukvara av testutrustning för modern radio design och utveckling

Vincent Ahlström

Supervisor : Suleman Khan
Examiner : Andrei Gurtov

External supervisor : Gustav Eklund & Markus Tollet

Abstract

The global engineering design house Syntronic has requested a further development of the open source Python framework PyVISA-sim to enable dynamic simulations of signals and measuring instruments which would streamline development of their internal radio equipment testing tool. This tool is used by a world leading telecommunications company when developing their next generation radio equipment. PyVISA-sim is used in lab environments to test applications without access to real connected instruments. The project detailed in this thesis strives to pinpoint Syntronic's needs, develop the requested functionality within the framework and have the changes implemented as part of the official GitHub repository, thereby making them available for anyone wanting to utilize them. To achieve this agile software development methods are utilized combined with an open source mindset. The resulting additions to PyVISA-sim can reduce the workload for all users in need of a more complex simulation method.

Acknowledgments

I would like to express my gratitude to the supervisors from Syntronic, Gustav Eklund and Markus Tollet, for helping shape this thesis and answering my many questions along the way. I also want to extend my sincere thanks to my university supervisor Suleman Khan and examiner Andrei Gurtov who has guided me through the writing process. Finally I want to acknowledge Fredrik Hörnqvist, the team manager at Syntronic, without whom this endeavor would not have been possible.

Contents

Abstract **iii**

Acknowledgments **iv**

Contents **v**

List of Figures **vii**

1 Introduction **1**
1.1 Motivation 1
1.2 Aim 1
1.3 Research questions 1
1.4 Delimitation 2

2 Background & related work **3**
2.1 PyVISA in the laboratory 3
2.2 PyVISA at Syntronic 4
2.3 Radio development 5
2.4 Related work 6

3 Theory **8**
3.1 Development principles 8
3.2 Code standard 8
3.3 Open source 10
3.4 VISA 12
3.5 PyVISA 12
3.6 PyVISA-sim 12
3.7 Testing 13

4 Method **15**
4.1 Pre-study 15
4.2 Implementation 18
4.3 Evaluation 21

5 Results **22**
5.1 Pre-study 22
5.2 Implementation 22
5.3 Evaluation 22

6 Discussion **24**
6.1 Results 24
6.2 Method 24
6.3 The work in a wider context 25

6.4 Future work 26

7 Conclusion **27**

Bibliography **29**

List of Figures

2.1 Using an instrument with PyVISA . . . 3
2.2 Typical test bench used with PASS . . . 4
2.3 NGMN 5G Design Principles [1] . . . 5
2.4 Prominent 6G applications [16] . . . 6
2.5 Typical Robot Framework module stack [14] . . . 7

3.1 Mypy typing conversion [13] . . . 10
3.2 Simplified pull request process [29] . . . 11
3.3 Device in yaml-file . . . 13

4.1 Requirement specification . . . 16
4.2 Connections property in YAML . . . 16
4.3 Component class . . . 17
4.4 Property class . . . 17
4.5 Flowchart of query process . . . 18
4.6 Flowchart of query process, 2nd iteration . . . 19
4.7 Connection in dialogues . . . 20
4.8 Flowchart of query process, 3rd and final iteration . . . 20

1 Introduction

1.1 Motivation

Syntronic is a global engineering design house developing a simulation software for the automation and testing of modules for the development of next-generation radio products. During development of the simulation software, they do not always have access to the physical instruments which are necessary to perform the simulations. The simulation software is used to simulate advanced measurement tools before testing them in a lab environment. However, the current simulations are limited to pre-defined static responses and do not mimic real-world behaviour for all use cases, which hinders the development process. The current simulations also only work for some of the instruments, limiting the use cases for the software. Additionally, most instruments currently return a default "error" message, which could be improved upon. Therefore, Syntronic requires a better framework to streamline the development process and improve the functionality of the simulations.

1.2 Aim

The aim of this thesis is to develop a Python framework for dynamic simulations of signals and measuring instruments. The framework will be built by further developing the PyVISA-sim framework and enabling integration with existing Syntronic software.

The framework should fulfill the demands of Syntronic while, in the spirit of open source, staying relevant for other users. This can be achieved by utilizing established open source software development methods.

1.3 Research questions

- How can development of the automatizing/testing tool be streamlined, utilizing a dynamic simulation tool?
- What needs to be implemented to make this tool sufficient for Syntronic's use cases?
- Is it possible to make the requested functionalities suitable for the current framework?

1.4 Delimitation

The software will further develop the open-source software PyVISA-sim, written in Python, which is what the current framework is based upon. This will ensure maximum compatibility with the existing code base. The main users are software developers at Syntronic, accustomed to the current framework. Interaction should therefore strive for consistency with the original version. The framework should also remain viable for other users.

There will not be a focus on specific instruments and how these are simulated. This would take up too much time and demand a deeper knowledge of physics than what I posses. The thesis will instead try to create broad solutions allowing for many use cases.

2 Background & related work

2.1 PyVISA in the laboratory

PyVISA provides a uniform interface to communicate with a variety of different types of test and measurement instruments, including oscilloscopes, function generators and multimeters, among others [23]. It allows the user to control these instruments and read data from them using Python scripts, which can be particularly useful in a laboratory setting where the user wants to automate measurements or perform data analysis from a distance. An example of how PyVISA might be used in a laboratory is found in Figure 2.1. In this example, PyVISA is used to connect to an instrument with a GPIB address of 12, reset the instrument, configure it to measure DC voltage, take a measurement and then disconnect from the instrument.

```python
import pyvisa

# Connect to the instrument

rm = pyvisa.ResourceManager()
instrument = rm.open_resource('GPIB0::14::INSTR')

# Set up the instrument

instrument.write('*RST') # Reset the instrument
# Configure the instrument to measure DC voltage
instrument.write('CONF:VOLT:DC 10,0.1')

# Take a measurement

measurement = instrument.query('MEAS:VOLT:DC?')
print(measurement)

instrument.close() # Disconnect from the isntrument
```

Figure 2.1: Using an instrument with PyVISA

However, PyVISA has its own limitations [23]. One of the main limitations of PyVISA is that it only provides a virtual representation of the instruments. This can limit its ability to fully replicate the behavior of the actual instruments, especially in cases where the instruments are complex or have advanced features. Another limitation is that it can only interact with the instruments that it has drivers for, meaning that it is not compatible with all types of instruments.

2.2 PyVISA at Syntronic

Syntronic uses PyVISA in their proprietary software, named PASS, which they have been developing since 2016. PASS stands for Power Amplifier Standalone test System and is designed for PA measurement. By using PASS, researchers need no knowledge of SCPI (instrument commands) to control instruments while performing tests. Instead, Pass offers a GUI where all steps of the process, from instrument setup to graphing of the results, can be achieved with pre-written scripts. A typical PASS test bench setup can be found in Figure 2.2. This setup combines two signals, runs them through the DUT and measures the outgoing signals.

