UNIVERSIDAD SAN FRANCISCO DE QUITO

Colegio de Ciencias e Ingeniería

Desarrollo de una interfaz gráfica remota para un caracterizador de semiconductores y un analizador de redes

Juan Pablo Santos Santacruz

Mauricio Iturralde, Ph.D., Director de Tesis

Tesis de grado presentada como requisito para la obtención del título de Ingeniero en Sistemas

Quito, mayo de 2014

Universidad San Francisco de Quito
Colegio de Ciencias e Ingeniería

HOJA DE APROBACIÓN DE TESIS

Desarrollo de una interfaz gráfica remota para un caracterizador de semiconductores y un analizador de redes

Juan Pablo Santos Santacruz

Mauricio Iturralde, Ph.D.
Director de Tesis .....

Fausto Pasmay, M.Sc.
Miembro del Comité de Tesis .....

Mauricio Iturralde, Ph.D.
Miembro del Comité de Tesis .....

Lionel Trojman, Ph.D.
Miembro del Comité de Tesis .....

Ximena Córdova, Ph.D.
Decana de la Escuela de Ingeniería .....
Colegio de Ciencias e Ingeniería

Quito, mayo de 2014

Firma:

Nombre: Juan Pablo Santos Santacruz

C. I.: 1717823346

Fecha: Quito, mayo de 2014

RESUMEN

Este proyecto de tesis presenta el desarrollo de una aplicación para el control y automatización de un Sistema Caracterizador de Semiconductores Keithley K4200-SCS y de un Analizador de Redes Agilent E5071c. El software se desarrolló en cuatros partes diferentes. Todas implementadas utilizando el lenguaje de programación Python.

La primera parte es una librería de comunicación general que se encarga de enviar comandos a los instrumentos y de recuperar los datos generados por una medición. Todo a través del protocolo TCP/IP.

La segunda parte es una librería orientada a objetos que valida y genera comandos para realizar mediciones en el Keithley K4200-SCS. Esta librería expone la funcionalidad básica del caracterizador de semiconductores.

La tercera parte es una librería orientada a objetos que valida y genera comandos para el Agilent E5071c.

La cuarta parte consta de una interfaz gráfica que hace uso de estas librerías para presentar una interfaz unificada para el Agilent E5071c y el Keithley K4200-SCS. La librería utilizada para la interfaz gráfica fue PyQt.

ABSTRACT

This thesis projects presents the development of an application for the control and automatization of a Keithley K4200 Semiconductor Characterization System and and Agilent E5071c Vector Network Analyzer. The application was developed in four different parts. Everything was implemented using the Python programming language

The first part is a library that implements general transmission of commands to the instruments, as well as recovery of the data produced by a measurements. Everything is done over TCP/IP.

The second part involves an object oriented library used to validate and generate commands to make measurements with the Keithley K4200-SCS. This library exposes the basic functionality of the device.

The third part involves an object oriented library used to validate and generate commands to make measurements with the Agilent E5071c.

The fourth part is a graphical user interface that uses the previously developed libraries to present a unified interface for the Keithely K4200-SCS and the Agilent E5071c. The library used for this development was PyQt.

Índice

Índice de Figuras

Índice de Tablas

Lista de acrónimos

DUT: Device Under Test
GPIB: General Purpose Interface Bus
IEEE: Institute of Electric and Electronic Engineers
KITE: Keithley Interactive Test Enviromment
KXCI: Keithley External Control Interface
OSI: Open Systems Interconnection
RAII: Resource Allocation is Initialization
RF: Radiofrecuencia
SCPI: Standard Commands for Programmable Instruments
SCS: Semiconductor Characterization System
SMU: Source Measure Unit
SOLT: Short, Open, Load, Through
TCP/IP: Transmission Control Protocol/Internet Protocol
UTP: Unshielded Twisted Pair
USFQ: Universidad San Francisco de Quito
VNA: Vector Network Analyzer
YAML: YAML Ain't Markup Language

1. Introducción

1.1 Antecedentes

Los equipos de investigación científica cuentan con sus propias herramientas para realizar mediciones. Sin embargo, es común que estas herramientas de software tengan limitaciones que impidan la utilización del equipo a su máximo potencial. Por esta razón la gran mayoría de dispositivos proveen interfaces de programación que permiten adecuar el dispositivo a cualquier propósito. En general, estas interfaces tienden a ser de bajo nivel e implican la formación de series de comandos para poner al aparato en el estado deseado para una medición.

Independientemente del fabricante de cada dispositivo, el protocolo de comunicación y la sintaxis de los comandos utilizados suelen seguir estándares comunes. Sin embargo, tal estandarización no establece nombres para cada uno de los comandos, por lo que éstos son definidos por cada uno de los fabricantes. Es la responsabilidad del fabricante proveer tanto las implementaciones del protocolo de comunicación en forma de librerías para lenguajes de programación específicos, como la documentación independiente del lenguaje de programación que se utilice. Gracias a esta estandarización es posible utilizar las mismas librerías de comunicación para cualquiera de los aparatos. Así, el programa escrito es responsable de enviar los comandos correctos a través de la interfaz provista por el fabricante del aparato.

El desarrollo de un software hecho a medida, permite integrar el funcionamiento de varios aparatos a través de una sola interfaz. Por ejemplo, es posible realizar acciones en un analizador de redes, en función de condiciones presentes en un caracterizador de semiconductores. Este tipo de integración es la principal motivación detrás de la implementación de software hecho a medida para manejo de aparatos de investigación. Tal integración es capaz de coordinar mediciones utilizando dos aparatos diferentes al mismo tiempo.

Para la presente tesis se desarrolló una interfaz unificada para un analizador vectorial de redes y para un caracterizador de semiconductores. Este desarrollo permite realizar mediciones en cada uno de estos equipos a través de la misma interfaz gráfica.

1.2 Equipos involucrados en esta tesis

1.2.1 Agilent E5071C, Analizador Vectorial de Redes

Este es un equipo que permite medir la matriz de dispersión de un circuito, antena o cualquier otro dispositivo bajo pruebas o Device Under Test (DUT). Su nombre en inglés es Vector Network Analyzer por lo que usualmente se refiere como VNA. La matriz de dispersión multiplicada por el vector de potencia incidente produce el vector de potencia reflejada. El modelo específico disponible en la USFQ es la versión de dos puertos (también existe una versión de cuatro puertos). Este dispositivo es capaz de generar corrientes alternas desde los 9 kHz hasta los 9Ghz. Al mismo tiempo que proporciona esta corriente, el dispositivo es capaz de medir la dispersión de potencia en el mismo puerto y así obtener los parámetros de la matriz de dispersión del DUT. El aparato puede

operar en cuatro modos básicos:

1. Medición del parámetro S11: parámetro que define la dispersión en el primer puerto. También puede ser visto como el coeficiente de reflexión en el puerto de entrada.
2. Medición del parámetro S12: parámetro que define la ganancia de tensión en reversa
3. Medición del parámetro S21: parámetro que define la ganancia de tensión en directa.
4. Medición del parámetro S22: parámetro que define la dispersión en el segundo puerto. También puede ser visto como el coeficiente de reflexión en el puerto de salida.

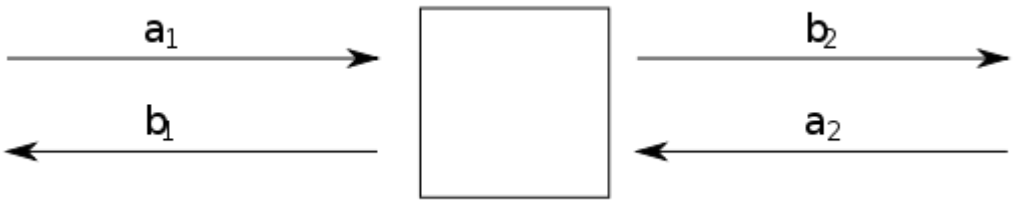


Figura 1: Red de dos puertos. La caja central representa el dispositivo bajo pruebas o DUT

Cada uno de estos parámetros es miembro de la matriz de parámetros S, que caracteriza la red de dos puertos. La relación entre a_1, a_2, b_1 y b_2 y los parámetros S se puede observar en la Figura 2. A partir de esta matriz y utilizando la impedancia característica de cada uno de los puertos del VNA es posible utilizar relaciones basadas en álgebra matricial para derivar la matriz de parámetros-Y o matriz de admitancia o la matriz de parámetros-Z o matriz de impedancia de un DUT (Caspers, 2007). La transformación entre estas matrices se puede hacer

directamente en el aparato o en software en una computadora personal. La conversión entre la matriz de admitancia y la matriz de impedancia es todavía más sencilla ya que una es la inversa de la otra. (Caspers, 2007). Estas relaciones se pueden ver en el Anexo 3.

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

Figura 2: Relación establecida por los parámetros S en una red de dos puertos

La comunicación con el equipo se puede hacer a través de Ethernet. El firmware del dispositivo provee un servidor que utiliza el protocolo Transmission Protocol/Internet Protocol (TCP/IP) en el puerto 5025 que escucha comandos, los valida y los ejecuta. El servidor se encuentra en ejecución todo el tiempo, por lo que es posible manipular el estado del dispositivo tanto desde la pantalla táctil y teclas integradas como desde un cliente conectado al puerto 5025 del VNA. La sintaxis de los comandos sigue el estándar Standard Commands for Programmable Instruments (SCPI), que es parte del estándar IEEE-488 o GPIB. Aunque la capa física de la conexión ya no sea basada en el protocolo IEEE-488, la sintaxis de los comandos si está basada en este estándar.

Debido al hecho que el VNA funciona con corriente alterna es necesario calcular coeficientes de corrección para las medidas que se hacen con este aparato. Por esto, en la presente tesis se desarrolló un módulo de calibración. El módulo calibra el VNA con los tres tipos básicos de calibración: LOAD, OPEN y

THROUGH. Cada uno de estos modos debe ser calibrado utilizando tres modos LOAD, OPEN y SHORT. Una vez que la calibración de los tres tipos para cada uno de los tres modos ha terminado, el módulo instruye al aparato a guardar los coeficientes calculados durante el proceso de calibración. Esto garantiza que las medidas realizadas en el futuro sean correctas.

La calibración se hace utilizando kits provistos por el fabricante y tienen características conocidas. El módulo de calibración desarrollado para la presente tesis solo soporta el kit de calibración Agilent 85033E.

1.2.2 Keithley K4200-SCS, sistema caracterizador de semiconductores

El K4200 es un caracterizador de semiconductores que permite aplicar voltajes y corrientes muy precisas entre y a través de polos en un circuito. Al mismo tiempo que aplica un voltaje, el aparato es capaz de medir la corriente inducida por tal voltaje. Esto posibilita obtener curvas de de voltaje-corriente, que permiten obtener características de diferentes semiconductores (What is a Source-Measure Unit, 2010). De igual forma puede aplicar una corriente a través de un circuito y al mismo tiempo medir el voltaje que resulta a partir de tal corriente. El Keithley 4200-SCS es capaz de realizar estas mediciones aplicando voltajes y corrientes muy pequeñas, en el orden de los nanoamperios (What is a Source-Measure Unit, 2010).

El Keithley 4200-SCS utiliza unidades de medición-fuente (Source Measure Unit o SMU por sus siglas en inglés) para medir y proveer potencia al mismo tiempo al circuito. La medición se hace a través del método de Kelvin de los cuatro cables para tomar en cuenta la resistencia de los cables de conexión entre

el caracterizador de semiconductores y el DUT (What is a Source-Measure ...). El aparato puede aceptar hasta ocho SMUs para caracterizar ocho circuitos, al mismo tiempo. El equipo instalado en el laboratorio de Radiofrecuencia y Microelectrónica de la USFQ tiene cuatro tarjetas SMU instaladas. Además de esto es posible instalar fuentes exclusivas de voltaje (llamados VS o Voltage Source) en el aparato.

La configuración por comandos del equipo se hace a través de canales asignados a cada uno de los SMUs. Cada canal corresponde a un SMU y puede ser configurado como fuente de corriente o de voltaje. A cada canal se le asigna una de las cuatro funciones de *sourcing* disponibles en el equipo:

1. VAR1 o Sweep
2. VAR2 o Step
3. Constante
4. VAR1': VAR1 escalado por un factor específico

(Model 4200-SCS Semiconductor Characterization System Reference Manual, 2011)

Cada una de las funciones de *sourcing* tiene sus propios parámetros de configuración. La función VAR1 permite configurar barridos de voltaje especificando el inicio, fin e intervalo entre los cuales se desea realizar el barrido. La función VAR2 permite realizar mediciones en pasos, especificando el inicio, el intervalo (o paso), y el número de pasos. La función constante como su nombre lo indica, permite asignar un voltaje o corriente constante a ser aplicado por el SMU. VAR1' es un barrido como VAR1, pero cada paso se encuentra escalado por una

constante. Además de los parámetros antes especificados todas las funciones de *sourcing* toman el parámetro *compliance* para indicar hasta qué valor de voltaje se debe intentar para llegar a la corriente especificada o hasta qué valor de corriente se debe intentar para llegar al voltaje especificado (Model 4200-SCS Semiconductor Characterization System Reference Manual, 2011). Todos estos valores tienen rangos específicos que deben ser respetados por la aplicación que configura el caracterizador de semiconductores.

Las conexiones entre el caracterizador y los puntos específicos en el DUT se realizan de forma manual, con la ayuda de un amplificador óptico (estilo microscopio); y una serie de perillas que mueven los terminales y los ponen en contacto con los puntos específicos del DUT. Debido a que este proceso es manual no se abordará el tema en esta tesis.

De la misma manera que el VNA, el caracterizador tienen la capacidad de aceptar comandos a través de una conexión Ethernet y provee un servidor TCP/IP que escucha en el puerto 1025 y acepta comandos para manipular el estado del caracterizador de semiconductores. A diferencia del Agilent E5071C, el caracterizador requiere que un software específico llamado Keithley External Control Interface (KXCI), se encuentre abierto en el dispositivo y tenga acceso exclusivo al hardware dentro del dispositivo. Esto impide que se realicen modificaciones al estado del Keithley 4200-SCS mientras se está programando a través de la conexión remota TCP/IP.

A diferencia del VNA, el caracterizador de semiconductores define los comandos en una sintaxis específica de este fabricante. Esta sintaxis se encuentra ampliamente documentada en los manuales de usuario del aparato. Sin

embargo, los estándares en el resto de capas del modelo OSI siguen los mismos estándares de cualquier pila TCP/IP.

1.3 Justificación

Este proyecto pretende implementar interfaces de control del VNA y del caracterizador de semiconductores en la misma interfaz gráfica. Además se prevé que se pueda usar los aparatos desde lugares remotos dentro del mismo campus de la USFQ. Utilizar los programas desde una computadora personal y no directamente desde la interfaz de los mismos, también previene el uso incorrecto de los aparatos. El riesgo de uso incorrecto es alto debido al hecho de que ambos aparatos funcionan con Windows XP y por tanto pueden ejecutar cualquier programa para Windows XP. La interfaz permite solo realizar mediciones, por lo que se evita la posible instalación de programas innecesarios en los aparatos. De igual forma se puede evitar la necesidad de acceso físico a los aparatos para realizar mediciones. Lo único necesario para manejar el equipo remotamente y para obtener los datos producidos, es encontrarse en la misma subred en la que está el aparato. Gracias al hecho de que ambos aparatos utilizan Ethernet para la comunicación es posible conectarse a ambos aparatos y enviar comandos a cada uno de ellos, de manera simultánea. Las conexiones entre la computadora personal y los aparatos se realiza a través de cables unshielded twisted pair (UTP) comunes y de bajo costo. Esto es algo que no era posible antes del desarrollo del software de la presente tesis.