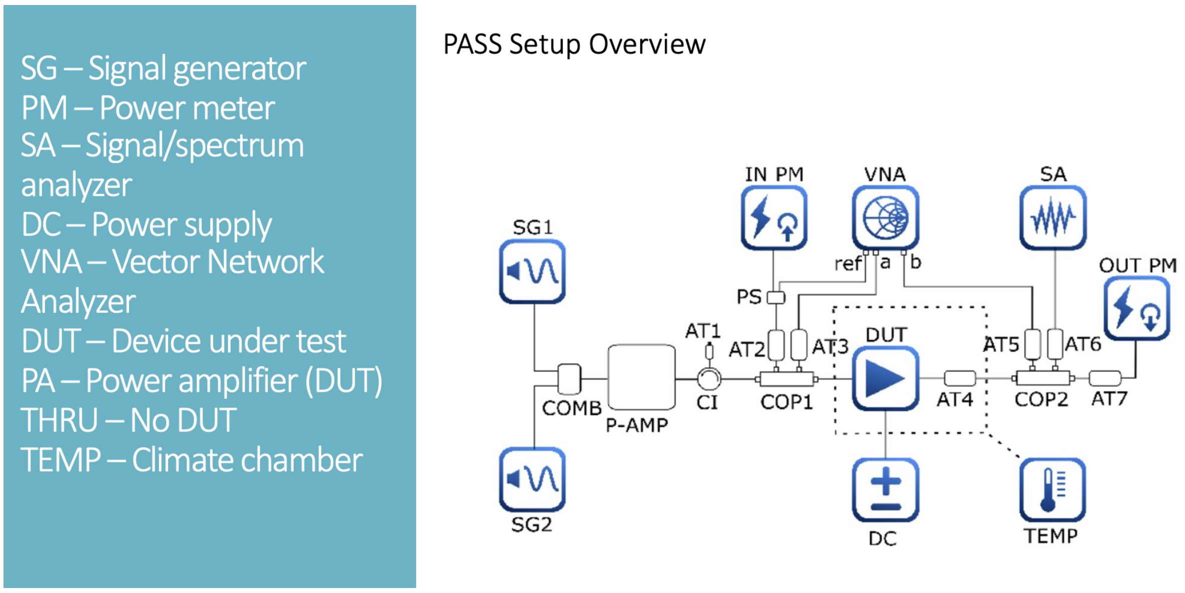


Figure 2.2: Typical test bench used with PASS

Syntronic currently do not have access to all required instruments when developing PASS, instead they use the framework PyVISA-sim to simulate a simplified version of the missing instruments. The static nature of these simulations make them very crude and not optimal when testing measurements. When, for example, doing PA measurement there is a need to dynamically change settings to get a resulting curve that resembles a realistic measurement.

The reason for not having access to the physical instruments is that the developers are usually working remotely. In order for the development process to be efficient they therefore need to run tests by performing the before mentioned simulations. They also share instruments with other departments which means that the instruments may not always be available even when the developers are working in the laboratory.

2.3 Radio development

The main purpose of PASS is to assist the telecommunications market in developing the next generation of radio products used for high speed internet connection.

Better internet connectivity is essential for a wide range of important areas including economic growth, education, healthcare, social inclusion, public safety, innovation and quality of life. High-speed internet enables businesses to reach new customers, expand their operations and provides entrepreneurs with new opportunities to start their own business.

It also improves educational outcomes, and provides opportunities for students in remote and underprivileged areas by allowing access to educational resources from anywhere [12]. Remote health services and telemedicine can be provided with better internet connection, which can help improve healthcare access and quality, especially in rural and remote areas. Internet access is a key factor in bridging the digital divide, which can provide underprivileged communities with access to information and opportunities that they would otherwise be unable to reach. Better internet connection also plays a crucial role in public safety and emergency management, allowing emergency services to respond quickly to situations and providing real-time data to help with decision-making. Additionally, high-speed internet is essential for research, innovation and development of technology like 5G and IoT.

It also improves people's quality of life by providing access to entertainment, social media and other online services that can be used for leisure, learning and personal growth. Therefore, it is important to continue investing in expanding and improving internet infrastructure to ensure that everyone has access to high-speed internet.

As systems and products get more interconnected with the development of 6G, it is vital to have clear design rules. The Next Generation Mobile Networks Alliance has outlined some design principles, concerning 5G networks, visable in Figure 2.3. These are described by Ahmad et al. to "highlight the need for highly elastic and robust systems" [1]. They further write: "The radio part needs extreme spectrum efficiency, cost-effective dense deployment, effective coordination, interference cancellation and dynamic radio topologies".

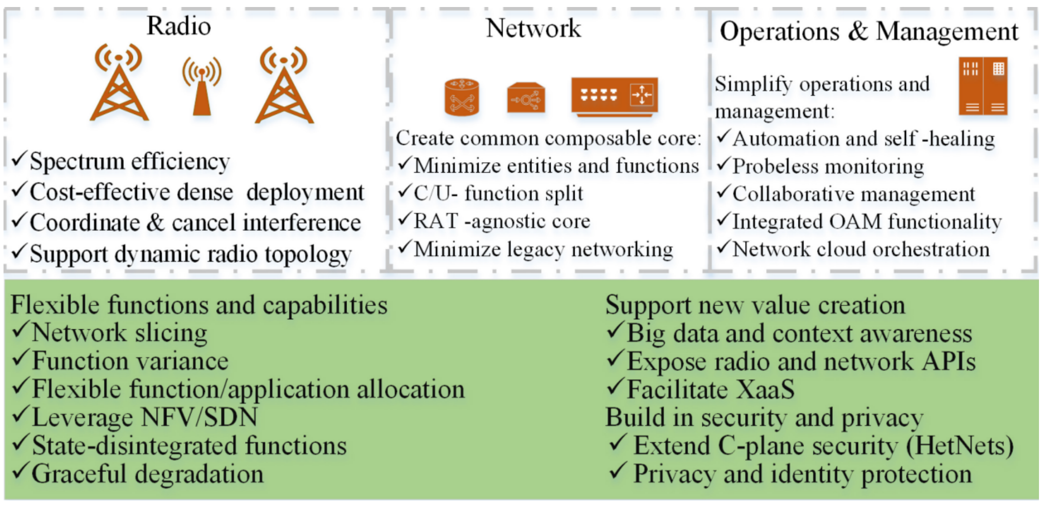

Radio
✓Spectrum efficiency
✓Cost-effective dense deployment
✓Coordinate & cancel interference
✓Support dynamic radio topology
Network
Create common composable core:
✓Minimize entities and functions
✓C/U- function split
✓RAT -agnostic core
✓Minimize legacy networking
Operations & Management
Simplify operations and management:
✓Automation and self -healing
✓Probeless monitoring
✓Collaborative management
✓Integrated OAM functionality
✓Network cloud orchestration
Flexible functions and capabilities
✓Network slicing
✓Function variance
✓Flexible function/application allocation
✓Leverage NFV/SDN
✓State-disintegrated functions
✓Graceful degradation
Support new value creation
✓Big data and context awareness
✓Expose radio and network APIs
✓Facilitate XaaS
Build in security and privacy
✓Extend C-plane security (HetNets)
✓Privacy and identity protection

Figure 2.3: NGMN 5G Design Principles [1]

These needs will only grow as the world fully adopts the 5G networks and starts moving towards 6G. According to Porambage et al., New radio equipment will need to be developed that can handle new applications such as "UAV based mobility, Connected Autonomous Vehicles (CAV), Smart Grid 2.0, Collaborative Robots, Hyper-Intelligent Healthcare, Industry 5.0, Digital Twin and Extended Reality" [16]. These needs and more are visualized in Figure 2.4

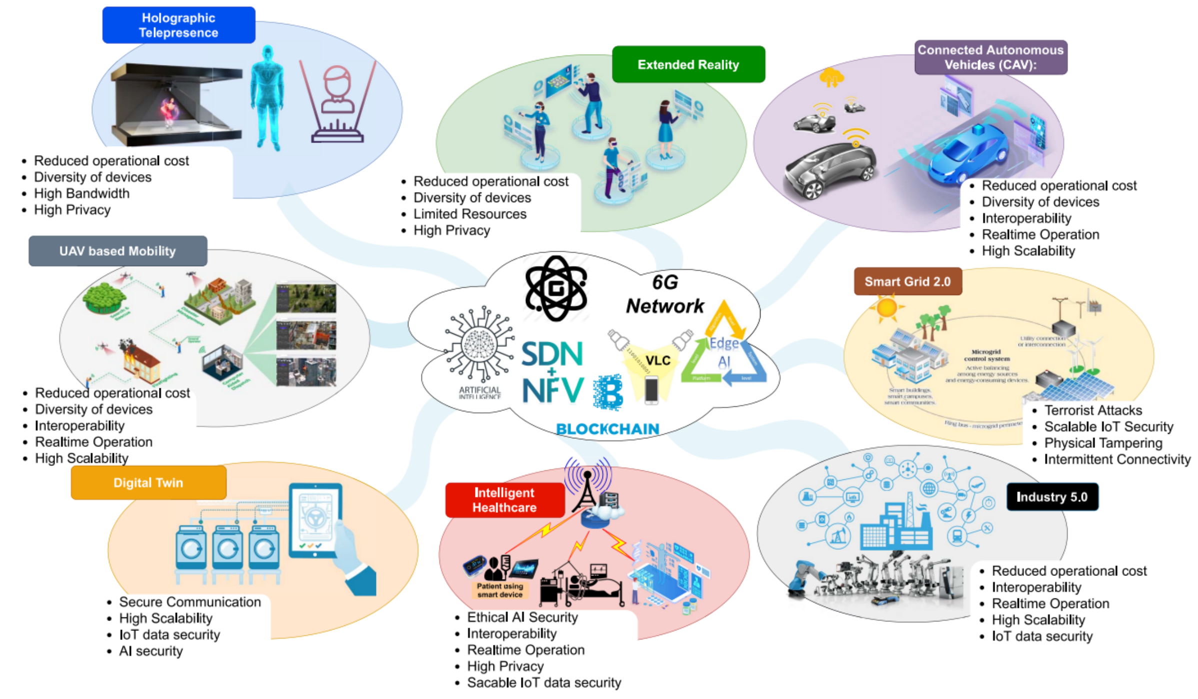


Figure 2.4: Prominent 6G applications [16]

2.4 Related work

In the thesis *RF performance testing using Robot Framework*, commissioned by Oy LM Ericsson AB, J.Ollanketo investigates a framework for testing RF conformance in small cell base station development, that could be an alternative to Syntronics testing software [14]. This framework is also using PyVISA and requires physical access to the measurement tools.

The Robot Framework is a generic open-source automation framework for acceptance testing, acceptance test-driven development (ATDD) and robotic process automation (RPA) [14]. It is a popular tool for automating test cases and can be used for a wide range of applications, including RF performance testing. The Robot Framework provides a simple and easy-to-use interface for creating test cases and running automated tests, making it well-suited for RF performance testing.

Using Robot Framework for RF performance testing, simplifies the process of automating the testing process and makes it more efficient [14]. It can be used to automate repetitive tasks such as configuring test equipment and collecting data, and the test cases can be easily modified to adapt to changes in the testing requirements. Additionally, the Robot Framework can be integrated with other tools such as continuous integration platforms, making it easy to automate the testing process and provide real-time feedback on test results.
In Figure 2.5, J.Ollanketo demonstrates a typical Robot Framework module stack which includes PyVISA and a VISA backend which is provided by PyVISA-sim when when used for

simulations [14]. This project would likely suffer from the same limitations as PASS due to the limitations of PyVISA-sim.

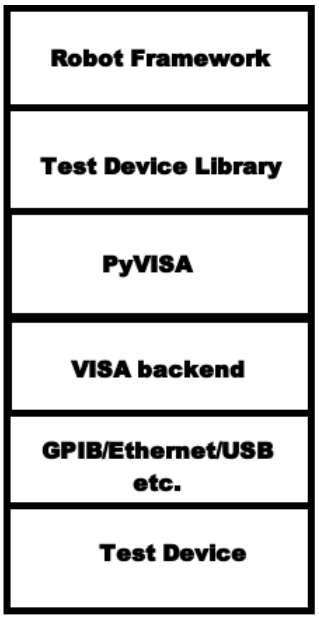


Figure 2.5: Typical Robot Framework module stack [14]

J.Ollanketo concludes that the Robot framework could replace their current testing system at the cost of a major rework [14]. The implementation would bring more flexibility with simulations and a centralized database for test place definitions. There is no mention regarding how the static simulations would impact any testing.

3 Theory

3.1 Development principles

There are many approaches to software development. This project will use an agile approach due to the loose constraints of the research questions. There might be challenges that were not apparent from the start. Agile software development is a methodology that is based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams [3]. This is accomplished by working in close proximity and having oral discussions instead of communicating via writing.

The focus of this approach is the ability to deliver working software quickly and to respond to changes in customer requirements as they arise. This stems from the more traditional "waterfall" methodology where the entire process is planned out in advance and executed in a linear, sequential fashion [11]. Agile methods are designed to help teams adapt to changing circumstances and deliver high-quality, working software more efficiently.

3.2 Code standard

PyVISA-sim combines multiple tools and code standards to keep the general code quality high. Many of these are pillars of the Python community, maintaining a uniformity among open source development projects. Following a Python code standard, such as PEP 8, can help make code more readable and consistent, which makes it easier for other people to understand and work with.

Consistency in how code is written and formatted can help reduce the amount of mental effort required to understand the code, since the reader does not need to spend as much time trying to figure out the conventions used. This can be especially important when working on a team, or when other people will need to read and maintain the code in the future [2].

In addition, following a well-established code standard can make it easier to share code and collaborate with others, since it helps to ensure that everyone is using the same conventions. This can make it easier to merge changes and avoid conflicts.

It is also helpful when maintaining the code in the future. As the code base grows, and you work on different parts at different times, it can be difficult to remember all of the conventions used. Following a standard can help to ensure that your code remains consistent even as it evolves over time.

Following codes standards increases the chance of your code being accepted and understood by others. This might be as part of open source community, a company or in your own personal projects [7].

In short, following a Python code standard can help make code more readable, consistent and maintainable, which saves time and effort in the long run.

PEP 8

PEP 8 is a set of coding standards for Python programs, written by the Python community [30]. It provides a set of guidelines and best practices for writing Python code, with the goal of making code easier to read, understand and maintain. Some of the key points covered by PEP 8 include:

- Use 4 spaces for indentation, rather than tabs.
- Limit all lines to a maximum of 79 characters.
- Use blank lines to separate functions, class definitions and to break up large blocks of code.
- Use inline comments sparingly and put them on a separate line from the code they are commenting.
- Use descriptive variable names and avoid abbreviations.
- Put imports at the top of the file and group them in the following order: standard library imports, third-party library imports and local application imports.

Black

Black is a PEP 8 compliant opinionated formatter for Python [27]. It is designed to simplify writing code that is easy to read and understand, by automatically formatting code in a way that follows the Python style guide.

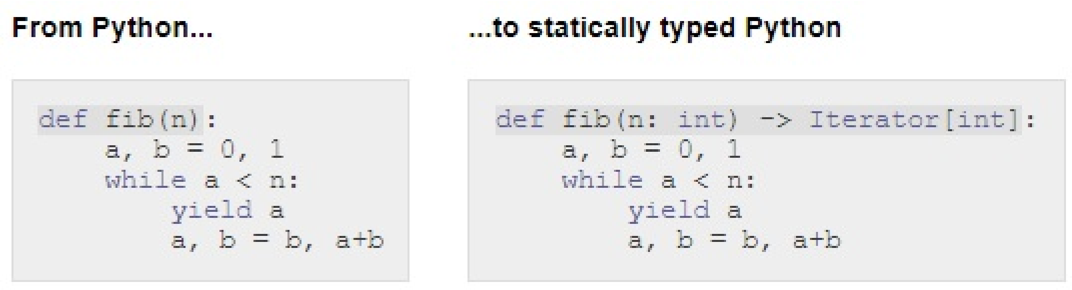
Black reformats Python code to follow a consistent style, with no configuration required [26]. It can be run on an entire codebase, or on individual files, and it can be integrated with code editors and version control systems.

Flake8

Flake8 is a Python tool that checks code for style and formatting errors, as well as for potential bugs and other issues based on a set of rules [6] [25]. It is a wrapper to several other tools, including PyFlakes [18], pycodestyle [17] and Ned Batchelder's McCabe script [10], and it can be used to enforce a consistent coding style in a project [5].

Mypy

Mypy is a static type checker for Python [13]. It is a tool that checks Python code for type errors and it assists with catching type-related bugs in code before being run. Mypy controls type annotations according to PEP 484 [15]. For an example see Figure 3.1.



```
def fib(n):
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

```
def fib(n: int) -> Iterator[int]:
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

Figure 3.1: Mypy typing conversion [13]

3.3 Open source

General

Open source software development is a type of software development where the source code of the software is made freely available to the public. This means that anyone can access, modify and distribute the software, subject to certain terms and conditions outlined in the open source license.

One of the main advantages of open source software is that it allows for collaborative development, where multiple individuals and organizations can work together to improve and evolve the software. This can lead to a faster pace of development, more robust and feature-rich software, and more diverse contributions from a global community of developers [4].