Debido al hecho de que la interfaz para el caracterizador y para el analizador vectorial de redes es una sola es posible realizar una medida sobre un dispositivo semiconductor, utilizando ambos aparatos al mismo tiempo. Se pretende realizar una medición utilizando el VNA como fuente de corriente alterna de alta frecuencia y el caracterizador como fuente de corriente directa y como

amperímetro y voltímetro. De esta manera se pueden realizar mediciones aprovechando todos los aparatos del Laboratorio de Microelectrónica de la USFQ.

El programa también tiene la capacidad tanto de guardar la configuración utilizada para el analizador vectorial de redes, como la configuración utilizada para el caracterizador de semiconductores. Esto permite guardar configuraciones en disco y cargarlas según sea necesario, lo que hace posible restaurar el aparato a una configuración específica cuando se requiera. Esta funcionalidad está soportada en Keithley Interactive Test Enviromment (KITE), pero no esta soportada de ninguna manera en el analizador de redes. Debido a que la interfaz es única para ambos aparatos, el archivo de configuración define los parámetros tanto para el Keithley K4200-SCS como para el Agilent E5071C.

La realización de esta tesis permite medir capacitancias a diferentes frecuencias y en diferentes polarizaciones en un mismo semiconductor. Es importante decir que al momento de la escritura de este documento, el Laboratorio de Microelectrónica de la USFQ es el único que cuenta con un caracterizador de semiconductores y con un analizador de redes.

1.4 Objetivos

1.4.1 Objetivo general

Permitir el control remoto simultáneo del Analizador Vectorial de Redes y del Caracterizador de Semiconductores a través de una red local en el Laboratorio de Radiofrecuencia y Microelectrónica de la USFQ, utilizando una interfaz unificada para ambos aparatos.

1.4.2 Objetivos específicos

1. Desarrollar una librería orientada a objetos en Python que permita el control programático del Sistema Caracterizador de Semiconductores Keithley K4200-SCS desde un programa escrito en Python o C.
2. Desarrollar una librería orientada a objetos en Python que permita el control programático del Analizador Vectorial de Redes Agilent E5071C desde un programa escrito en Python o C.
3. Desarrollar una interfaz gráfica unificada que permita utilizar el VNA y el caracterizador de semiconductores de forma simultánea.
4. Desarrollar una interfaz gráfica para la calibración del VNA.

1.4.3 Metas

1. Diseñar una librería orientada a objetos para el control del caracterizador de semiconductores.
2. Diseñar una librería orientada a objetos para el control del analizador vectorial de redes.
3. Escribir pruebas unitarias con los resultados deseados en la librería utilizada para controlar el caracterizador de semiconductores.
4. Implementar las clases de la librería para el caracterizador de semiconductores.
5. Implementar las clases de la librería para el analizador vectorial de redes.
6. Utilizar las librerías antes desarrolladas para controlar el estado del aparato a través de un programa escrito en Python.

7. Diseñar una interfaz gráfica que contenga los elementos necesarios para el control simultáneo del VNA y del caracterizador de semiconductores.
8. Diseñar una interfaz gráfica para la calibración del VNA
9. Utilizar las librerías antes desarrolladas desde la interfaz gráfica para controlar el analizador de redes y el caracterizador de semiconductores.
10. Utilizar una interfaz gráfica para llamar a métodos de la librería a través de una interfaz amigable.

1.4.4 Actividades

1. Implementar la librería de comunicación entre los aparatos y la computadora utilizando sockets de bajo nivel.
2. Escribir las pruebas unitarias que verifiquen que los comandos producidos por la librería del caracterizador de semiconductores sean correctos.
3. Implementar el generador de comandos para configuración de canales y de funciones de *sourcing*.
4. Implementar el generador de comandos para configuración de la función de barrido (*sweep*).
5. Implementar el generador de comandos para configuración de la función de barridos por pasos (*step sweep*)
6. Implementar el generador de comandos para configuración de la función de valores constantes.
7. Implementar el validador de datos para el caracterizador de semiconductores.

8. Integrar el validador de datos en los generadores de comandos
9. Implementar las funciones de extracción de datos del caracterizador de semiconductores
10. Implementar las funciones de guardado de datos en el disco duro de archivos CSV (comma separated values o valores separados por comas) basado en las preferencias del usuario (i.e nombre de archivo) para el caracterizador de semiconductores
11. Implementar la función de barrido basado en inicio y fin, en el analizador vectorial de redes.
12. Implementar el método de barrido basado en centro y amplitud, en el analizador vectorial de redes.
13. Implementar el método de configuración de marcadores y movimiento de los mismos para el analizador vectorial de redes.
14. Implementar el método de selección de formato de datos para el analizador vectorial de redes.
15. Implementar el método de selección del parámetro S a ser medido por el analizador vectorial de redes.
16. Implementar el método de preferencias en la selección de auto-escalamiento de los datos.
17. Implementar los métodos necesarios para la extracción de datos del analizador vectorial de redes.
18. Implementar los métodos que verifican si el aparato ha acabado de ejecutar

un comando

19. Implementar los métodos necesarios para la calibración del aparato.
20. Implementar los métodos necesario para la escritura de archivos CSV en el disco duro de la computadora que controla en analizador vectorial de redes.
21. Diseñar una interfaz gráfica en QtDesigner única para el analizador de redes y para el caracterizador de semiconductores.
22. Diseñar una interfaz gráfica para el diálogo de calibración del analizador de redes
23. Generar código Python a partir de los archivos producidos por QtDesigner.
24. Conectar los diferentes eventos generados por la interfaz gráfica a métodos específicos (o *slots*) que realicen las mediciones necesarias.
25. Escribir los métodos necesarios (*slots*) para que las interacciones con la interfaz gráfica produzcan los comandos correctos en los equipos.
26. Escribir los métodos necesarios de serialización y deserialización del estado de la interfaz gráfica a un archivo de tipo YAML (YAML Ain't a Markup Language).
27. Escribir los métodos que comprimen el archivo YAML con gzip y lo guardan con extensión .config.
28. Empaquetar la aplicación junto con las librerías dependientes y el intérprete de Python utilizando *Py2EXE* en un solo ejecutable que no tiene dependencias externas.

29. Transferir el ejecutable al computador donde se va a utilizar

2. Marco Teórico

2.1 Lenguajes de programación

2.1.1 Python

El lenguaje de programación Python es un lenguaje interpretado con notificación fuerte, y dinámica. Tradicionalmente una vez que un programa se ha escrito, otro programa conocido como compilador, transforma el código fuente escrito por el programador a lenguaje de máquina, que luego es ejecutado por el CPU de un computador. El lenguaje interpretado significa que un programa llamado intérprete ejecuta el código.

En el caso de Python, el intérprete primero compila el código Python a una serie instrucciones independientes del CPU llamadas *bytecode* (Python Glossary: Bytecode). Luego estas instrucciones son ejecutadas por la máquina virtual de Python. El hecho de que el código es interpretado permite ejecutar el código de forma directa sin necesidad de transformarlo primero a instrucciones de una arquitectura específica de procesadores. Esto hace también que el código Python pueda ser ejecutado en cualquier sistema operativo que tenga un intérprete de Python disponible. El hecho de que la notificación sea fuerte significa que una vez que una variable tiene un tipo, no es posible adaptarla o modificarla para que tenga un tipo diferente, a menos de que se lo haga de manera explícita. Tipificado dinámico significa que Python deduce los tipos de los variables durante el tiempo de ejecución y no es necesario especificarlos dentro del código fuente del programa.

La elección de Python como lenguaje para la implementación del programa hace posible que el programa funcione en cualquier sistema operativo que provea un intérprete de Python. Sin embargo, para que esto sea posible también es necesario que todas las librerías (especialmente aquellas escritas en C), estén disponibles para todos los sistemas operativos antes listados. Todas las librerías usadas en la presente tesis son escritas en Python o se encuentran integradas en la librería estándar de Python, por lo que esto no es un impedimento a la portabilidad entre sistemas operativos de la aplicación.

2.1.2 C++

C++ es un lenguaje de programación compilado y orientado a objetos basado en C. Originalmente llamado C with Classes, C++ tuvo la intención de traer la programación orientada a objeto al lenguaje C (Stroustrup, 1994). C++ es ahora bastante más que eso y tiene una extensa librería estándar que provee al menos una docena de estructuras de datos, soporte para metaprogramación, una librería de hilos de ejecución, inferencia de tipos y la posibilidad de utilizar un recolector de basura en lugar de manejo manual de memoria como es normal en C (Stroustrup, 1994). Toda la librería utilizada para la interfaz gráfica en la presente tesis se encuentra implementada en C++. Sin embargo, se utilizan módulos nativos de Python para interactuar con ella sin necesidad de escribir C++. Ninguna parte de esta tesis fue escrita en C++.

2.2 Estándares de comunicación

2.2.1 SCPI

La sintaxis de comandos utilizados por muchos aparatos científicos se basa

en el estándar SCPI promulgado por la IEEE como parte opcional del estándar IEEE-488 o estándar GPIB. SCPI permite estructurar los comandos como un árbol que ordena los comandos en una jerarquía específica (Hafok, 2006).. Además de esto la invocación de comandos involucra recorrer el árbol de comandos en una ruta específica y ejecutar comandos que corresponden a las hojas del árbol (Hafok, 2006). Así, SCPI define símbolos que significan bajar un nivel en el árbol y símbolos que significan regresar a la raíz del árbol. El analizador de redes utiliza este estándar para definir la sintaxis de sus comandos (User Manual for E5071C, 2013).

2.2.2 Ethernet

Toda la comunicación con los aparatos se hace sobre Ethernet y sobre el protocolo TCP/IP. Ethernet es probablemente el estándar de comunicación más utilizado en el mundo (Pedreiras et al, 2002). Permite armar redes locales y redes anchas de bajo costo utilizando cables de par de cobre. Permite velocidades desde 1Mbps hasta 10Gbps. Este es el protocolo de capa física utilizado para comunicarse con los aparatos. En la capa de red el protocolo es IP y en la capa de aplicación el protocolo es SCPI, en el caso del analizador de redes (User for Manual E5071C, 2013). En el caso del sistema caracterizador de semiconductores la capa de aplicación es propietaria de Keithley y definida por ellos.

2.3 Librerías utilizadas

2.3.1 PyQT

PyQT es una interface para Python para acceder a la librería Qt. Qt a su

vez es una librería para C++ que provee una interface gráfica, facilidades de acceso a red, hilos de ejecución, estructuras de datos, una librería de señales, multimedia y gráficos. En la presente tesis se limitó el uso de Qt para la implementación de la interfaz gráfica del programa. La librería es orientada a objetos y permite conectar métodos de una clase a eventos generados en la interfaz gráfica del programa (Hilsheimer). En general, Qt permite que objetos generen señales que luego serán conectados a *slots* o métodos en otro objeto que reaccionarán (serán llamados) frente a la emisión de señales. Esta es una implementación del patrón de diseño del Observador (Hilsheimer). De acuerdo con Digia (empresa que desarrolla Qt), el paradigma señal-slot valida tipos de variables en tiempo de compilación, en contraste con el paradigma de *callbacks* utilizado en librerías como GTK o en Windows. La desventaja es que C++ no soporta el paradigma señal-slot por su cuenta, por lo que los programas escritos con Qt deben ser preprocesados por la herramienta *moc* para convertirlos en código C++ válido (Why Does Qt Use Moc...). *moc* es una herramienta de línea de comandos que toma código de forma señal-slot de Qt y lo convierte a código C++ válido.

El dibujado de los diferentes *widgets* en la pantalla se realiza por el motor de renderizado nativo de cada una de las plataformas donde Qt esta soportado, por lo que las aplicaciones lucen nativas en cualquier sistema operativo.

La mayor ventaja de usar Qt es su portabilidad. Qt cuenta con versiones nativas para Mac OS X, Linux y Windows (Senyk). La librería se encarga de abstraer las diferencias entre los sistemas operativos en lo que concierne al dibujado de widgets e interacción con interfaces nativas de cada sistema

operativo (por ejemplo, el dialogo de selección de archivos). Esto permite escribir el código una vez y correrlo en cualquier sistema operativo. (Senyk)

2.3.2 Sockets

Para la comunicación en red con los aparatos se utilizó la librería *socket*, que es parte de la librería estándar de Python. Sobre esta librería se implementó una capa de aplicación que permite convertir el *stream* de datos de TCP a un protocolo orientado a mensajes.

2.3.3 Nose y unittest

Nose es una librería para Python que busca y ejecuta las pruebas unitarias de una aplicación de forma automática. Unittest es una librería integrada con Python que permite escribir, evaluar pruebas unitarias y mostrar el resultado de su ejecución. Unittest permite escribir clases con métodos con declaraciones que deben cumplirse para que la prueba unitaria se considere completada satisfactoriamente (unittest, Unit Testing Framework).

2.3.4 YAML y PyYAML

YAML Ain't a Markup Language (YAML) es un formato de serialización de datos amigable, con librerías para muchos lenguaje de programación. El objetivo es crear un formato de texto más fácil de leer que XML. Está diseñado de tal forma que sus construcciones sean fáciles de relacionar con estructuras de datos presentes en lenguajes de programación de alto nivel (Ben-Kiki, Evans). Así YAML soporta listas, diccionarios, escalares, número enteros y en punto flotante.

PyYAML es una librería que expone los métodos de libyaml (escrito en C),

a través de una interfaz que puede ser utilizada desde Python. Esta librería se utiliza para serializar la configuración en memoria del programa al disco duro y para leer esta configuración a una estructura de datos en memoria que se usa para restaurar el programa al estado guardado en el archivo.

2.3.5 Gzip

Gzip es una librería de compresión de datos de propósito general desarrollada en 1992 por el proyecto GNU. Utiliza el algoritmo DEFLATE para compresión (GZIP file format specification, 1996). A su vez el algoritmo DEFLATE es una combinación del algoritmo LZ77 y de codificación a través de árboles de Huffmann (GZIP file format specification, 1996). Los archivos de configuración del programa son guardados en formato YAML y comprimidos utilizando gzip. La librería de gzip utilizada en el desarrollo de la aplicación es la provista como parte de la librería estándar de Python.

2.4 Calibración SOLT (Short, Open, Load, Thru)

La calibración de un analizador vectorial de redes pretende eliminar los errores sistemáticos presentes en la medición de parámetros de dispersión (parámetros S). Para una calibración SOLT son necesarios cuatro estándares: un estándar short, un estándar open, un estándar load y un estándar thru. Los estándares son parte del sustrato de calibración utilizado para calibrar el VNA. A partir de esta serie de estándares con valores de capacitancia e inductancia conocidos y de mediciones realizadas sobre los estándares, es posible resolver un sistema lineal de ecuaciones para obtener una serie de coeficientes de error que luego pueden ser usados para eliminar errores sistemáticos en la mediciones.

Los valores de inductancia y capacitancia de los estándares en el sustrato de calibración son provistos por el fabricante. (Understanding VNA Calibration, 16)

El sistema de ecuaciones lineales en el modelo de corrección de errores de calibración SOLT tiene un sistema de doce ecuaciones diferentes. Este sistema de ecuación es común a toda calibración SOLT. Sin embargo, el modelo de corrección de errores específico depende del fabricante del analizador de redes. Para obtener los coeficientes de error para una calibración de dos puertos se mide un short, un open y un load en cada uno de los puertos además de un thru entre los dos puertos. (Understanding VNA Calibration, 17)

2.5 Metodología

El desarrollo de esta tesis se realizó en respuesta al requerimiento de dos profesores a tiempo completo del Departamento de Ingeniería Electrónica de la USFQ. Como en cualquier proyecto de software el primer paso en el desarrollo fue reunirse con aquellos que tienen la necesidad. En esta reunión se debe recopilar la mayor cantidad de información sobre los requerimientos y expectativas que tienen los usuarios finales del programa. Con esta información es posible dividir el desarrollo en diferentes partes que luego serán sometidas a un proceso de desarrollo iterativo e incremental (Larman y Basili, 2003). El siguiente paso consiste en investigar como se va a implementar determinada funcionalidad. Esto se realiza utilizando los manuales de utilización provistos por el fabricante de los aparatos.