Open source development is usually done through communities and in most cases, developers work on the project voluntarily. The main goal is to create software that is transparent, accessible and adaptable. [4]

While open source software development has many advantages, there are also some potential disadvantages to consider. Open source software is often developed and maintained by a community of volunteers, and there may not be a dedicated support team in place to help users with technical issues. This can make it more difficult to get help if you encounter a problem or bug. Also, because open source software is developed by volunteers, it can be more difficult to secure funding and resources to support development and maintenance of the software. This can lead to slower development and fewer features being added over time [4].

The health and vitality of an open source project depends largely on the community that supports it. While many projects have large and active communities, others have small or inactive communities, which means that the project may not be well maintained and it could be difficult to obtain support [7].

GitHub

When contributing to open source software using GitHub, some aspects differ from proprietary development. GitHub recommends publicly approaching the current maintainer before writing any code [7]. Not all work will be accepted and the project might be heading in another direction than first perceived by the contributing developer. A good start is opening an issue describing the problem or idea you have encountered in detail. A GitHub issue is a place where you can post information and have a corresponding conversation. These can later be referenced when starting work in a pull request. The maintainers might have relevant input before you start work on your implementation. Not all features will be accepted to the main project. There is always the option to make a new fork which you can maintain by yourself if you have differing views on the final decision of the maintainer.

The next step is opening a pull request if your issue gets positive feedback [7]. The community might have more suggestions or feedback and this is a good forum for such discussion. Whenever you update your repository the pull request will be updated showcasing the changes. Creating it early in the process is good for transparency. The pull request process, when merging from a different branch, is visualized in Figure 3.2 from the Tuleap project. This can also be done, as in our case, from a fork of the project. When new code is pushed the pull request will be updated showing all changes made.

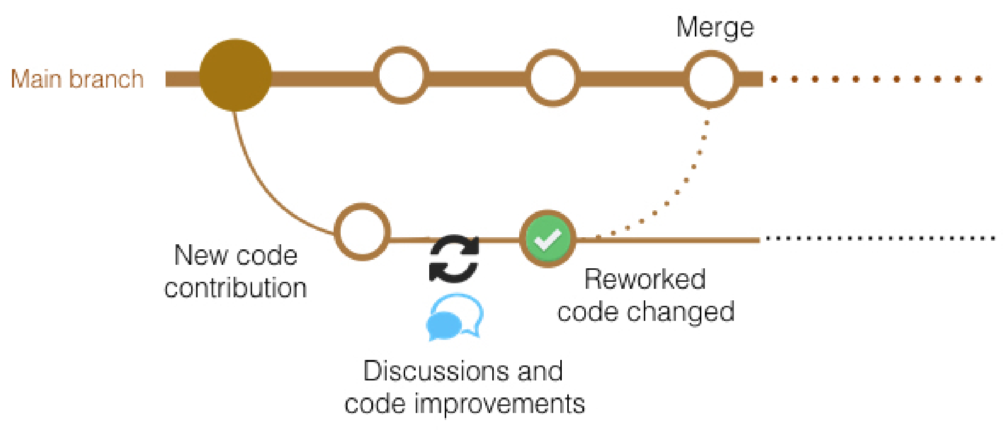


Figure 3.2: Simplified pull request process [29]

The development work can be done either as a branch of the project or as a fork. To create a new branch you need access to be part of the main repository. Everyone can create a new fork of an open source project where they can make any change wanted.

When finalised and ready to merge, there are three possible upcoming scenarios [7].

1. You are asked to make some changes. This is an important part of the open source development process to keep the code clean and consistent.
2. The contribution is not accepted. The maintainer always has the last word and everyone can not always agree, especially on bigger projects.
3. The pull request is accepted. The new code merges from your fork into the main repository and becomes a part of the open source software.

3.4 VISA

VISA is an abbreviation of Virtual Instrument Software Architecture which is a standard created by the VXIplug&play Systems Alliance, now part of the IVI Foundation, used for communication between measuring instruments and computers [28]. It was created as part of a bigger effort in reducing development costs for their end-users. VISA defines a common set of commands and protocols for communicating with instruments from different manufacturers. Expanding further on VISA would go beyond the scope of this study as no real VISA functionality will be used when simulating the instruments.

3.5 PyVISA

PyVISA is a Python package that provides a high-level interface to the National Instruments VISA library [23]. PyVISA makes it easier to use VISA in Python by providing a simple, Pythonic interface that hides much of the complexity of the VISA library. PyVISA is commonly used to automate tests and measurements, as well as for controlling scientific instruments in general.

PyVISA is particularly useful in a laboratory setting where users want to automate measurements or perform data analysis from a distance [23]. Using PyVISA, users can easily create Python scripts that connect to an instrument, configure it to perform a specific measurement, acquire data and perform data analysis. For example, PyVISA can be used to connect to an instrument with a GPIB address, reset it, configure it to measure DC voltage, take a measurement and then disconnect from the instrument.

In summary, PyVISA is a powerful tool that allows users to control and communicate with test and measurement instruments using Python scripts [23]. It is widely used in laboratory settings, where users need to automate measurements and perform data analysis, but it can also be used for instrument simulation.

3.6 PyVISA-sim

PyVISA-sim is one of multiple back end frameworks built for PyVISA to add new functionalities. The frameworks purpose is to enable simulations of devices in order to test applications without physical access [22]. The simulated devices and their behavior are kept in a YAML-file [31], each device consisting of the following components:

- **Eom**, which stands for "end of message" and instructs the computer how to parse entered commands.
- **Error**, which lets the user set a default error message. This component can also be expanded to differentiate a command error and a query error.
- **Dialogues**, which allow the user to simulate communication with the device. Each dialogue is made up of a query and an optional response. If no response has been given there will be no response from the device. No other data is read in a dialogue.
- **Properties**, which can be viewed as a dialogue with memory. Each property has a standard value, a setter, a getter and some specs. The specs define the property type and value limits.

A YAML device is visualized in Figure 3.3

```yaml
device 1:
    eom:
      ASRL INSTR:
        q: "\r\n"
        r: "\n"
      GPIB INSTR:
        q: "\n"
        r: "\n"
    error: ERROR
    dialogues:
      - q: "?IDN"
        r: "LSG Serial #1234"
      - q: "!CAL"
        r: OK
    properties:
      frequency:
        default: 100.0
        getter:
          q: "?FREQ"
          r: "{:.2f}"
        setter:
          q: "!FREQ {:.2f}"
          r: OK
          e: 'FREQ_ERROR'
        specs:
          min: 1
          max: 100000
          type: float
```

Figure 3.3: Device in yaml-file

However, it has some limitations [22]. One is that it has limited instrument simulation capabilities and may not support all the functionality of real instruments. Another is that PyVISA-sim may not support all the functionality provided by the VISA library such as event handling and advanced VISA functionality. Additionally, PyVISA-sim is a Python library, so it may not be compatible with other programming languages or platforms. Furthermore, the simulation may not closely match the real instrument behaviour. Also, since not all instrument vendors have been implemented, the developer will have to create new resources to add additional ones.

3.7 Testing

PyVISA-sim has an extensive collection of test cases utilizing the Python framework Pytest. It is used to write and run unit tests, and supplies a set of features for running and managing the tests [19]. Some of the key features of Pytest include the ability to run tests in parallel, support for test parametrization and a built-in mechanism for collecting and reporting test results.

Unit testing is a software testing method where individual units or components of a software are tested in isolation [9]. The purpose of unit testing is to validate that each unit of the software is working as intended and meets the specified requirements. This is typically done by writing test cases that exercise the individual units and verify their behavior. Unit tests are ideally written before writing the code being tested [8]. By having a strong test foundation it is easier to stay focused on the purpose of the code. If the tests are being utilized regularly during development new bugs are easily caught when introduced.

4 Method

4.1 Pre-study

Setting up development environment

I was given a computer with Windows 11 installed to use for this project. Syntronic is developing PASS using Python version 3.6 so I decided to use it as well to ensure compatibility. To install PyVISA-sim I followed the official installation guide [20] using the Python pip package-management system. I used Microsoft's Visual Studio Code as main code editor for the project. I was also given the YAML-file Syntronic used with PASS.

Requirement specification

In order to understand more precisely what Syntronic's expectations for this thesis was I requested a requirement specification. I was given the following list and the image in Figure 4.1.