Se desarrolla una parte de la aplicación y se la revisa con el usuario. Se registran las observaciones hechas por el usuario y se implementan para la siguiente reunión con el usuario. Al principio de esta reunión se verifica que cada

uno de los requerimientos de la reunión anterior se encuentren implementados de forma satisfactoria y se procede a revisar el resto de funcionalidad. Se vuelven a anotar las observaciones del usuario y se vuelven a implementar los cambios. Es posible que estos cambios incluyan cambios de diseño, ya que separar diseño e implementación no es posible (Larman y Basili, 2003) . Una vez que el usuario está satisfecho con la funcionalidad de una parte de la aplicación, se procede a la siguiente parte de la aplicación y se vuelve a realizar el modelo iterativo. Una vez que todas las partes hayan sido sometidas al proceso iterativo, la aplicación se considera lista.

No se espera a terminar la etapa de diseño para empezar la etapa de desarrollo como el modelo *Waterfall*. Tampoco se espera al final de la implementación para realizar pruebas de la aplicación completa. Esto aumenta la flexibilidad cuando existen cambios de requerimientos. Sin importar cuan exhaustiva sea la etapa de levantamiento de requerimiento, siempre van a haber detalles que solo van a surgir en la etapa de implementación (Larman y Basili, 2003)

```
def test_step_sweep(self):
    chan_command = "DE CH{ch_number},'{volt_name}','{curr_name}',2,2"
    func_command = "SS IP {start},{step},{steps},{compliance}"

    ch_number = 3
    curr_name = random_id(CurrentVoltage.CURRENT)
    volt_name = random_id(CurrentVoltage.VOLTAGE)
    start = 0.01
    step = 0.02
    steps = 15
    compliance = 0.01

    chan_command = chan_command.format(ch_number=ch_number, curr_name=curr_name, volt_name=volt_name)
    func_command = func_command.format(ch_number=ch_number, start=start, step=step, steps=steps, compliance=compliance)

    step_smu = SMUStep(ch_number, SourceMode.CURRENT, SourceType.CURRENT, start, step, steps, compliance, volt_name,
                       curr_name)

    self.assertEqual(chan_command, step_smu._get_chan_cmd())
    self.assertEqual(func_command, step_smu._get_var2_cmd())
```

Figura 3: Prueba unitaria para generación de comandos de sweep

2.6 Revisión literaria

Aunque el modelado por computadora y el software cada día sean de mayor importancia en el proceso científico, no ha existido un análisis serio de la teoría detrás del desarrollo de este tipo especifico de código como ha sido mencionado por Lane y Gobet (2012). El análisis más profundo sobre el tema fue hecho por los mismos dos autores (2012). En este análisis se explora una metodología que busca probar en cada iteración de desarrollo cuan fielmente el software se apega a la teoría científica subyacente, además de determinar relaciones más generales entre el código fuente de los programas científicos y la teoría.

Al ser el software de importancia tan grande en la investigación científica es necesario seguir prácticas de desarrollo y diseño para evitar en el futuro problemas con el mantenimiento como Banker, Davis y Slaughter mencionan (1998). También Rice y Boisvert (1996) en “From Scientific Software Libraries to Problem Solving Enviromments” han dado algunas recomendaciones en la investigación científica que cada día depende más y más del software para su

realización. Una de las recomendaciones más importantes hechas por Rice y Boisvert es la flexibilidad que debe tener el software desarrollado para actividades científicas, justificando esta flexibilidad con la posibilidad de reuso en proyectos futuros, pero sin mencionar que un énfasis demasiado grande en la flexibilidad puede demorar el desarrollo y por ende aumentar sus costos.

Además de las cuestiones de flexibilidad, mantenibilidad y apego a la teoría abordadas por los autores antes mencionados, Kohn, Kumfert, Painter y Ribbens (2000), se plantean los problemas de escalabilidad en grandes arreglos de computadoras destinados a realizar grandes cantidades de cálculos sobre datos provenientes de la investigación científica. También los mismos autores abordan el problema de la integración de la gran variedad de tecnologías diferentes utilizadas en el proceso científico y como se puede desacoplar el lenguaje de programación de la funcionalidad de una librería o programa científico.

La velocidad a la que la tecnología avanza es conocida por todos, sin embargo los proyectos de investigación científica no avanzan a esta misma velocidad, llegando a durar años e incluso décadas. Debido a esto no es extraño encontrar en la práctica sistemas de apoyo a la investigación científica utilizando lenguajes de programación tan obsoletos como Fortran77 que a su vez utilizan lenguaje novedosos como Python como interface de usuario. De aquí el problema de integración propuesto por Kohn et al: ¿cómo se puede integrar tecnologías tan diferentes en un sistema común? Según los mismos autores la solución ya está hecha e implementada en software corporativo de otras áreas, como la de negocios. Aunque los autores si aceptan que las soluciones no pueden ser simplemente tomadas del dominio empresarial y aplicadas al dominio científico de

manera directa, no hacen suficiente énfasis en la gran cantidad de problemas que lconlleva este tipo de integración.

Kohn et al también plantean el problema de la escalabilidad y la paralelización del software científico. Los mismos autores sugieren muy acertadamente aplicar las misma técnicas de escalabilidad utilizadas en el dominio empresarial en el dominio científico, ya que éste es un problema resuelto en el área de negocios. El tema de paralelización sin embargo no corre con tan buena suerte, ya que la implementación de algoritmos paralelos sigue siendo un problema abierto tan solo con unas pocas soluciones heurísticas.

Cabe mencionar que autores como Oliphant mencionan que todo el software en el dominio científico ha sido o está siendo reemplazado por programas escritos en Python. Todas las librerías utilizadas para el proceso de datos científicos y para la interacción con equipos científicos se está estandarizando como librerías para Python. Librerías que han reemplazado casi totalmente a Matlab, Octave o R. Esta es una de las razones por las que la presente tesis utiliza el lenguaje de programación Python como lenguaje principal de implementación. Python es un lenguaje especialmente bueno cuando se requiere automatizar equipo científico.

3. Solución propuesta

3.1 Ambiente de desarrollo

El desarrollo de la aplicación se realizó bajo un sistema operativo Linux (específicamente Archlinux). Como intérprete de Python se utilizó la versión 2.7.2. Para escribir los archivos de código se utilizó el editor vim. El código se mantuvo bajo control de versiones utilizando git y se mantuvo un repositorio remoto permanente en Github para monitoreo del progreso de la aplicación por parte de los profesores de ingeniería electrónica. Para mantener un ambiente de desarrollo estable independiente de actualizaciones de software y de sistema operativo se utilizó *virtualenv*. *virtualenv* permite definir ambientes virtuales para la ejecución de programas escritos en Python. El programa permite definir ambientes con nombre en carpetas independientes. Cada ambiente tiene su propio intérprete de Python con una versión específica y sus propias librerías en versiones específicas. Esto permite separar la instalación de Python del sistema operativo, de la versión de Python requerida por la aplicación. Los paquetes de Python necesarios para la aplicación se instalaron utilizando la herramienta *pip,* que viene incluida con cualquier instalación de Python.

3.2 Configuración de red de los aparatos de la computadora personal

La configuración de red se hace a través de un *switch* al que se conectan el Agilent E5071c, el Keithely K-4200-SCS y la computadora personal que controlará ambos equipos. El esquema de la red se puede observar en la Figura 4

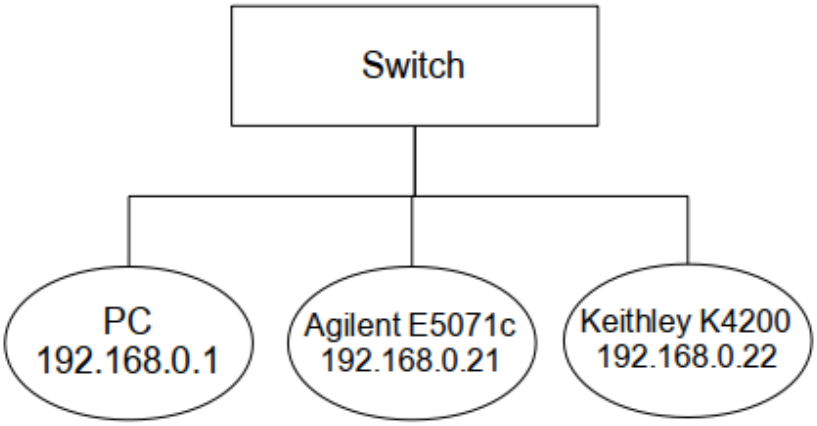


Figura 4: Diagrama de red de los dispositivos de medición y la computadora personal

3.3 Funcionalidad común

La aplicación tiene la capacidad de guardar configuraciones y abrir configuraciones antes determinadas. Esta funcionalidad es común para las interfaces del caracterizador de semiconductores y para la interfaz del analizador de redes. El usuario puede guardar una configuración en un archivo en cualquier lugar del disco duro y abrir una configuración que ha sido guardada antes. Todo esto se maneja desde una barra de menú en la parte superior de la aplicación. Los archivos tienen formato YAML y son comprimidos con gzip antes de guardarse. Las funciones que el usuario tiene a su disposición se pueden observar en la Figura 5

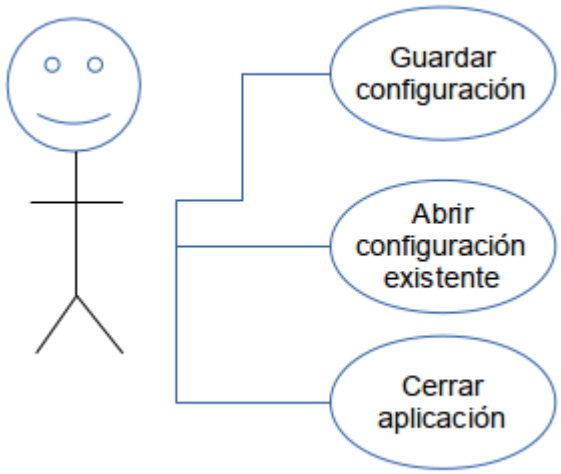


Figura 5: Diagrama de caso de uso de la funcionalidad común de la aplicación

3.4 Sección Keithley K4200-SCS

La interfaz gráfica para el Keithley K4200-SCS permite configurar SMUs en diferentes modos, y especificar la dirección IP del Keithley K4200-SCS y el archivo en donde se guardaran los datos generados por la configuración actual. También se incluye un botón que inicia el ciclo de medición del aparato. El botón se desactiva mientras el aparato se encuentra midiendo y se reactiva tan pronto la medición termina. En la Figura 6 se puede observar el botón debajo de las dos cajas de texto para la dirección IP y para el archivo.

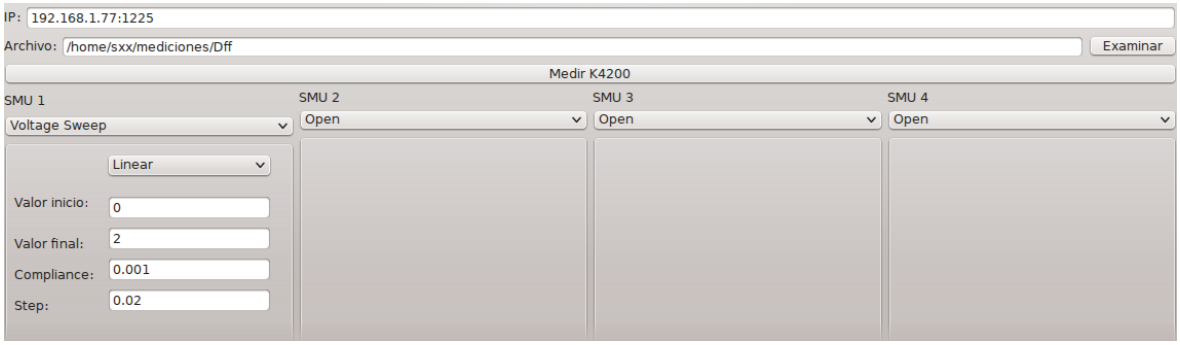


Figura 6: Interfaz gráfica para la sección de configuración para el Keithley K4200-SCS

El usuario tiene la capacidad de configurar un SMU, de configurar una dirección IP y de configurar en donde se guardarán los datos generados por el aparato. Las funciones se resumen en la Figura 7

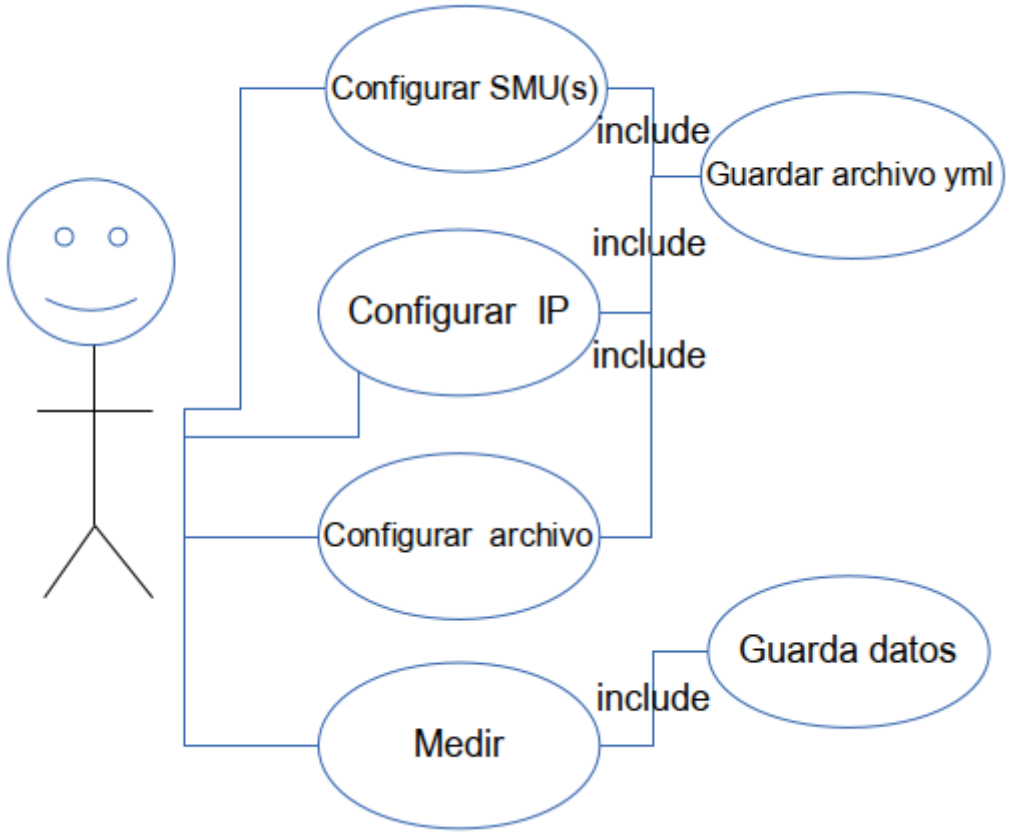


Figura 7: Diagrama de caso de usos de la interfaz para el Keithley K4200-SCS

La configuración se guarda automáticamente en un archivo en el mismo

directorio desde donde se ejecuta la aplicación. Así, cuando el usuario cierra la aplicación toda la configuración se almacena. Al momento que la aplicación se abre de nuevo, la configuración del usuario que se encuentra en disco se restaura y la interfaz gráfica se encuentra en el mismo estado en el que estaba antes de ser cerrada.