- Motivation: Simulate a Device Under test (DUT) from lab instruments point of view.
- Clear separation between PyVISA-sim and Syntronics software named PASS (see Figure 4.1.
- Use case: Power sweep, a standard measurement on a Power Amplifier (PA).
- Technical Specs:
  - Simulate a DUT, which may be characterized by its output given a specific input. (e.g., input/output power and dependencies on other quantities such as input frequency)
  - Simulate the signal path to instruments, introducing measurement artefacts such as frequency dependent offsets in couplers
  - Simulate measurement configurations of instruments such as calibrations, frequencies, bandwidths and offsets
  - Enable the instrument-to-instrument communication necessary to simulate the DUT

– User interaction with PyVISA-sim via YAML should be expanded to allow for more flexible input such as user defined functions (currently static responses are supported).

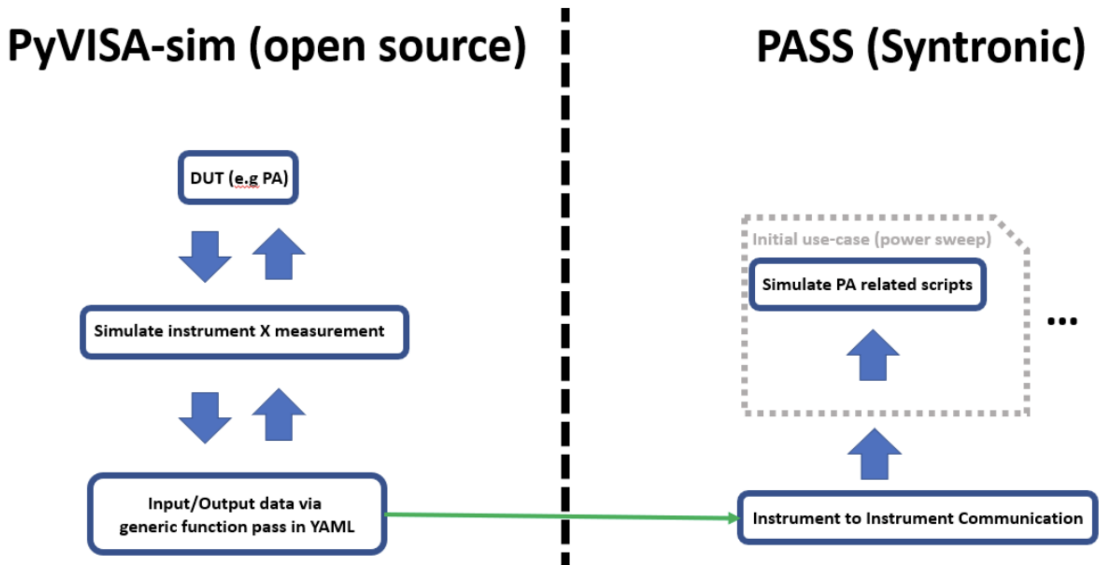

PyVISA-sim (open source)
PASS (Syntronic)
DUT (e.g PA)
Simulate instrument X measurement
Input/Output data via generic function pass in YAML
Initial use-case (power sweep)
Simulate PA related scripts
...
Instrument to Instrument Communication

Figure 4.1: Requirement specification

From discussions, I, together with the team, decided to implement a connections property to the devices found in the YAML-file. See Figure 4.2

```
Connections:
    power:
        q: "READ1"
        source: GPIB0::2::INSTR
        source_parameter: amplitude
        function: 0.23 * (amplitude ** 2) - 3.4
```

Figure 4.2: Connections property in YAML

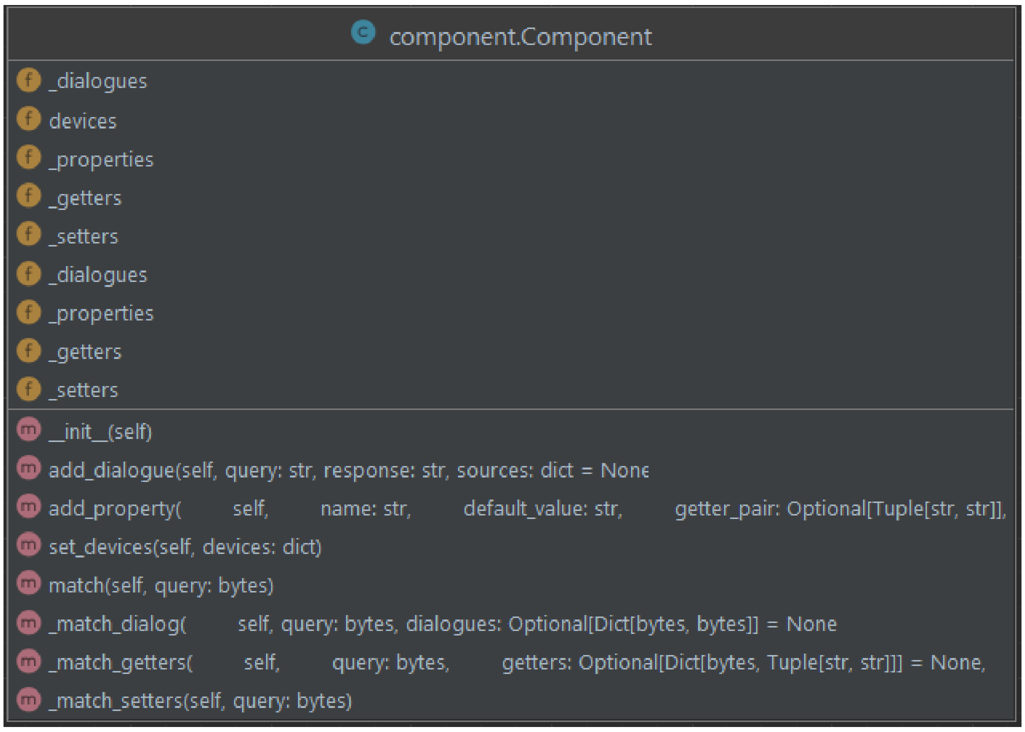
With this we hoped to implement instrument to instrument communication using functions too simulate measurements.

Understanding the PyVISA-sim framework

The team of developers at Syntronic could not provide an insight into how the PyVISA-sim framework functioned. The current simulations either used the dialogue system or made calls to properties, but there was no deeper understanding as to how the commands were parsed and how one could access the current state of a simulated instrument.

I wanted to visualize the frameworks classes so I used the diagram package found in the IntelliJ code editor to generate UML class diagrams. Two classes caught my interest, the Component class and the Property class.

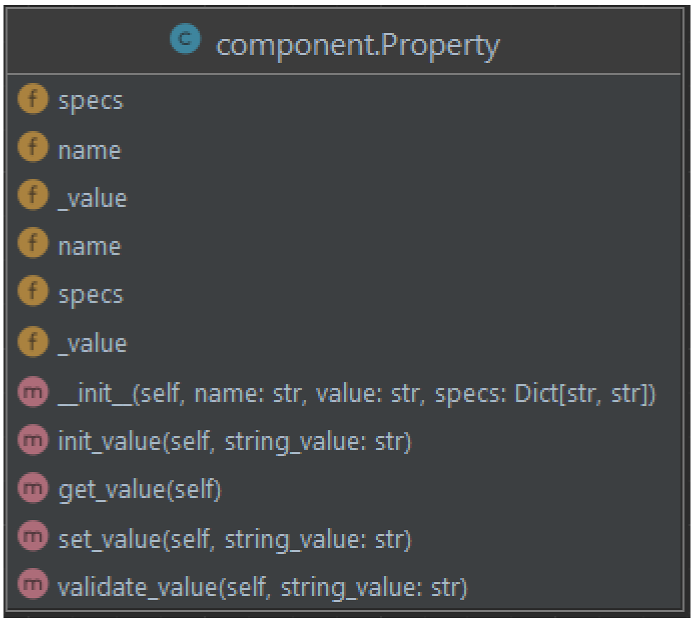
The Component class, see Figure 4.3, stores all information about an instrument found in the YAML-file. This is also where the matching happens when a query is sent. This would be a suitable place to save connections to other instruments.



```
component.Component
_dialogues
devices
_properties
_getters
_setters
_dialogues
_properties
_getters
_setters
__init__(self)
add_dialogue(self, query: str, response: str, sources: dict = None
add_property(     self,     name: str,     default_value: str,     getter_pair: Optional[Tuple[str, str]],
set_devices(self, devices: dict)
match(self, query: bytes)
_match_dialog(     self, query: bytes, dialogues: Optional[Dict[bytes, bytes]] = None
_match_getters(     self,     query: bytes,     getters: Optional[Dict[bytes, Tuple[str, str]]] = None,
_match_setters(self, query: bytes)
```

Figure 4.3: Component class

The Property class, see Figure 4.4, stores values of the properties found in the instrument. If I want dynamic simulations I would need access to these values.



```
component.Property
specs
name
_value
name
specs
_value
__init__(self, name: str, value: str, specs: Dict[str, str])
init_value(self, string_value: str)
get_value(self)
set_value(self, string_value: str)
validate_value(self, string_value: str)
```

Figure 4.4: Property class

Open source approach

To keep the project in the spirit of open source development, I created an issue in the PyVISA-sim GitHub repository [20], explaining the given task and inquired if this would be appreciated as part of the software. The current maintainer gave a positive response confirming that this indeed would fit in the project scope. After this feedback I created my own fork of the project and a pull request connected to this.