Cuando el usuario selecciona el botón *Medir K4200,* el programa genera una serie de comandos que corresponden a la configuración que el usuario ha especificado y los envía a la dirección IP especificada por el usuario anteriormente. Una vez que la medición ha finalizado, el programa guarda los datos según el nombre del archivo que ha sido seleccionado. Se añade un sufijo "_chN.csv" al nombre seleccionado por el usuario, donde N es el número de canal que ha sido usado para medir. Si la medición especifica voltaje, los datos generados son de corrientes, y si la medición especifica corriente, los datos generados son voltaje. Es posible que se generen varios archivos si varios canales están habilitados. Debido a que la presente interfaz puede configurar hasta cuatro canales al mismo tiempo, es posible que se generen hasta cuatro archivos diferentes.

A causa de limitaciones en la interfaz de programación del Keithley K4200-SCS es imposible realizar mediciones en forma de *step o sweep* en más de un canal a la vez . Es decir, es imposible tener dos canales realizando un *sweep* al mismo tiempo o dos canales realizando un *step* al mismo tiempo (Model 4200-SCS Semiconductor Characterization System Reference Manual, 2011). Esta es una limitación que no se puede superar de forma adecuada sin modificaciones a la interfaz de programación por parte del fabricante del dispositivo.

3.5 Sección Agilent E5071C

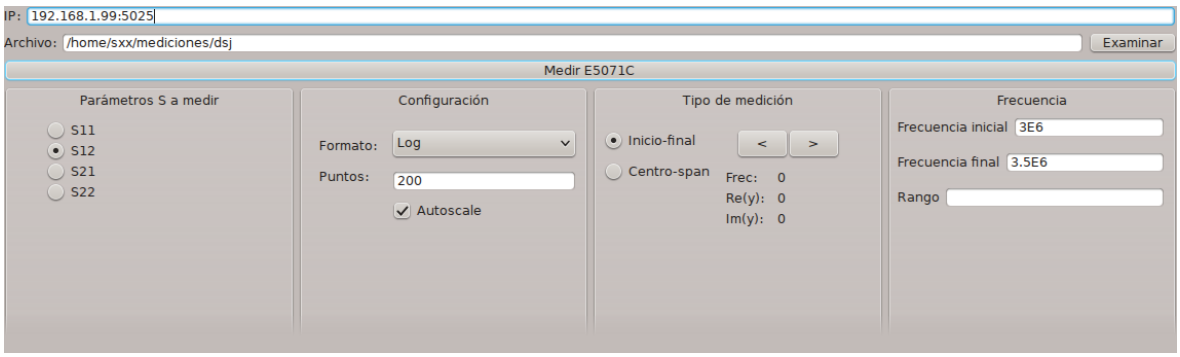


Figura 8: Interfaz gráfica de la sección de configuración del Agilent E5071c

La interfaz de configuración del Agilent E5071C permite operar un solo *trace* de medición en el analizador de redes. Permite seleccionar el parámetro de la matriz de dispersión, la cantidad de puntos de barrido, el formato de los datos a presentarse y las frecuencias entre las que se va a realizar el barrido. Cada una de estas opciones se pueden observar en la Figura 8. Además de esto permite que el usuario especifique un nombre de archivo y lugar donde se guardarán los datos de frecuencia y datos de potencia que se generen en una determinada medición. En la Figura 9 se resume la funcionalidad del módulo para el analizador de redes. Al momento que el usuario selecciona el botón *Medir E5071C* se generan comandos que configuran el VNA según los parámetros ingresados por el usuario. Debido a la naturaleza del dispositivo no es necesario esperar que una medición esté realizada para obtener los datos generados, por lo que tan pronto la configuración termina, los datos son leídos y guardados en el archivo. Los datos de potencia se graban en un archivo con sufijo "_vna.csv" y los datos de frecuencia se graban en un archivo con sufijo "_freqdata.csv". Naturalmente, el inicio del nombre del archivo es el especificado por el usuario.

Además de soportar configuración y extracción de tablas de datos para

diferentes parámetros de la matriz de dispersión, también es posible utilizar un marcador que encuentre el valor de potencia para una frecuencia específica. Debido a que algunos de los formatos de datos pueden representar la potencia utilizando números complejos, se provee tanto la parte real como la parte imaginaria de la medición en la interfaz gráfica. Esta parte de la aplicación debe funcionar en tiempo real, ya que el usuario espera mover el marcador y recibir el valor correspondiente de forma inmediata.

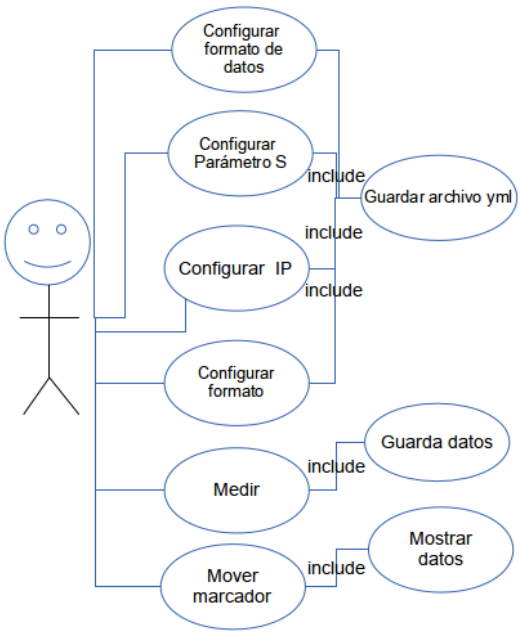


Figura 9: Diagrama de caso de uso Agilent E5071C

3.6 Calibración E5071C

En la Figura 11 se puede observar la interfaz desarrollada para la calibración. Permite realizar calibración de los tres tipos básicos: OPEN, SHORT y THRU. Una vez que el usuario selecciona un tipo de calibración utilizando el *combobox* en la parte de arriba de la interfaz se puede seleccionar el modo de

calibración que se va a realizar para ese tipo de calibración. Cuando usuario selecciona uno de los tres botones la aplicación le instruye a conectar las partes del kit de calibración para el modo de calibración seleccionado a través de una caja de texto. Una vez que la conexión está realizada, el usuario selecciona el botón OK (como se observa en la Figura 10) y el equipo procede a realizar la calibración según el tipo y modo de calibración seleccionada. Al mismo tiempo se deshabilitan el resto de botones de este cuadro de diálogo para evitar el mal-funcionamiento del aparato. La aplicación espera a que el comando de ejecución de la calibración termine, para volver a habilitar los botones que realizan la calibración. Una vez que tres de los modos de calibración para cada tipo de calibración se han realizado, la aplicación procede a guardar los coeficientes de calibración calculados. Una vez que los coeficientes se han guardado, el equipo hace uso de estos de manera automática en mediciones futuras.

Figura 10: Instrucciones de calibración

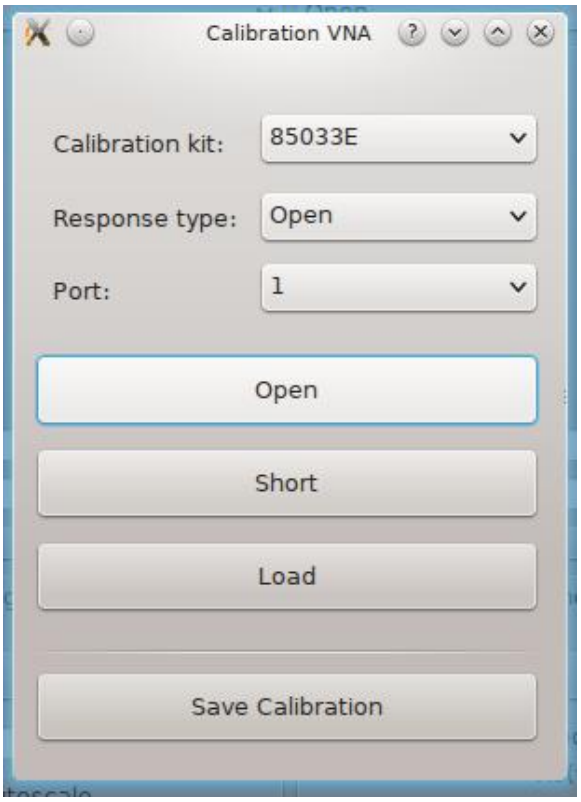


Figura 11: Interfaz de calibración el analizador de redes

3.7 Arquitectura general de la aplicación

Como se presentó en la sección anterior, la aplicación se divide en dos partes, una controla el analizador de redes y la otra controla el caracterizador de semiconductores. Sin embargo, el inicio de la aplicación es común. En la primera fase de la aplicación se crea un objeto *ui* que representa la interfaz gráfica que ha sido diseñada en QtDesigner. La clase que especifica las propiedades de este objeto se genera automáticamente a partir de un archivo creador por QtDesigner. La herramienta que convierte los archivos .uic (formateados como XML) a clases Python se llama *pyuic4 (*Using Qt Designer). Posteriormente con un objeto que es instancia de esta clase se configura la interfaz gráfica dentro de una ventana representada por un objeto de Qt llamado *QMainWindow.*

Una vez que la infraestructura básica de la aplicación se encuentra funcionando es necesario conectar las señales producidas por cada uno de los componentes de la interfaz gráfica a los métodos que se encargarán de realizar las tareas correspondientes. Todas las conexiones de señales se realizan a slots de la clase *SlotContainer.* Se crea exactamente un objeto de esta clase durante la ejecución de la aplicación y los métodos de esta clase funcionan como *slots* a los que se conectarán las diferentes señales generadas por la interfaz gráfica. Estos métodos intentan delegar la mayor parte de la funcionalidad a métodos externos implementados en archivos diferentes. En cambio, los miembros de datos de esta clase se utilizan para mantener estructuras de datos internas de la aplicación, que se encargan de guardar el estado no visible de la aplicación (por ejemplo, el nombre del archivo de configuración abierto o el número de canales activados para una medición en el caracterizador de semiconductores). En este objeto también se mantienen objetos que por motivos de desempeño deben crearse una sola vez y luego reutilizarse. Este es el caso del objeto que se comunica con el Agilent E5071c para mover marcadores. Es inadmisible en un sistema en tiempo real que cada vez que se necesita mover el marcador sobre un *trace* sea necesario crear un nuevo objeto *VnaChannel* y por ende un nuevo *socket*.

Una representación gráfica de la arquitectura de la aplicación se puede observar en la Figura 12

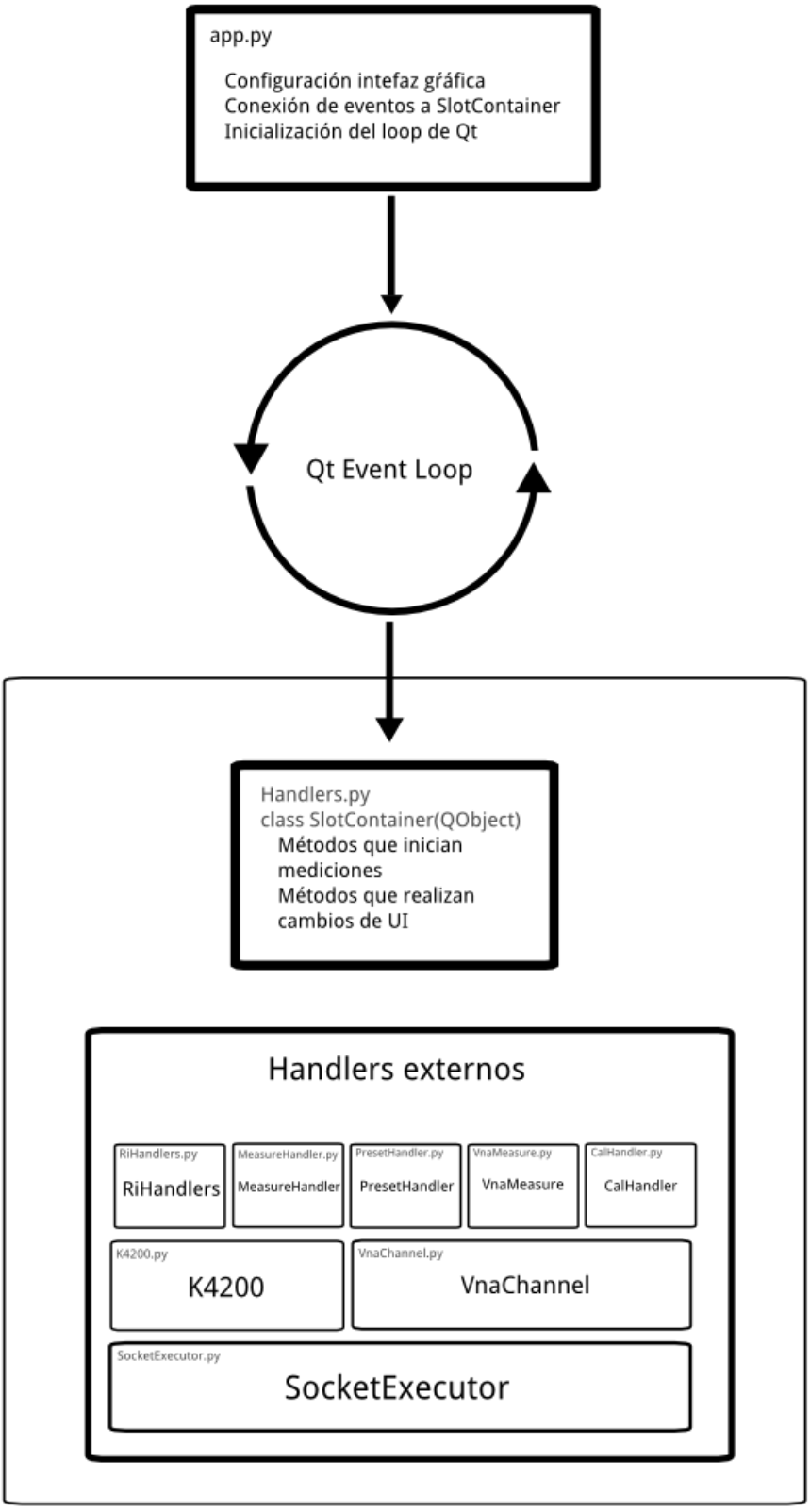


Figura 12: Arquitectura general de la aplicación

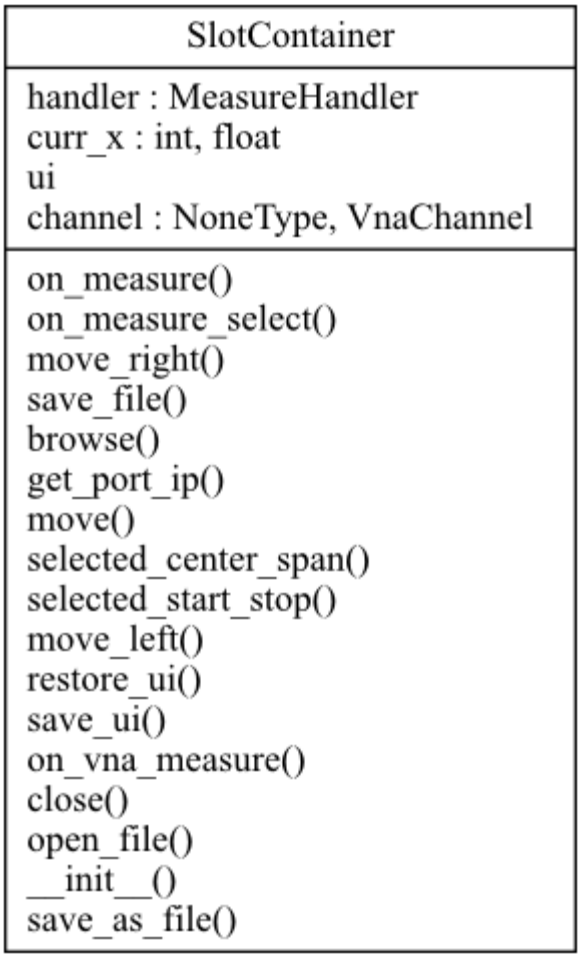


Figura 13: SlotContainer

3.7.1 SlotContainer

La clase *SlotContainer actúa* como punto central donde se encuentran todos los métodos que se ejecutan en primera instancia en respuesta a las señales de la aplicación. Al mismo tiempo este objeto brinda acceso a las estructuras de datos de la aplicación a todos los *slots* (métodos delegados) que lo requieran. Se ha intentado minimizar la cantidad de estado compartido para reducir la complejidad de la aplicación. El hecho de que los métodos de

SlotContainer (ver Figura 13) sean utilizados como *slots* hace necesario que esta clase herede de *QObject* (o de una clase derivada de ésta). Esto además provee métodos utilitarios para los *slots,* por ejemplo acceso al método estático *sender* que permite obtener una referencia al objeto que emitió una señal.