4.2 Implementation

Development

In the original version devices were initiated independently from each other. There was no way for direct device to device communication. In order for connections to work this had to be implemented. To solve this each initiated device was added to an array. Upon initiation this array was also included as a reference providing the required access from within each device.

In the first iteration the connections were implemented by adding a connections property to devices in the YAML-file as decided in the pre-study. A connection class was created to handle this new property. During the initiation of each device, an instance of this class now is created in the same way as the other properties. During the parsing of a query, it will be compared with each connection query. If a match is found the the new logic will be run. A match for the connections source will be fetched using the devices array created before. If found the correct source parameter will be fetched and inserted into the function. Finally the function is run and the result returned as an answer to the query. See flowchart in Figure 4.5.

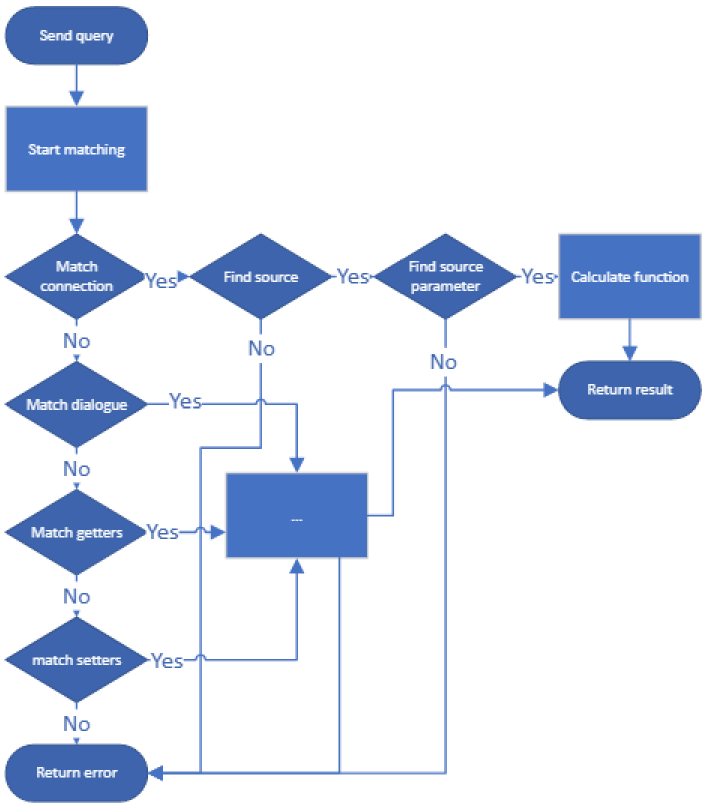


Figure 4.5: Flowchart of query process

With the connection implemented the main task was accomplished. I demonstrated my implementation to the Syntronic team and they confirmed that this addition was working as they intended. As feedback,, two improvements were suggested:

1. Is it possible to fetch multiple parameters in one connection? Perhaps as a list?
2. Is it possible to fetch parameters from the device itself?

The first suggestion was implemented with one change. Instead of a list, a dictionary was used. This allowed for multiple properties coming from different devices with device name as key and property as value.

The second suggestion already worked as the device itself was included in the devices array from which the parameters were fetched. After discussion with the team we decided to implement a feature where, instead of a device name, "self" could be used to indicate that the property came from the device itself.

Flowchart with new additions can be found in Figure 4.6.

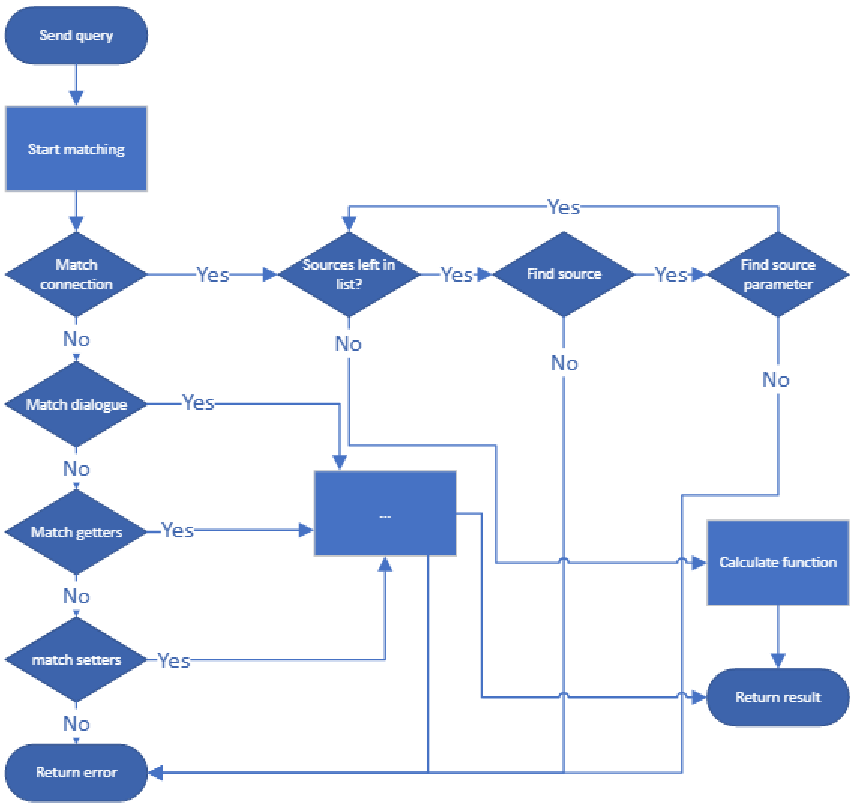


Figure 4.6: Flowchart of query process, 2nd iteration

A set of new tests were added to the already existing tests. This will be delved upon further in the evaluation section.

With these additions the framework had all functionality required by Syntronic. The next step was trying to get my changes merged the official PyVISA-sim repository. The maintainer was supportive of the implementation but suggested some changes. Most were small edits and optimizations but there was one bigger change. He brought up the idea of having connections be part of the dialogues system, and perhaps the setter/getter systems as well. This way of handling it would reduce the amount of new code and lessen the complexity of this new functionality. In my first two iterations I purposefully avoided changing the existing functionality in order to minimize the risk of breaking something.

My third and final iteration implemented all new suggestions including moving connections into an expanded dialogue system. The new way of writing a connection in the YAML-file is shown in Figure 4.7. A flowchart of the last iteration is found in Figure 4.8

```
dialogues:
    - q: "*IDN?"
      r: "HEWLETT-PACKARD, E3632A, 0, 1.2-5.0-1.0"
    - q: "*SYST:ERR?"
      r: "NO ERROR"
    - q: "SOUR1:POW?"
      r: "2"
    - q: "READ1"
      r: "0.23 * ({amplitude} ** 2) - {frequency}"
      sources: [
        {source_name: "GPIB0::2::65535::INSTR",
        source_parameter: amplitude},
        {source_name: "GPIB0::22::65535::INSTR",
        source_parameter: frequency}
      ]
```

Figure 4.7: Connection in dialogues

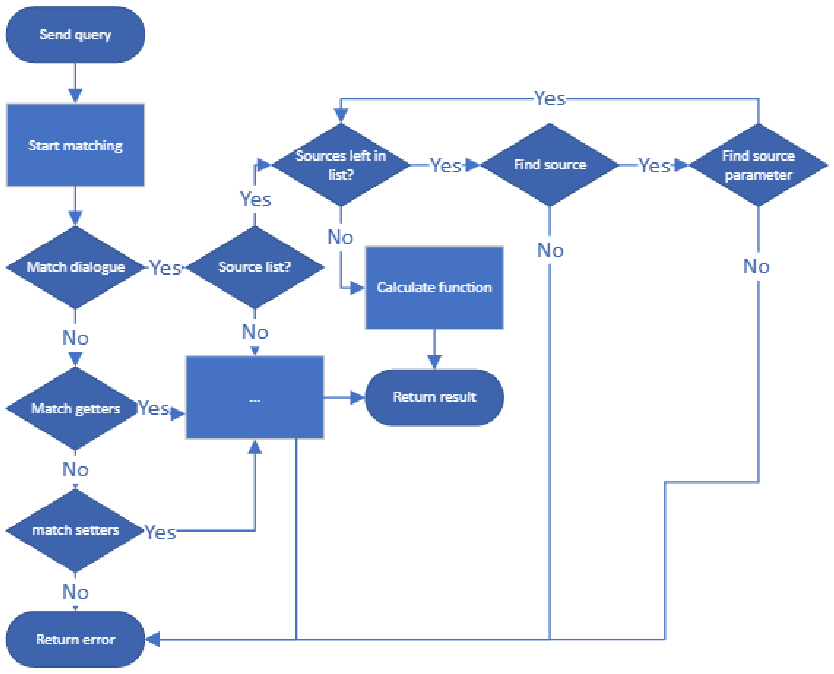


Figure 4.8: Flowchart of query process, 3rd and final iteration

4.3 Evaluation

Testing

During development I used a testing script utilizing the new YAML-file. The connections were tested, then retested after changing a property of the connected device. This made sure that the connection was dynamic and reacted to changes. Before pushing to the pull request these tests were added to the existing set of tests.

Here I encountered a problem were the code passed all tests on my local computer but failed the checks when pushing to the pull request. This was due to me mistakenly not using the latest version of the framework, not because of the code additions. After updating, the code passed all checks.

5 Results

5.1 Pre-study

The pre-study determined that an addition to the already used YAML-file would be viable for creating the simulations. If the framework could recognize a connection property, including a function, and set up connections between devices using this, the Syntronic team were confident it would have the functionality required to simulate what their automatizing/testing tool demanded, thereby streamlining the development process. The YAML-file was already the main way users set up instruments and their properties so this implementation would keep additions as close to the original in functionality as possible. This implementation would also keep PyVISA-sim separated from PASS, maintaining the same abstraction as the original.

When suggesting this new feature in the main PyVISA-sim repository on GitHub, the maintainer confirmed that this would belong within PyVISA-sim.

5.2 Implementation

The final version of the framework fulfilled the demands set by both the team at Syntronic and the maintainer of the official GitHub repository. Without removing any functionality the framework now has a way to, in the YAML-file, provide functions implementing parameters from other initialized components. The connected components are dynamically updated when the connected instruments are changed. Simulations are made by expanding a dialogue property with an internal sources property. This is a list of dictionaries consisting of a name and a parameter (which is referenced in the reply property). The reply must be a function readable by Python.

5.3 Evaluation

As my implementation passes the new unit tests I can conclude that the implementation is sufficient for Syntronics use cases. The expanded dialogue system now enables queries that return different values as instruments are updated.

In order for my fork to be merged with the main PyVISA-sim repository there are some styling complaints from Mypy in need of correction. This is not connected to the the implemented functionality. Another missing requirement is an update to the available documentation. This would need to include instructions on how to use the updated dialogues to connect instruments with functions. My research question regarding the open source software development was "Is it possible to make the requested functionalities suitable for the current framework" and reaching this far in the merging process proves that it it possible to combine Syntronics needs with the vision of the current PyVISA-sim maintainer.

There was one issue with meeting the needs of Syntronic while staying compliant with the PyVISA-sim framework. PASS is developed on Python version 3.6 and PyVISA-sim no longer supports versions older than 3.7. My final implementation is built using Python version 3.8. If Syntronic needs to stay on the older version they still have access to the earlier iterations on my branch, from before I upgraded. This does not limit the functionality. The only difference is that the earlier iterations use a new connection property instead of building further on the dialogue system.

6 Discussion

6.1 Results

The additions to the PyVISA-sim framework I have presented in this thesis have been developed for a very specific purpose requested by Syntronic. The connections currently have limitations which may make them unsuitable for other users than Syntronic.

One limitation is the way functions are called from the dialogues, which is not secure. I would not recommend using dynamic simulations from a YAML-file without knowing what code it contains. When simulating an instrument any code included in the function will be executed. Some laboratories may share files between multiple developers and researchers which could be a safety concern.

Another limitation is that it is currently not possible to connect multiple instruments in a chain. The dialogue query can only reply the current set value of an instruments property. This makes it impossible to use multiple DUTs. This could be avoided by using a more complex function, skipping all instruments in the chain and only connecting the original instruments.

These limitations have not hindered the possibility of getting the fork merged with the main PyVISA-sim repository.

The only identified issue from Syntronics point of view is the outdated Python version used with PASS. Upgrading older software is a big and not always worthwhile task. The presented solution, using an older implementation of my upgraded framework, gives them access to the requested simulation functionality at the cost of not having access to future updates of PyVISA-sim.

6.2 Method

There was a steep learning curve when getting used to working on the PyVISA-sim framework. There is no available code documentation, only on how to use the framework. A lot of the code is also imported from other frameworks hiding functionality. This turned out to not have an impact on my additions but the time spent trying to understand the parsing process could have been used elsewhere. If I was asked to make further additions, the development time would be significantly shortened.

I followed GitHub's recommended steps for working on open source development projects which describe the commonly occurring README-file where a "how to get started" text is often included [7]. If I was asked to work on a similar project in the future, the existence of such a text would heavily impact my interest in such a project.

When accepting this project there was a bigger emphasis on the simulation part and the literature study originally leaned into theory of measuring instruments and signal generation. Then, as the Syntronic team and I further discussed their problem and use cases we concluded that a lot of this was irrelevant when developing my solution. Therefore, this part was omitted and I chose a heavier focus on the challenges of open source development. If my code could be merged into the main PyVISA-sim repository that, by itself, would be a proof of quality.

The fast transition of focus was possible thanks to the agile development approach. The Syntronic team were always available for discussions where we could exchange ideas and problem solve together. There was no management involved in decisions so we could swiftly swap focus whenever needed. The Syntronic team knew what they required to solve their problems and I soon learned what would be possible within the bounds of the PyVISA-sim framework.

If the project had been developed for a more rigid company working this agile would not have been a valid approach [3]. Some companies rely heavier on following an original plan. It is important to set boundaries so an agile project can be executed without negatively impacting other parts of the company.

If I would have developed a tool outside of the PyVISA-sim framework the initial development process may had been faster and allowed for greater freedom. More tools can however complicate the development process so I can see why Syntronic wanted the requested functionality as part of PyVISA-sim. I also strongly believe in giving back to the community. By expanding the existing framework, my work can have an impact beyond Syntronic, creating more value, thereby justifying the time spent.

The execution of this project has relied heavily on information from web resources such as the different Python frameworks online documentation. These are first hand sources verified by the maintainers, but of different degrees of quality. A lot of the development decisions have also been based on discussions with the Syntronic team, relying on their needs rather than information from research literature.

6.3 The work in a wider context

Quality assurance

When developing open source software there must be a focus on stability. There may be dependencies created from other projects and regularly breaking existing functionality would harm the projects reputation and decrease future adaptation. At the time of writing this, 39 repositories and 14 packages are marked as dependent of PyVISA-sim on GitHub [21]. Trust is a cornerstone in any open source project and it is the reason why most projects will not let just anyone make changes. There are systems for approval set up to guarantee a certain quality and reliability, such as with proprietary development.