En base a su propósito, los métodos de *SlotContainer* pueden clasificarse en dos grupos:

1. Métodos que modifican los componentes de la interfaz gráfica. Una lista de estos métodos se pueden ver en la Tabla 1.
2. Métodos que realizan tareas de comunicación con los aparatos controlados. Ver la Tabla 2

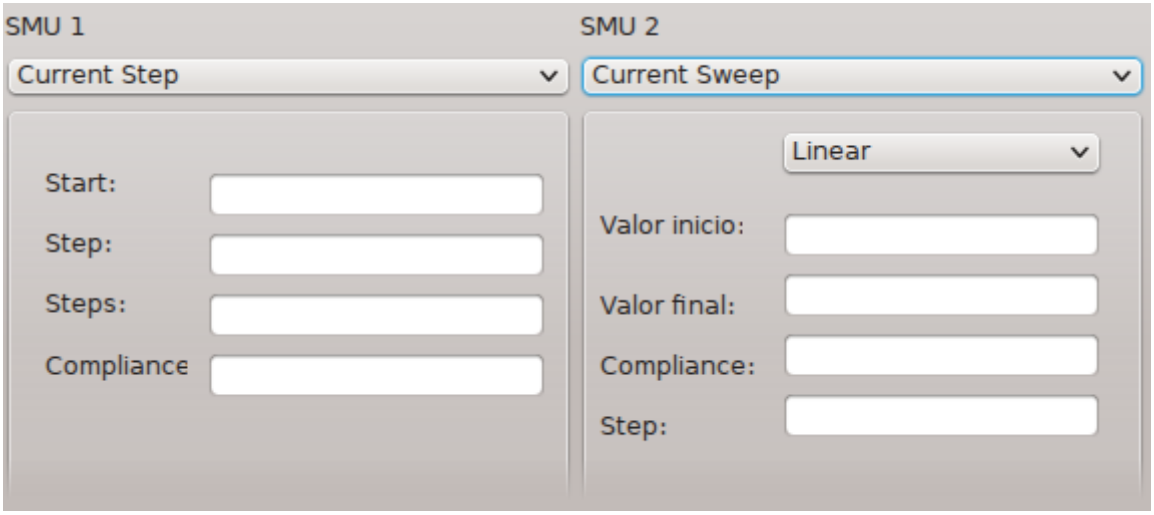


Figura 14: Interfaces diferentes para diferentes modos de operación de una unidad de fuente medición (Source Measure Unit o SMU)

Los métodos que modifican los componentes de la interfaz gráfica se usan para realizar modificaciones en tiempo de ejecución. Ejemplo de tal funcionalidad son las opciones de configuración para cada modo de operación del SMU. Es necesario mostrar interfaces diferentes cuando un SMU se encuentra en modo de barrido o cuando este se encuentra en modo constante como se puede observar

en la Figura 14

Método	Descripción
selected_start_stop	Muestra la interfaz de configuración del VNA para especificación de frecuencia como inicio-final
selected_center_span	Muestra la interfaz de configuración del VNA para especificación de frecuencia como centro-amplitud.
on_measure_select	Cambia la interfaz gráfica dependiendo del elemento del combobox seleccionado por el usuario. Se puede observar en la Figura 14

Tabla 1: Métodos que modifican la interfaz gráfica

Estos métodos reemplazan de forma dinámica los componentes de la aplicación con objetos que se encuentran definidos en otros archivos. Es decir, estos métodos se conectan a la señal generada en el cambio de selección en los *comboboxes* y reemplazan los componente de configuración con los apropiados. Para la interfaz del VNA sucede algo similar, los parámetros de configuración de frecuencia de barrido pueden especificarse en formato centro-amplitud y modo inicio-final. Es necesario modificar la interfaz gráfica en tiempo de ejecución para mostrar los diferentes componentes de configuración según sea necesario.

3.7.2 Métodos que realizan tareas de comunicación con los aparatos

Estos son los métodos que se encargan de extraer los parámetros de configuración proporcionados por el usuario y convertirlos a llamadas a las librerías que realizaran la validación de parámetros, generación de comandos y establecerán la comunicación sobre TCP/IP con los los aparatos a ser

controlados. Todas estas tareas se delegan a la librería de control escrita como parte de esta misma tesis. Los métodos clave de la capa de comunicación son parte de *SocketExecutor* y se pueden ver en la Tabla 2

Método	Descripción
__init__(self, ip, port=1225, expect_reply=True, endline="\0")	Constructor que inicializa la clase. Tiene parámetros que controlan el comportamiento del protocolo. Esto incluye IP de comunicación, puerto de comunicación, terminación de los mensajes y si es necesario esperar respuestas por parte del aparato después de la ejecución de un comandos (i.e ACKs).
execute_command(self, command)	Ejecuta comandos utilizando los parámetros configurados en el constructor del objeto.
ask(self, command)	Realiza la misma tarea que execute_command, pero además de esto lee los datos provistos por el aparato en respuesta al comando. El método devuelve la respuesta del aparato
close(self, command)	Sincroniza (flush) los buffers y cierra el socket de comunicación.

Tabla 2: Métodos que realizan tareas de comunicación

. Una tarea importante que si es realizada por estos métodos es convertir los parámetros a un formato usable por las librerías, como por ejemplo convertir cadenas de texto a sus representaciones equivalentes en punto flotante. Esto es necesario porque se debe validar que los valores de configuración estén en los rangos soportados por el aparato.

3.7.3 Generación de comandos y comunicación con los aparatos

La generación de comandos se realiza en clases totalmente separadas de la interfaz gráfica. Opcionalmente, la librería provee funcionalidad para ejecutar

los comandos a partir de una dirección IP y de un puerto de TCP. Debido a esto las funciones y clases implementadas para generar comandos pueden considerarse como una librería reutilizable por otros proyectos de software. De hecho, muchas de las pruebas realizadas durante esta tesis se realizaron utilizando las librerías desde una aplicación de líneas de comandos con código totalmente independiente de la aplicación gráfica. Esto es prueba de la independencia entre el código de generación de comandos, el código de ejecución de comando y el código relacionado a la interfaz gráfica.

3.8 Generación de comandos para el Keithley K4200 SCS

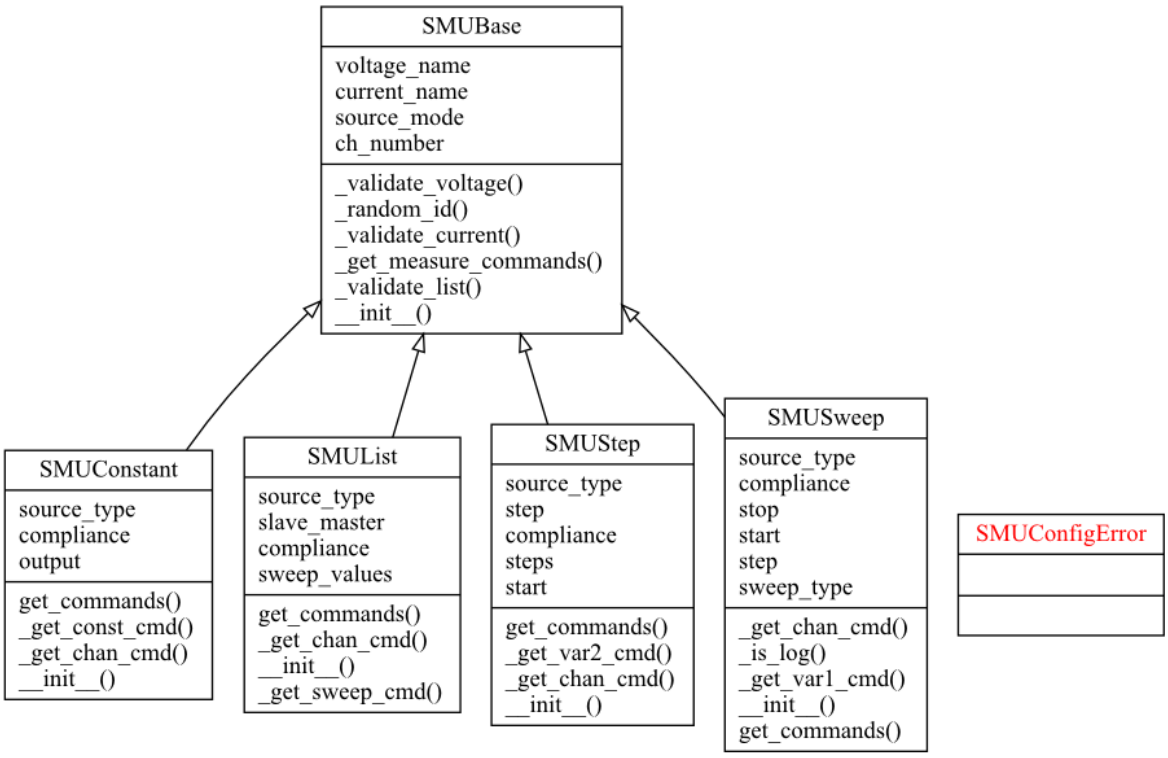


Figura 15: Diagrama de clases de la libreria de generación de comandos para el Keithley K4200

La librería de generación y ejecución de comandos para este aparato está estructurada en una jerarquía de clases (ver Figura 15). Existen clases que representan a unidades de medición-fuente en diferentes modos de funcionamiento. Cada una de estas clases heredan de una clase base que realiza tareas comunes para todos las unidades de medición-fuente. Los constructores de cada uno de los los objetos SMU se encargan de validar los datos a través de métodos en la clase base. La descripción se puede ver en la tabla 3. Se levanta una excepción si cualquiera de los parámetros de validación fallan. El aparato en sí se encuentra representado por otra clase K4200 que a su vez contiene una lista de objetos SMU. Cada instancia de las clases que representan SMUs se puede acoplar a un objeto K4200 utilizando el método *attach.* El objeto K4200 después

se encargará de pedir a cada uno de los objetos SMU los comandos necesarios para configurar un SMU al estado especificado en los miembros de datos del objeto. Es decir, las plantillas de comandos utilizadas para la generación se encuentran en estos objetos.

La descripción de cada una de las clases de generación de comandos para el Keithley K-4200 SCS se puede ver en la Tabla 3

Clase	Descripción
SMUBase	Clase base de la que derivan todas la clases de generación de comandos. Implementa métodos comunes de validación y comunicación.
SMUConstant	Clase que genera comandos para configurar un SMU con un voltaje o corriente constante.
SMUStep	Clase que genera comandos para configurar un SMU para realizar mediciones por pasos de voltaje o corriente.
SMUSweep	Clase que genera comandos para realizar barridos de voltaje o corriente.
SMUConfigError	Clase que deriva de la clase general de error de Python. Utilizada para reportar errores de configuración por parte del usuario de la librería.

Tabla 3: Clases K4200-SCS

La clase K4200 no deriva de ninguna de las clases anteriores, pero si define un error extra cuando el objeto no se utiliza de forma correcta. Esta excepción se levanta cuando se ha intentado realizar una medición sin antes configurar al menos un SMU.

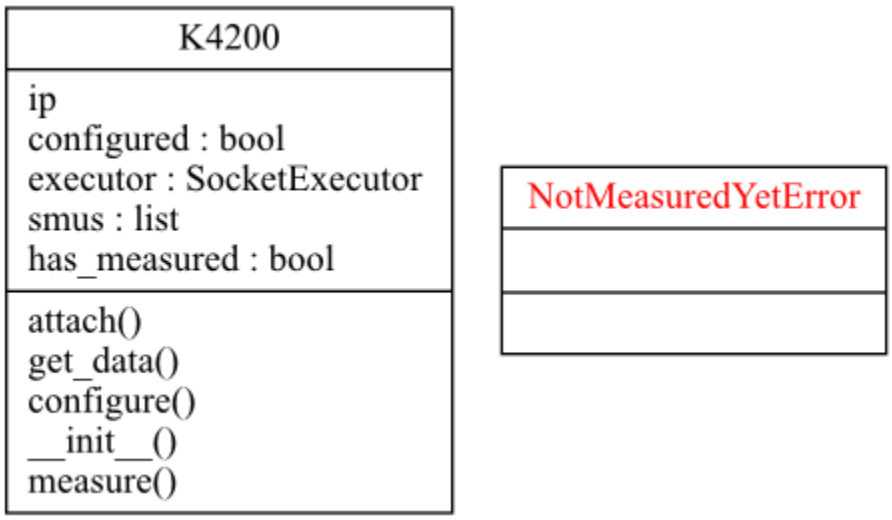


Figura 16: Diagrama de clases de objeto K4200

3.8.1 Comunicación con el K4200-SCS

El objeto que realiza la comunicación con el K4200-SCS es K4200. Como se puede observar en el diagrama de clases de la Figura 16, la clase tiene un miembro *SocketExecutor* (sobre el cual se discutirá más a profundidad más adelante) que se encarga de enviar comandos sobre una conexión TCP/IP. El objeto K4200 se encarga de ejecutar los comandos de cada uno de los SMUs en su lista a través del *SocketExecutor* que ha sido inicializado desde su constructor. Una vez que los comandos de configuración de cada SMU se han ejecutado el objeto ejecutará los comandos para realizar una medición. Una vez que todos los comandos han sido ejecutados es posible llamar al método *get_data* para ejecutar los comandos de extracción de datos del K4200-SCS.

3.8.2 Generación de comandos para el Agilent E5071C

La estructura de esta librería es diferente a la de la anterior. La configuración de este equipo es más sencilla. Simplemente es necesario un objeto que represente al aparato genere y ejecute comandos cuando se llamen a métodos de este objeto (ver Figura 17). Los métodos corresponden a diferente

funcionalidad del aparato físico. Métodos como *set_sparam(self, s)* que el parámetro S (ver Introducción) a ser medido por el aparato. La librería provee una clase de bajo nivel llamada *Vna* que permite controlar un número ilimitado de canales. Sin embargo, para propósitos de esta tesis se utiliza una clase derivada, llamada *VnaChannel,* de esta última que especifica un solo canal. Esta clase derivada facilita de manera sustancial el manejo del Agilent E5071c. Específicamente, evita la necesidad de indicar a que canal se refiere cada acción realizada sobre el VNA. La funcionalidad de cada uno de los métodos de *Vna* se puede observar en la Tabla 4.

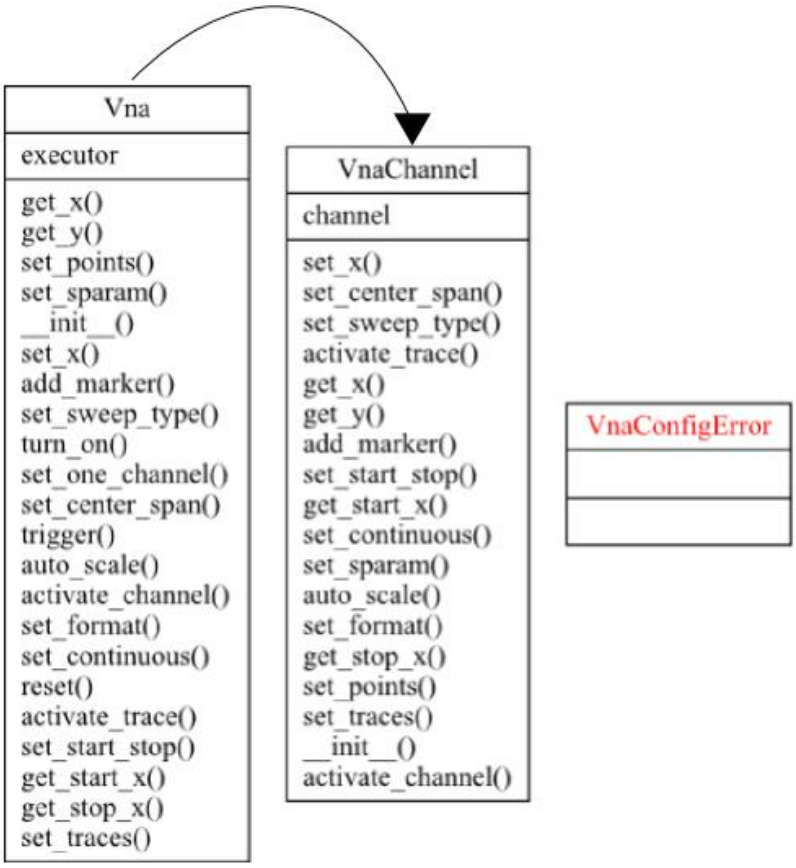


Figura 17: Diagrama de clases de la libreria de generación y ejecución de comandos del VNA. VnaChannel es una subclase de Vna

La clase *VnaChannel* deriva de la clase *Vna* y especifica un solo canal en todos sus métodos. Cada uno de los métodos de la clase base define una plantilla del comando necesario para cierta acción y utiliza la plantilla para generar los comandos que correspondan a los argumentos pasados al método. Después el mismo método utiliza el *SocketExecutor* miembro de esta misma clase para ejecutar los comandos en contra del aparato. Cabe decir que el objeto *SocketExecutor* encapsula la dirección IP y el puerto TCP que se usara para la comunicación. Este objeto es inicializado en el constructor de *VnaChannel.*

Método	Descripción
get_x(self, x)	Devuelve la posición actual en el eje *x* del marcador principal.
get_y(self, y)	Devuelve la posición actual eje *y* del marcador principal.
set_points(self, points)	Configura la cantidad de puntos de la medición.
set_sparam(self, param)	Selecciona el parámetro S que se va a medir.
set_x(self, x)	Configura la posición del marcador principal en el eje x.
add_marker(self, name)	Añade un marcado con nombre *name.*
set_sweep_type(self, stype)	Configura el tipo de sweep. Sweep center-span o sweep start-end.
turn_on(self, port)	Activa la salida en el puerto *port.*
set_one_channel(self)	Configura el objeto para usar un solo canal.
set_center_span(self, center, span)	Configura el centro y la amplitud de una medición.
trigger(self)	Habilita la función de trigger del aparato.
auto_scale(self)	Utiliza la función de escala automática en el eje *y.*
activate_channel(self, ch)	Activa un canal determinado y lo

	configura como predeterminado.
set_format(self, format)	Configura el formato de datos. Formatos como logarítmico o varias cartas de Smith.
set_continuous(self)	Instruye al aparato a mantener un modo de inicialización continuo.
reset(self)	Configura al aparato a la configuración inicial de encendido.
activate_trace(self, trace)	Activa un trace dado por el número *trace*
get_start_stop(self, start, stop)	Devuelve los valores de inicio y final de un sweep

Tabla 4: Métodos de Vna y VnaChannel

3.9 Comunicación con el Agilent E5071C

A diferencia del Keithley 4200-SCS, este aparato no se configura para luego realizar una medición. La configuración se da en tiempo real. Es decir, una vez que se modifica un parámetro, éste se ve reflejado inmediatamente en la configuración del VNA. Por tanto, el modelo de clases utilizado en el caracterizador de semiconductores no es apropiado para este aparato. La comunicación se debe dar de forma instantánea cuando se llama a cada uno de los métodos del objeto *VnaChannel.* Debido a esto los objetos de comunicación también reutilizan sockets entre comandos. Es decir, es necesario minimizar la latencia entre los comandos del usuario a la interfaz gráfica y los comandos que llegan al analizador vectorial de redes. Esto es especialmente importante en la funcionalidad de marcadores.

3.10 Calibración del Agilent E5071C

La calibración del VNA se encuentra implementada a través de métodos en *VnaChannel*. Los métodos en la *Vna* encapsulan los comandos SCPI para cada una de las operaciones en el VNA. Los métodos se pueden observar en la Tabla

4. Cuando el usuario abre el módulo de calibración de la aplicación el *slot* conectado a esta acción, se encarga de conectar las señales de los tres botones principales de la Figura 11 a los *slots* correspondientes. Cada uno de los *slots* lanza un nuevo hilo de ejecución que se encargará de llamar a los métodos de *VnaChannel* en el orden correspondiente. Es necesario ejecutar los comandos en un hilo de ejecución paralelo debido a que cualquier llamada que bloquee (como comunicación sobre un socket) paralizará la interfaz gráfica. Las actividades que se deben realizar para calibrar el aparato, en orden, son:

1. Selección del kit de calibración (solo el Agilent 85033E soportado en el presente desarrollo)
2. Selección del tipo de calibración (SHORT, OPEN y THRU)
3. Selección del modo de calibración (SHORT, OPEN y LOAD)
4. Medición de coeficientes de calibración

Antes de ejecutar la calibración a través de los tres pasos anteriores cada uno de los *slots* lanzan un hilo de ejecución en paralelo que revisará cada segundo si los coeficientes de calibración han terminado de calcularse. Este hilo rehabilita los botones del cuadro de dialogo de calibración una vez que una calibración específica ha terminado. Es necesario deshabilitar los botones de calibración cuando una calibración ya se encuentra en marcha debido a que si se intenta ejecutar ambas calibraciones al mismo tiempo el aparato responderá con un error. Además se cancelará la calibración en curso.

Cada uno de los hilos responsables de revisar si la calibración ha terminado para rehabilitar los botones de calibración, también es responsable de

revisar si los tres modos de calibración se han ejecutado y si es así, guardar la información de calibración a través del comando correspondiente.

3.11 Capa de comunicación

Cada uno de las clases antes descritas dependen de un objeto *CommandExecutor* que provea comandos *execute_command* y *ask* para la comunicación con el aparato. Específicamente las clases *K4200* y *VnaChannel* (en virtud de herencia de *Vna*) tienen como miembro un objeto *Executor* responsable de los detalles de bajo nivel de comunicación . Debido a esto se diseño una librería que herede de una clase base *CommandExecutor.* Las clases derivadas así implementan la comunicación con el aparato de formas específicas. Por ejemplo, para la presente tesis se utilizaron dos clases derivadas de *CommandExecutor* dependiendo de lo que se busca hacer: *MockExecutor* y *SocketExecutor*. Esto provee interfaces uniformes para ejecución de comandos para el resto de la aplicación y esconde a la implementación interna de la comunicación. La comunicación comprende tanto la ejecución de comandos como la lectura de datos producidos por el aparato.

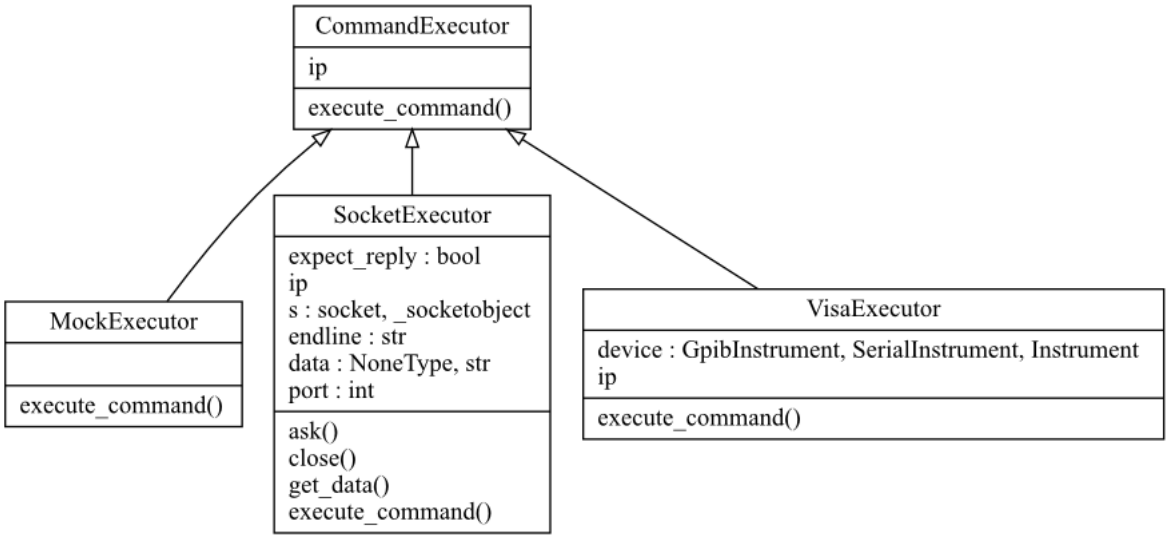


Figura 18: Diagrama de clases de CommandExecutor

En la Figura 18 se puede observar además una clase derivada de *CommandExecutor* llamada *VisaExecutor* que utiliza la librería PyVisa para la ejecución de comandos. La librería se encuentra implementada en la presente tesis, pero no es usada en el código debido a que no es compatible con dispositivos fabricados por Keithley.

3.11.1 SocketExecutor

Este es el objeto utilizado para ejecutar comandos a través de TCP. El objeto se encarga de tomar los comandos y según parámetros configurados a través del constructor enviar comandos que puedan ser comprendidos por el aparato que se esté controlando. Detalles en el protocolo de comunicación con los aparatos si dependen del aparato que se esté controlando. Por ejemplo, el Agilent E5071c requiere que todos los comandos que se ejecuten tengan un carácter LF (linefeed) al final de la cadena de texto que representa al comando. En cambio, el Keithley 4200-SCS requiere que todos los comandos terminen en un byte nulo (o “\0”) al estilo de las cadenas de texto del lenguaje de programación C. Debido a

esto el carácter que representa el fin de un comando debe ser configurable en la clase *SocketExecutor*. De la misma forma, el carácter que indica que los datos han terminado de leerse es variable entre dispositivos, por lo que también debe ser configurable en la clase *SocketExecutor*.

Como el nombre de la clase lo indica, la comunicación con los dispositivos se realiza a través de un socket provisto por la librería estándar de Python. Existen algunas consideraciones especiales que se deben tener en cuenta cuando se utilizan sockets sobre el protocolo TCP para comunicación. En primera instancia es necesario proveer métodos para que los usuarios de instancias de *SocketExecutor* puedan cerrar la conexión cuando hayan terminado de enviar comandos y de leer datos. Esto debido a que Python utiliza recolección de basura no determinística, por lo que no es posible garantizar que las referencias al socket utilizado por *SocketExecutor* serán cerradas cuando las variables que lo referencian salgan fuera de alcance. Es decir, es imposible escribir código tipo Resource Allocation Is Initialization (RAII) al estilo de C++ en Python. El lenguaje no garantiza que los métodos *finalize* definidos en una clase sean llamados de forma oportuna. En segunda instancia, la aplicación desarrollada requiere que los comandos que se envían a los aparatos sean enviados tan pronto como el usuario hace click sobre un elemento de la interfaz gráfica. Es bien conocido que TCP espera a llenar un cache antes de enviar paquetes a través de la red, de acuerdo al algoritmo de Nagle. La configuración específica del cache depende del sistema operativo. Debido a que los comandos son cadenas de texto de máximo 30 bytes, el algoritmo de Nagle no permite que los comandos se envíen de forma inmediata. Debido a esto es necesario deshabilitar el algoritmo de Nagle utilizando la opción

TCP_NODELAY provisto por el stack de TCP del kernel del sistema operativo. Esta opción se puede configurar utilizando *setsockopt* sobre el objeto *socket* correspondiente.

También es necesario tener en cuenta que los métodos para recibir información desde el *socket* no devuelven todos los datos con la primera llamada. Todavía más, la cantidad de veces necesarias para leer todos los datos es variable, por lo que es necesario verificar que se ha llegado al final del *stream* de datos. La clase *SocketExecutor* soluciona este problema buscando un carácter que indique el final de la transmisión y sale de un lazo infinito cuando esto sucede. La propiedad de la clase tiene como nombre *endline*.

3.11.2 MockExecutor

Para cuestiones de pruebas se implemento un *CommandExecutor* que en lugar de ejecutar comandos a través de una conexión de red simplemente imprime los comandos a *STDOUT* para propósitos de depurado. Es útil ver qué comandos están siendo ejecutados. El objeto ignora cualquier parámetro en el constructor y no hace más que imprimir el comando que le ha sido enviado. Gracias al polimorfismo de Python es posible reemplazar *SocketExecutor* con *MockExecutor* en el constructor de *K4200* o *Vna* y ver los comandos que serían ejecutados en el dispositivo. Las capacidades de prueba de *MockExecutor* son un tanto limitadas debido al hecho de que responde con cadenas de texto vaciás a cualquier petición de datos. Por esto, *MockExecutor* es más útil cuando se están probando comandos de configuración y no comandos de extracción de datos.

3.11.3 VisaExecutor

Existe una librería para Python llamada PyVISA que es capaz de conectarse a un dispositivo sobre TCP, escribir comandos y leer datos generados por el aparato. Es decir, PyVISA hace exactamente lo mismo que *SocketExecutor*. Sin embargo, esta librería no funciona de manera correcta cuando se utiliza con equipos de marca Keithley, debido a que éstos no se adhieren al estándar VISA. Aunque la librería es código abierto es imposible modificar el comportamiento de la misma debido a que depende de una librería escrita en C que se distribuye exclusivamente como un archivo binario al que se realizan llamadas durante la ejecución. Además de este problema, la librería solo se provee como un binario de 32 bits por lo que restringiría la aplicación a un sistema operativo de 32 bits cada vez menos común.

VisaExecutor fue implementado como parte de esta tesis, pero a causa de los problemas antes mencionados no se utiliza en ninguno de los módulos de la aplicación. *VisaExecutor* es simplemente una clase envolvente (o wrapper como se conoce en inglés) alrededor del objeto *Instrument* provisto por PyVISA. La idea es exponer la misma interface uniforme de los otros *CommandExecutor*s y utilizar VISA al mismo tiempo. Es posible reemplazar *SocketExecutor* con *VisaExecutor* siempre y cuando el programa se ejecute bajo un ambiente de 32 bits y no se utilice con equipos de marca Keithley.