During development I have used the included testing frameworks to assure that no previous functionality breaks and that all new written code follows the required code standards. When pushing my code to the pull request set up on the PyVISA-sim GitHub repository, many tests are run automatically. The results of these are available to anyone, so nothing can be changed in secret.

Working from a distance

After the Coronavirus pandemic, beginning in 2019, many companies have been forced to adapt to having a large part of the workforce working away from the office. This comes with challenges like the one Syntronic is facing. Developers need to be able to execute their work without physical access to the workplace.

The tools created to solve these problems must be both secure and reliable as to not compromise the companies integrity and efficiency. PyVISA-sim was already a great tool for simulating measuring instruments and signals and I believe my additions for dynamic simulations make it even more so.

6.4 Future work

A further extension of the connections could be implementing them straight on instrument properties via the existing setters and getters. This would allow for chaining instruments by applying functions right when a property is called. This would make them behave more as real instruments, resulting in a more intuitive functionality.

Syntronic have also expressed an interest in having a graphical user interface for connecting instruments in simulations. This could allow for a faster simulation bench setup. I have not investigated in whether there is any interest in having this as part of the PyVISA-sim framework. It may be more suitable as part of the PASS software.

7 Conclusion

With this thesis I have further developed the Python framework PyVISA-sim for the company Syntronic to allow for dynamic simulations of signals and measuring instruments. I have also followed open source software development principles in order to make the additions usable for developers outside of Syntronic. The code is written following the code standards used in earlier iterations of the framework and I have created unit tests to ensure full reliability for the many developers and teams already utilizing PyVISA-sim.

The dynamic simulations are now functional thanks to an expanded dialogue system where two instruments can be connected via the already existing YAML system. When creating these connections there is also an option to apply a mathematical function to the signal received from the connected instrument, imitating a device under test. Any number of instruments and properties can be applied to one connection using a Python List.

Multiple iterations were developed with feedback from both the developers at Syntronic and the maintainer of the original PyVISA-sim repository. The development discussions took place in a pull request on the official GitHub page where the community could make suggestions and give public feedback. The final iteration moved over to Python version 3.8 due to PyVISA-sim no longer supporting versions under 3.7. This breaks functionality with Syntronics tool PASS which is developed on version 3.6. The upgrade was a hard requirement in order to get the pull request approved for the main repository. If Syntronic decides to stay on the older version they still have access to my older iterations developed on version 3.6.

The upgraded framework resulting from this thesis opens up PyVISA-sim for new, more complex, use cases. Dynamic simulations streamlines testing of software without access to real physical instruments. In the future this can also be expanded to allow for chaining of simulated instruments, greatly reducing written code. This is now not only available for the team at Syntronic developing PASS, but for any developers working without reliable physical access to their intended instruments.

I believe it was a great choice from Syntronic to allow the development to be done using open source methods, thereby letting the world have access to the results. If all companies developed software this way we would all greatly benefit. These additions can have an especially big impact in the post pandemic society of today where working from home, for many developers, now is the norm and companies therefore are further restricting access to physical work places [24].

Bibliography

[1] Ijaz Ahmad, Shahriar Shahabuddin, Tanesh Kumar, Jude Okwuibe, Andrei Gurtov, and Mika Ylianttila. “Security for 5G and beyond”. In: *IEEE Communications Surveys & Tutorials* 21.4 (2019), pp. 3682–3722.

[2] Cathal Boogerd and Leon Moonen. “Assessing the value of coding standards: An empirical study”. In: *2008 IEEE International Conference on Software Maintenance*. 2008, pp. 277–286. DOI: `10.1109/ICSM.2008.4658076`.

[3] A. Cockburn and J. Highsmith. “Agile software development, the people factor”. In: *Computer* 34.11 (2001), pp. 131–133. DOI: `10.1109/2.963450`.

[4] Jovica Đurković, Vuk Vuković, and Lazar Raković. “Open source approach in software development—Advantages and disadvantages”. In: *Manag Inforation Syst* 3 (2008), pp. 029–33.

[5] *Flake8*. 2022. URL: `https://github.com/pycqa/flake8` (visited on 12/20/2022).

[6] *Flake8: Your Tool For Style Guide Enforcement*. 2022. URL: `https://flake8.pycqa.org/en/latest/` (visited on 12/20/2022).

[7] *How to Contribute to Open Source*. 2022. URL: `http://opensource.guide/how-to-contribute` (visited on 12/15/2022).

[8] Haley Hunter-Zinck, Alexandre Fioravante De Siqueira, Váleri N Vásquez, Richard Barnes, and Ciera C Martinez. *Ten simple rules on writing clean and reliable open-source scientific software*. 2021.

[9] N. Juristo, A.M. Moreno, and W. Strigel. “Guest Editors’ Introduction: Software Testing Practices in Industry”. In: *IEEE Software* 23.4 (2006), pp. 19–21. DOI: `10.1109/MS.2006.104`.

[10] *McCabe complexity checker*. 2022. URL: `https://github.com/PyCQA/mccabe` (visited on 12/20/2022).

[11] Mike McCormick. “Waterfall vs. Agile methodology”. In: *MPCS, N/A* 3 (2012).

[12] Muriuki Mureithi. “The Internet journey for Kenya: The interplay of disruptive innovation and entrepreneurship in fueling rapid growth”. In: *Digital Kenya* (2017), pp. 27–53.

[13] *mypy*. 2022. URL: `http://mypy-lang.org/` (visited on 12/20/2022).

[14] Joni Ollanketo. “RF performance testing using Robot Framework”. In: (2016).

[15] *PEP 484 – Type Hints*. 2022. URL: https://peps.python.org/pep-0484/ (visited on 12/20/2022).

[16] Pawani Porambage, Gürkan Gür, Diana Pamela Moya Osorio, Madhusanka Liyanage, Andrei Gurtov, and Mika Ylianttila. "The roadmap to 6G security and privacy". In: *IEEE Open Journal of the Communications Society* 2 (2021), pp. 1094–1122.

[17] *pycodestyle (formerly called pep8) - Python style guide checker*. 2022. URL: https://github.com/PyCQA/pycodestyle (visited on 12/20/2022).

[18] *Pyflakes*. 2022. URL: https://github.com/PyCQA/pyflakes (visited on 12/20/2022).

[19] *pytest: helps you write better programs*. 2022. URL: https://docs.pytest.org/en/7.2.x/ (visited on 12/15/2022).

[20] *PyVISA-sim*. 2022. URL: https://github.com/pyvisa/pyvisa-sim (visited on 12/19/2022).

[21] *PyVISA-sim dependency graph*. 2022. URL: https://github.com/pyvisa/pyvisa-sim/network/dependents (visited on 12/29/2022).

[22] *PyVISA-sim: Simulator backend for PyVISA*. 2022. URL: https://pyvisa.readthedocs.io/projects/pyvisa-sim/en/latest/index.html (visited on 12/15/2022).

[23] *PyVISA: Control your instruments with Python*. 2022. URL: https://pyvisa.readthedocs.io/en/latest/ (visited on 12/15/2022).

[24] Daniel Russo, Paul HP Hanel, and Niels van Berkel. "Understanding Developers Well-Being and Productivity: A Longitudinal Analysis of the COVID-19 Pandemic". In: *arXiv preprint arXiv:2111.10349* (2021).

[25] *The Big Ol' List of Rules*. 2022. URL: https://www.flake8rules.com/ (visited on 12/20/2022).

[26] *The Black code style*. 2022. URL: https://black.readthedocs.io/en/stable/the_black_code_style/current_style.html (visited on 12/20/2022).

[27] *The Uncompromising Code Formatter*. 2022. URL: https://pypi.org/project/black/ (visited on 12/20/2022).

[28] *The VISA Library Specification*. 2022. URL: https://www.ivifoundation.org/downloads/Architecture%20Specifications/vpp43_2022-05-19.pdf (visited on 12/15/2022).

[29] *Tuleap Pull Request, open source code review tool*. 2022. URL: https://blog.tuleap.org/tuleap-pull-request-open-source-code-review-tool (visited on 12/29/2022).

[30] Guido Van Rossum, Barry Warsaw, and Nick Coghlan. "PEP 8–style guide for python code". In: *Python. org* 1565 (2001).

[31] *YAML Ain't Markup Language*. 2022. URL: https://yaml.org (visited on 12/16/2022).