4. Análisis de Resultados

4.1 Descripción de resultados

Una vez terminado el desarrollo de la aplicación para control del Sistema Caracterizador de Semiconductores y del Analizador de Redes para el Laboratorio de Microelectrónica y Radiofrecuencia de la USFQ se puede revisar que los objetivos propuestos se han cumplido. De nuevo, los objetivos planteados fueron los siguientes:

1. Desarrollar una librería orientada a objetos en Python que permita el control programático del Sistema Caracterizador de Semiconductores Keithley K4200-SCS desde un programa escrito en Python o C.
2. Desarrollar una librería orientada a objetos en Python que permita el control programático del Analizador Vectorial de Redes Agilent E5071C desde un programa escrito en Python o C
3. Desarrollar una interfaz gráfica unificada que permita utilizar el VNA y el caracterizador de semiconductores de forma simultanea.
4. Desarrollar una interfaz gráfica para la calibración del VNA

Aunque no se planteó conocer el funcionamiento del protocolo de comunicación del Keithley K4200-SCS, una de las partes más complicadas en el desarrollo de la aplicación ha sido determinar como funciona el protocolo de comunicación, ya que muchos detalles no están documentados en el manual. Estos detalles se descubrieron a través de prueba y error. Detalles como el hecho que el Keithley K4200-SCS espera que todos los comandos recibidos estén terminados con un carácter nulo. Esto no se encuentra documentado en el

manual de programación del equipo. De igual forma, tampoco se encuentra documentado el hecho que la transmisión de datos de una medición termina con un carácter nulo. Documentación sobre los protocolos de comunicación facilitaría de forma sustancial el trabajo de desarrolladores en el futuro.

A continuación se realiza la verificación de las metas:

Meta	Estado
Diseñar una librería orientada a objetos para el control del caracterizador de semiconductores. Ver Figura 15 y Figura 16	Cumplida
Diseñar una librería orientada a objetos para el control del analizador vectorial de redes. Ver Figura 17	Cumplida
Escribir pruebas unitarias con los resultados deseados en la librería utilizada para controlar el caracterizador de semiconductores. Ver Figura 3	Cumplida
Implementar las clases de la librería para el caracterizador de semiconductores. Ver Figura 16 y Figura 17	Cumplida
Implementar las clases de la librería para el analizador vectorial de redes. Ver Figura 17	Cumplida

Utilizar las librerías antes desarrolladas para controlar el estado del aparato a través de un programa escrito en Python. Ver Figura 13	Cumplida
Diseñar una interfaz gráfica que contenga los elementos necesarios para el control simultaneo del VNA y del caracterizador de semiconductores. Ver Figura 19	Cumplida
Diseñar una interfaz gráfica para la calibración del VNA. Ver Figura 20	Cumplida
Utilizar las librerías antes desarrolladas desde la interfaz gráfica para controlar el analizador de redes y el caracterizador de semiconductores. Ver capítulo 3	Cumplida
Utilizar una interfaz gráfica para llamar a métodos de la librería a través de una interfaz amigable. Ver capítulo 3	Cumplida

Tabla 5: Metas

4.2 Objetivo 1

Efectivamente se ha desarrollado una librería orientada a objetos que se puede utilizar para controlar el caracterizador de semiconductores. En la Figura 15 y en la Figura 16 se pueden ver las clases que han sido desarrolladas para interactuar con el caracterizador de semiconductores sin tener que recurrir a sockets o al uso directo de comandos. Las clases implementadas son:

- K4200
- SMU
- SMUSweep
- SMUConstant
- SMUStep

La funcionalidad expuesta por la librería es la siguiente:

1. Barridos de voltaje o corriente;
2. Barridos de voltaje o corriente por pasos;
3. Barrido de listas de voltajes o de corrientes;
4. Corrientes o voltaje constante;
5. Recuperación de datos del aparato.

Aunque la librería se encuentre escrita en Python es posible utilizarla desde el lenguaje de programación C gracias a las amplias facilidades de interacción con C que Python provee (por ejemplo a través del C Foreign Function Interface de Python). Esta librería además incluye una serie de pruebas unitarias que se pueden utilizar para desarrollo futuro seguro. Es importante notar que los métodos de generación de comandos no dependen de los métodos de ejecución de comandos en esta librería, por lo que es posible utilizarlos en otras librerías si fuera necesario.

4.3 Objetivo 2

Se desarrolló una librería que permite la utilización de las funciones básicas del analizador de redes desde un programa remoto. Las funciones básicas implementadas incluyen:

1. Marcadores (variables dependiendo del formato de datos elegidos);
2. Calibración Open, Short y Through;
3. Configuración centro-amplitud;
4. Configuración inicio-final;
5. Configuración de parámetro S a medir;
6. Configuración de formato de datos (logarítmico, lineal, varios formatos de carta de Smith);
7. Configuración de auto-escalamiento;
8. Extracción de datos del aparato.

El usuario de la librería solo debe proveer la dirección IP y el puerto donde se esta ejecutando el servidor de comandos remoto del VNA y después utilizar los métodos provistos por la librería para realizar mediciones. En la Figura 17 se pueden observar los métodos del objeto *VnaChannel*, que constituye la parte central de la librería. A diferencia de la librería para el caracterizador de semiconductores, la generación de comandos y la ejecución de los mismos se realizan dentro del mismo método, por lo que es imposible utilizarlos de manera independiente.

De la misma forma que en la librería para el caracterizador de semiconductores es posible utilizar C como lenguaje de programación debido a la amplia funcionalidad de interacción con C provista por la librería estándar de Python.

4.4 Objetivo 3

Se implementó una interfaz gráfica que hace llamadas a las librerías desarrolladas al principio de la presente tesis. Esta interfaz contiene elementos para el caracterizador de semiconductores y para el analizador de redes. La interfaz completa se puede observar en la Figura 19. Como se puede observar en la parte de arriba se encuentran los controles relacionados con el K4200-SCS y en la parte de abajo los controles relacionados con el E5071c. Cada una de las partes tiene cajas de texto independientes para la dirección IP y puerto en el que está configurado el aparato y para el archivo en el que se va a guardar los datos generados en una medición.

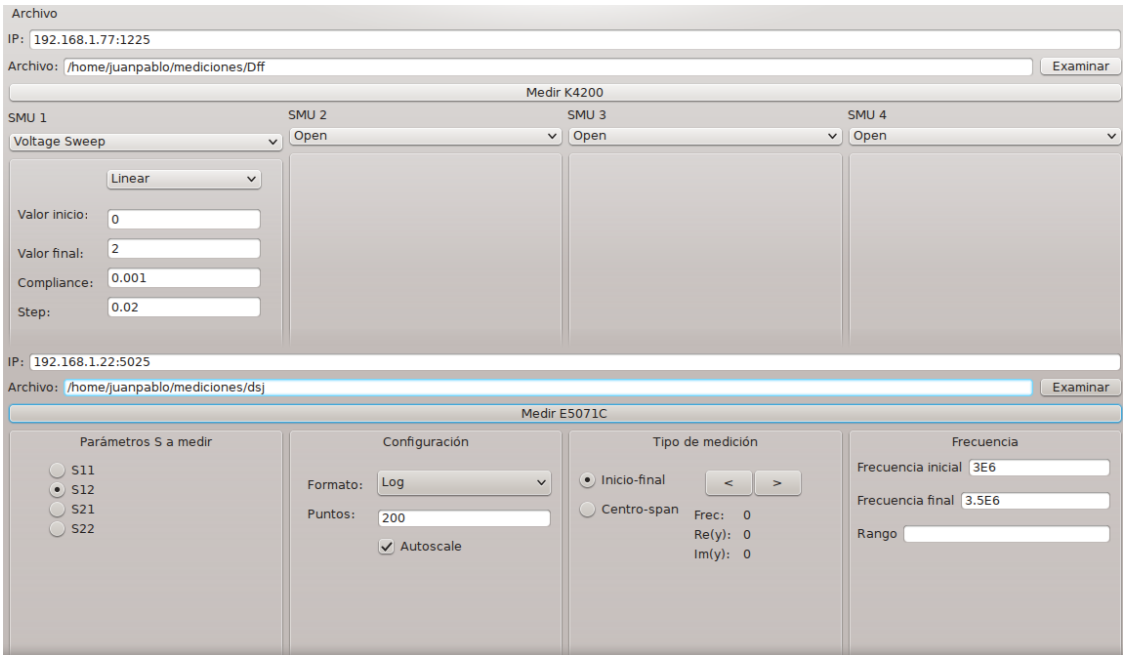
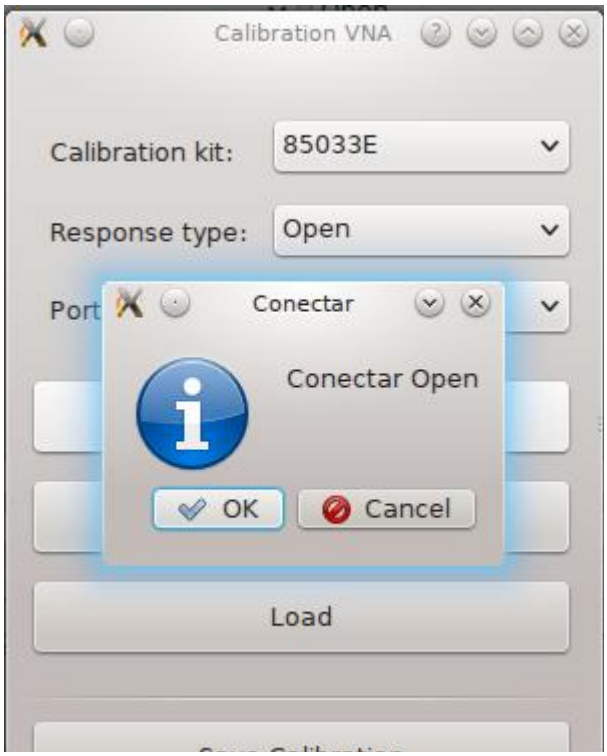


Figura 19: Interfaz completa

4.5 Objetivo 4

Se planteó implementar una interfaz gráfica para la calibración del E5071c. La interfaz implementada se puede ver en la Figura 11. Esta interfaz permite realizar la calibración básica necesaria para realizar mediciones precisas. Es posible ingresar al cuadro de diálogo de calibración a través del menú superior de la interfaz gráfica. El menú se puede observar en la Figura 21 . Como se ve en la Figura 11 se puede calibrar con los métodos Open, Short y Through para cada uno de los puertos, además se puede seleccionar el modelo del kit de calibración que se va a utilizar para calibrar. Una vez que se ha realizado la calibración se puede utilizar botón “Save Calibration” para guardar la calibración realizada. El VNA por si solo se encarga de utilizar los valores de calibración guardados para compensar en las mediciones en el futuro. Gracias a esta funcionalidad no es necesario guardar los valores generados por la calibración en el computador.



La interfaz de calibración desarrollada guía al usuario paso a paso para calibrar el aparato. Una vez que el usuario selecciona el tipo de calibración que desea realizar, la interfaz le indica que parte del kit debe conectar para completar cada tipo de calibración. Una parte del proceso de calibración se puede observar en la Figura 20

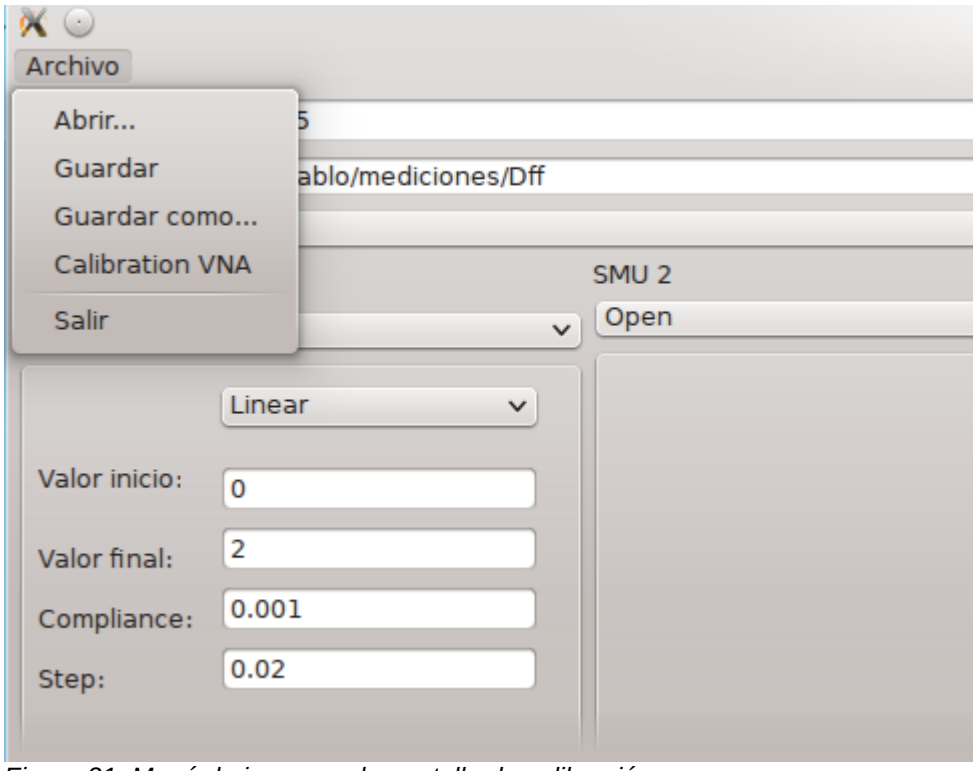


Figura 21: Menú de ingreso a la pantalla de calibración

5. Conclusiones y recomendaciones

5.1 Conclusiones

- Actualmente existe una interfaz remota para control del analizador de redes y del caracterizador de semiconductores que permite realizar mediciones sin necesidad de interactuar con los aparatos de forma directa. Ahora ni siquiera es necesario estar en el mismo laboratorio para realizar experimentos. Todo se puede realizar remotamente.
- Actualmente es posible realizar mediciones y guardar los datos producidos por estas mediciones en un computador remoto. Ya no es necesario transferir los datos a una computadora personal una vez que se han realizado las mediciones de un experimento. Los datos se transfieren de forma automática una vez que la medición ha terminado.
- Los valores utilizados para las mediciones son validados antes de ser enviados al aparato. La interfaz gráfica diseñada no permite que se generen comandos con valores fuera del rango soportado por el aparato y así evita el mal uso de los dispositivos del laboratorio.
- Ahora es posible guardar configuraciones específicas de los aparatos en archivos en disco e intercambiarlas con otros usuarios. Esto hace sumamente fácil replicar experimentos entre diferentes científicos que manejen el mismo aparato de medición.
- Ahora es posible calibrar el analizador de redes utilizando una interfaz gráfica mucho más sencilla que la interfaz provista por el fabricante. Como ya se mencionó, la interfaz de calibración desarrollada guía al usuario paso

a paso en el proceso de calibración.

- La librería escrita puede ser utilizada para implementar interfaces alternativas para el caracterizador de semiconductores que cumplan propósitos diferentes. Por ejemplo se podría implementar una interfaz que solo realice barridos de voltaje y no barridos de corriente. Todo esto sin necesidad de generar comandos con la sintaxis descrita por el fabricante del aparato. La generación de comandos es hecha por la librería.
- La librería escrita para el analizador de redes se podría utilizar para implementar programas que realicen mediciones de forma programática en el analizador de redes. Por ejemplo, es posible escribir un programa que ejecute la misma prueba todas las veces que es ejecutado. Así, se pueden automatizar experimentos sin necesidad de escribir ningún comando SCPI ni lidiar con la comunicación sobre *sockets*. La librería desarrollada se encarga de todos estos detalles de bajo nivel.
- Debido a que muchas de las operaciones realizadas sobre los equipos toman largos tiempos es posible que las llamadas a un *socket* para leer información bloqueen la ejecución del programa debido a que los datos no están listos. Por esto, es necesario utilizar procesamientos con varios hilos para evitar el bloqueo de la interfaz gráfica. Además es necesario notificar al usuario cuando se están realizando operaciones largas en otros hilos de ejecución.
- La capa de comunicación es re-usable para cualquier instrumento científico que espere comunicación sobre el protocolo TCP/IP como capa de

transporte y capa de red, respectivamente. Es decir, en futuros trabajos de automatización de experimentos es posible utilizar esta capa de comunicación, aunque los equipos que se estén automatizando no sean los mismos.

- El uso de pruebas unitarias para la implementación de la librería de comunicación con el caracterizador de semiconductores hace posible modificar la librería y añadir más funcionalidad con la confianza que la generación de comandos no ha dejado de funcionar de forma correcta. Es decir, es posible verificar que los cambios son correctos si las pruebas unitarias pasan.
- Aunque la librería escrita para el analizador de redes solo soporte un canal (es decir, una sola medición a la vez) se deja la posibilidad abierta de ampliar el soporte a varios canales. Esto gracias al hecho de que la funcionalidad de un solo canal esta implementada en una clase derivada. Solo basta derivar la clase de nuevo y re-implementar los métodos para varios canales.
- Las librerías desarrolladas permiten la automatización de los equipos sin necesidad de aprender la sintaxis de SCPI para el analizador de redes o la sintaxis definida por Keithley para el caracterizador de semiconductores. Lo único necesario es el conocimiento del lenguaje de programación Python.
- Durante la tesis fue necesario experimentar para descifrar como funciona el protocolo propietario de comunicación de Keithley. El protocolo no se encuentra documentado en ningún lugar. Una parte sustancial del trabajo

práctico en esta tesis fue la realización de ingeniería inversa del protocolo y su implementación dentro de la capa de comunicación.

5.2 Recomendaciones

- Es necesario definir el alcance del desarrollo de la aplicación y llegar a un acuerdo entre el usuario final de la aplicación y el desarrollador, para evitar que se siga añadiendo funcionalidades que no se tenían previstas. Por tanto es fundamental que el alcance se encuentre documentado para saber con certeza cuando ha concluido el desarrollo de la aplicación y el momento en que la aplicación se puede considerar lista.

- Es recomendable planear la arquitectura de la aplicación antes de empezar el desarrollo. Se puede hacer estimaciones de tiempo de desarrollo más fácilmente cuando se sabe cuáles son los componentes de la aplicación que se deben desarrollar.

- Se recomienda la interacción permanente con los usuarios finales de la aplicación, para definir detalles de la funcionalidad. Por ejemplo, los usuarios solicitaron que los marcadores en el analizador de redes puedan ser movidos utilizando unicamente el teclado. Pequeños detalles como estos no pueden ser previstos por el desarrollador de la aplicación, sino que se obtienen a través de la retro-alimentación de los usuarios.

- Aunque siempre se considera la mayor cantidad de escenarios posibles para el uso de la aplicación, siempre se descubren escenarios nuevos una vez que la aplicación se pone a funcionar con los usuarios finales de la misma. Cada escenario corresponde a un caso de uso que pudo no haber

sido considerado. Por esto se recomienda realizar pruebas uso de la aplicación con los usuarios finales lo más seguido posible.

- Limitar el alcance de algunas funciones puede hacer el desarrollo de la aplicación mucho más rápido. Por ejemplo, para el desarrollo de la aplicación del analizador de redes se limitó el número de canales a uno, debido a que los usuarios no necesitan realizar varias mediciones al mismo tiempo. Debido a esto es recomendable mantener contacto constante con los usuarios
- Se puede reducir funcionalidad en una parte del proyecto para aumentar funcionalidad en otra parte. Una vez que se limitó la funcionalidad en el analizador de redes a un único canal hubo tiempo y recursos para implementar la interfaz de calibración. Por esto es recomendable dar prioridad a diferentes partes de la aplicación.
- La implementación de *mocks* u objetos que emulan el comportamiento de otros objetos permite depurar el comportamiento de la aplicación sin necesidad de tener los equipos presentes. Por eso se recomienda utilizar un framework de *mocking* si se planea hacer desarrollo guiado por pruebas unitarias. Especialmente si el acceso a equipos es limitado.
- Mantener todo el código fuente en un repositorio remoto permite a los usuarios de la aplicación revisar de forma continua el avance en el desarrollo. Se recomienda utilizar una plataforma de compartición de código como Github.

5.3 Desarrollo futuro

Existen comandos expuestos por KXCI que no han sido encapsulados en métodos Python. En el futuro se podrían desarrollar métodos que encapsulen esta funcionalidad. Por ejemplo, no se implementaron métodos para los diferentes comandos utilizados para iniciar una medición. En la presente tesis siempre se inicia la medición utilizando “MD DO1”, pero KXCI expone tres métodos diferentes con diferencia sutiles entre ellos. La librería podría exponer métodos diferentes para cada uno de estos modos de medición. Aunque no es indispensable para realizar experimentos, podría ser útil.

En la librería desarrollada para el analizador de redes E5071c siempre se asume que se va a utilizar un solo canal. Sobre el trabajo realizado en esta tesis es posible implementar una clase que permita usar más de una canal a la vez. De igual forma, la librería asume que solamente se va a utilizar un *trace* y los métodos expuestos no permiten cambiar esto. En el futuro se podría derivar la clase *Vna* e implementar una clase que permita configurar varios canales y varios *traces* a la vez.

Como se indicó en varias partes de la tesis, algunos métodos bloquean hasta que los datos se encuentren listos. En el futuro se podría re-implementar estos métodos de tal forma que acepten una función *callback* que será llamada cuando la operación que bloquea haya terminado. También sería posible implementar el protocolo en un framework asíncrono como Twisted. Esto eliminaría la complejidad del uso de hilos de ejecución en los programas que utilizan las librerías.

6. Bibliografía

Banker R., Davis G y Slaughter S. Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study. *Managment Science.* 433-450

Basili, Victor y Craig Larman. (2003). Iterative and incremental development: A brief history. IEEE Computer Society.

Ben-Kiki Oren y Clark Evans. YAML Ain't Markup Language (YAML™) Version 1.2. 2009.

Booch, Grady. Object Oriented Analysis and Design with Applications. 3Era edición. Pearson Education. 2007

Caspers, Fritz. RF Engineering Basic Concepts: Basic Concepts: S Parameters. Organisation européenne pour la recherche nucléaire. 2007

Córdova, Francisco. Redes II: Ethernet. Universidad San Francisco de Quito.

GZIP file format specification (1996). RFC 1952. IESG Network Working Group

Hafok, Heiko, Dirk Muders y Michael Oldberg (2006). APEX SCPI socket command syntax and backend data stream format. Atacama Pathfinder Experiment.

Hickey. A.M. (2007). Requirement Elicitation Technique Selection: how do experts do it? Requirement Engineering Conference. Proceedings. 11th IEEE International

Hilsheimer, Volker. Academic Solutions to Academic Problems. Qt Quarterly. <http://doc.qt.digia.com/qq/qq15-academic.html#theobserverpattern>.

IEEE488. IEEE-488 interface bus (HP-IB/GP-IB). IEC 60488-2 First edition 2004-05

Kohn S., Kumfert G., Painter J. y Ribbens C. Divorcing Language Dependencies from a Scientific Software Library. Lawrence Livermore National Laboratory.

Lane, P. R., & Gobet, F. (2012). A theory-driven testing methodology for

developing scientific software. Journal Of Experimental & Theoretical Artificial Intelligence, 24(4), 421-456.

Oliphant, Travis. (2013). Python for Scientific Computing Computing in Science and Engineering.

Pedreiras, P., Almeida, L., & Gai, P. (2002) The FTT-Ethernet protocol: Merging flexibility, timeliness and efficiency. In Proceedings of the 14th Euromicro Conference on Real-Time Systems (p. 152). IEEE Computer Society.

Reference Manual, Model 4200-SCS Semiconductor Characterization System. 4200-901 Rev N. Sept 2011

Rice J., Boisvert R. From Scientific Software Libraries to Problem Solving Enviromments. IEEE Computational Science and Engineering

Sanner, M.F. Python: A Programming Language for software Integration and Development. The Scripps Research Institute.

Senyk, Thomas. QPA: The Qt Platform Abstraction. 2011 <http://qt-project.org/wiki/Qt-Platform-Abstraction>

Stroustrup, Bjarne. The Design and Evolution of C++. Pearson Education. 1994.

The Python Software Foundation. Glossary. <http://docs.python.org/glossary.html#bytecode>

Tratt, Laurence. Dynamically Type Languages. Advances in Computers. Vol 77. pag 149 – 184

Understanding VNA Calibration. Anritus. http://anlage.umd.edu/Anritsu_understanding-vna-calibration.pdf

Unittest – Unit Testing Framework. <docs.python.org/2/library/unittest.html>

User Manual for E5071C ENA Network Analyzer (Operation and Programming).

Using Qt Designer. <http://pyqt.sourceforge.net/Docs/PyQt4/designer.html>

What is a Source-Measure Unit?. Keithley Whitepapers. http://www.ni.com/white-paper/6850/en/. 2010

What is PyQT. Riverbank Computing, Limited. <http://www.riverbankcomputing.co.uk/software/pyqt/intro>

Why does Qt Use Moc for Signals and Slots. Qt Project Documentation. <http://qt-project.org/doc/qt-5.0/qtdoc/why-moc.html>

Wolfram S, (2002). *A New Kind of Science*. 2nd ed. United States: Wolfram Media.

Anexo 1 – Manual de uso de la librería para el K4200-SCS

Procedimiento General

El patrón básico de utilización de la librería es a través el siguiente procedimiento:

1. Crear objetos SMU que representen las mediciones que se van a realizar en una unidad SMU
2. Crear un objeto K4200 utilizando la dirección IP del K4200-SCS físico y el puerto en donde se esta ejecutando el servidor de KXCI
3. Utilizar el método `attach` del objeto K4200 para añadir los objetos SMU al objeto K4200
4. Ejecutar el método `measure` para ejecutar la medición-fuente
5. Leer los datos utilizando `get_data`

Ejemplo 1:

Para realizar un barrido de corriente en el canal 1 del K4200-SCS en la dirección IP 192.168.0.4 y en el puerto 2205, empezando en 0.01A y terminando en 0.05A y pasos de 0.001A con un compliance de 2V hacemos lo siguiente:

Importamos las librerías necesarias

```
from SMUSweep import SMUSweep

from K4200 import K4200

from SourceMode import SourceMode

from SourceType import SourceType
```

Crear el objeto SMUSweep:

```
sweep_smu = SMUSweep(1, SourceMode.CURRENT,
SourceType.CURRENT, 0.01, 0.05, 0.001, 2)
```

Crear el objeto K4200:

```
device = K4200("192.168.0.4", 2205)
```

Añadir el objeto SMUSweep al objeto K4200:

```
device.attach(sweep_smu)
```

Realizar la medición:

```
device.measure()
```

La llamada `measure()` bloquea hasta que la medición haya terminado. Es recomendable que las llamadas a este método se hagan desde un hilo de ejecución diferente al de la interfaz gráfica.

Recuperar los datos producidos por la medición:

```
data = device.get_data()
```

La llamada anterior devuelve una cadena de texto con todos los datos, tal cual emitió el equipo. Si se llama al método `get_data` antes de realizar una medición se levanta la excepción `NotMeasureYetError`.

Debido a que no se pasaron parámetros con los nombres de la corriente y del voltaje al constructor de `SMUSweep` la clase generará de forma automática nombres únicos para cada voltaje y para cada corriente. Esta funcionalidad es común para todos los objetos *SMU*

Ejemplo 2

Para realizar la misma medición del Ejemplo 1, pero añadiendo una medición por pasos al mismo tiempo, solo es cuestión de añadir un objeto SMUStep al objeto K4200 (llamado *device* en este ejemplo).

Iniciando un objeto SMUStep para barrido de voltaje por pasos:

```
step_smu = SMUStep(2, SourceMode.VOLTAGE,
SourceType.VOLTAGE, 0, 3, 10, 0.001)
```

Añadiendo el nuevo SMU al objeto K4200:

```
device.attach(step_smu)

device.attach(sweep_smu)
```

Una vez que se ejecute `device.measure()` se realizará un barrido de corriente cada vez que se realice un paso en el barrido de voltaje por pasos.

Ejemplo 3

El programa en Python que realiza esta medición sería entonces:

```
from SMUSweep import SMUSweep

from K4200 import K4200

from SourceMode import SourceMode

from SourceType import SourceType

sweep_smu = SMUSweep(1, SourceMode.CURRENT,
SourceType.CURRENT, 0.01, 0.05, 0.001, 2)

device = K4200("192.168.0.4", 2205)

device.attach(step_smu)
```

```
device.attach(sweep_smu)

device.measure() # Bloquea hasta que termine la medición

data = device.get_data()
```

Anexo 2 – Manual de uso de la librería para el E5071c

Ejemplo 1

El uso de la librería para el E5071c es más sencillo. Primero se crea un objeto `VnaChannel`, que luego se podrá usar para interactuar con el analizador de redes. Se asume que el analizador de redes esta en el IP 192.168.0.2 y el servidor que recibe comandos SCPI escucha en el puerto 5025. Se realizará una medición con auto-escalamiento de 500MHz a 1000MHz y se producirán datos en formato logarítmico. Todo esto para el parámetro S11. El procedimiento es el siguiente:

Crear el objeto que representa al E5071c:

```
channel = VnaChannel("192.168.0.2", 5025)
```

Se selecciona el parámetro:

```
channel.set_sparam(1, SParameters.S11)
```

Se seleccionan las frecuencias:

```
channel.set_start_stop(500E6, 1000E6)
```

Se selecciona formato de datos logarítmico

```
channel.set_format(DataFormat.LIN)
```

Se habilita el auto-escalamiento:

```
channel.set_auto_scale()
```

Se leen los datos

```
channel.get_data()
```

De la misma forma que en la librería del caracterizador de

semiconductores, el comando que lee los datos bloquea hasta que los datos se han acabado de leer. Por eso, es recomendable llamar a este método desde un hilo de ejecución independiente. El programa completo es entonces:

```
from lib.util.VnaEnums import SweepType

from lib.util.VnaEnums import SParameters

from lib.util.VnaEnums import DataFormat

from VnaChannel import VnaChannel


channel = VnaChannel("192.168.0.2", 5025)

channel.set_sparam(1, Sparameters.S11)

channel.set_start_stop(500E6, 1000E6)

channel.set_format(DataFormat.LIN)

channel.set_auto_scale()

channel.get_data() # Bloquea hasta leer todos los datos
```

Anexo 3 – Conversión de parámetros S a parámetros Z y parámetros Y

La conversión entre las matrices de dispersión, impedancia y admitancia se realiza a través de simples relaciones matemáticas. Específicamente, si S es la matriz de parámetros S la matriz de parámetros Z (parámetros de impedancia) esta dada por:

$$Z = \sqrt{z}(1_N + S)(1_N - S)^{-1}\sqrt{z}$$

Donde $\sqrt{z}$ es una matriz cuadrada que incluye la raíz cuadrada de la impedancia característica de cada puerto en su diagonal principal.

En la ecuación de arriba 1_N es la matriz identidad de NxN, donde N es la cantidad de puertos de la red.

En términos de la matriz de parámetros Z, la matriz de parámetros Y está dada por:

$$Y = Z^{-1}$$

Es decir, Y es la inversa multiplicativa de la Z.