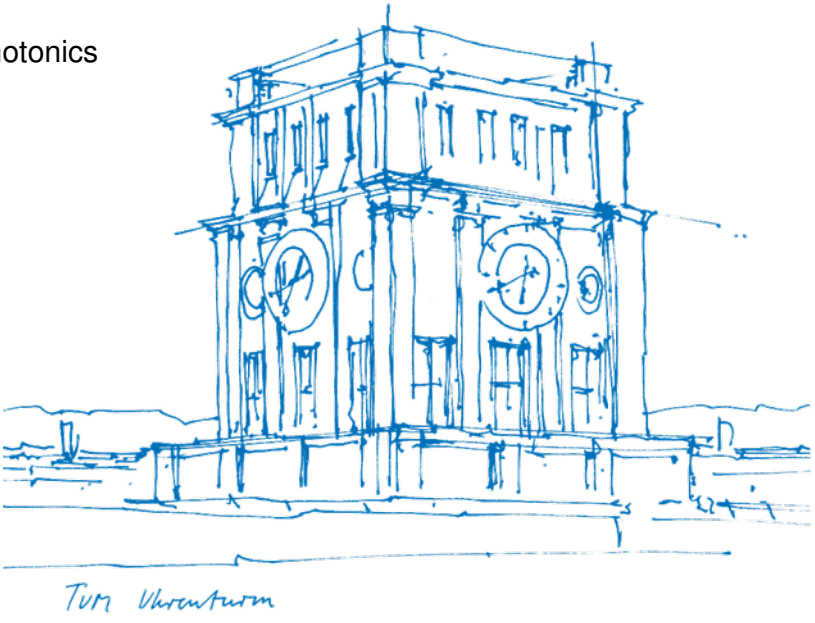
Python User Interface for the mbsolve Project

Mariem Kthiri

Report for Engineer Practice
At the Faculty of Electrical Engineering and Information Technology at the Technical University of München.

Supervised by
Michael Riesch, B.Sc.
Professorship of Computational Photonics

Submitted on
München, 02.05.2017



Contents

1 Introduction

Quantum cascade laser is a type of semiconductor emitting mid-infrared portion of electromagnetic spectrum. This frequency range allows a multitude of applications especially in the field of spectroscopy where it serves the detection of toxic chemicals, explosives, and drugs. In this context, the mbsolve Project provides the required simulation relying on C++ programming with a Python user interface. This Engineer's practice aims at the design of this interface.
Chapter 2 provides the theoretical background and starting point. In chapter 3, the conception and realisation of the project will be described and detailed. The outcome is presented in chapter 4. The discussion of the results concludes this work.

2 Theoretical Background

In this project,a high performing programming language was required for the simulation because of the large size of data and the elaborate numerical treatments. C++ fullfills these criteria not to mention that it is available everywhere and reasonably well standardized. Nevertheless it has some drawbacks: First of all it is not interactive and implementing it for user-interfaces can be quite complex(see Tab. 2.1).
That is why it was opted for Python for the interface which brings flexibilty interactivity and simplicity. To sum it up, the goal is to create a common interface that can be provided with different C++ libraries, these will be wrapped in Python modules and loaded dynamically during the execution of the program.

C++	**Python**
Advantages	**Advantages**
High performance and speed	Flexibility (fast edit-build-debug cycle)
useful for intensive tasks	Interactivity
Parallelization techniques	(create, change, view objects at runtime)
Drawbacks	**Drawbacks**
Non-interactive	relatively slow
Writing user-interfaces	Limitations with memory intensive tasks
is complex	Limitations with database access

Table 2.1 Characterictics of C++ and Python.

3 Conception and Realisation

3.1 Literature and basics

At the beginning, it was necessary to acquire some basics about quantum mechanics (density matrix, Maxwell-Bloch equations..) by reading some introductory literature [1]. A concrete example for the simulation and the numerical solving of the Maxwell-Bloch system was described in the paper of Ziolkowski.
QuTiP, a Python Toolbox for simulating the dynamics of open quantum systems, was then the starting point. It offered a variety of examples using Jupyter Notebooks: an interactive tool designed to expose code blocks with human-friendly text which makes data analysis easier to share and reproduce, used in this case to describe quantum mechanic systems, such as Single-Atom Lasing, Quantum Monte Carlo Trajectories...

3.2 Python/C++ Interface

Creating the Python/C++ Interface was possible through different means:

- Boost.Python: Boost.Python is an open source C++ library which provides an interface for binding C++ classes and functions to Python. It does enable a safe wrapping of C++ functions and support many of its features. But it is bound to GCC which leads to a huge dependance and it uses extensively C++ template causing possible compilation problems and the use of a large amount of memory.

- ctpyes: ctypes is a foreign library for Python, that allows calling DLLs or shared libraries and wrapping them in pure Python. It is a convenient way to reach for a few functions within a DLL, but not to make large C++ libraries available to Python in performance-critical situations.

- SWIG (Simplified Wrapper and Interface Generator): It is a language neutral compiler that turns ANSI C/C++ declarations into scripting language interfaces. It targets Python, Tcl, Perl, MATLAB, etc.. One of its important features is that it is simple and completely automated (generates a fully working Python extension module). In fact, it creates two different files; a C/C++ source file (module_name)_wrap.c or (module_name)_wrap.cxx and a Python source file (module_name).py. The generated C/C++ source file contains the low-level wrappers that need to be compiled and linked with the rest of the C/C++ library to create an extension module. The Python source file contains high-level support code. This is the file that will be imported(see Fig.3.1).

After installing SWIG, a Python module was created starting from an interface file and a header file written in C++. The next step was to wrap a C++ class in order to create instances of this class and modify its attributes in the Python Testprogram. Then the complete C++

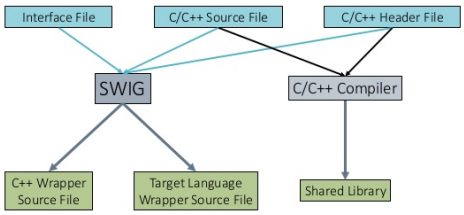


Figure 3.1 Functionality of SWIG

library was gradually elaborated, adding each time required attributes and declarations.

3.3 Serialisation of input parameters

The next point to handle was the input parameters, how to extract them from an XML File and then store them with results at the end. The first option was to use exml or lxml libraries, which could parse XML files but the resulting objects having NoType made handling them problematic. Therefore, a simple object-XML mapper for Python called dexml was applied to serialize the Metadata by defining subclasses of the class dexml.Model and saving the parameters as instances.

3.4 Storing results

Furthermore, it was necessary to specify the file format for the simulation results, which lead to the following options:

- XML: XML's biggest advantage is that it provides developers with a tool that concisely and unambiguously defines the format of data records. However it is not suitable for large amount of data.

- VTK: It is a powerful tool to viisualize scalars, vectors, complex numbers.. but requires a diffcult setup and an understanding of the framework.

- HDF5: HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data. HDF5 is portable and extensible, allowing applications to evolve in their use of HDF5. The HDF5 Technology suite includes tools and applications for managing, manipulating, viewing, and analyzing data in the HDF5 format.

All that considered, HDF5 is the most suitable file format not only to store large amount of data (results) but also different types simultaneously (Metadata).

3.5 Git

After settling the C++ library that contains all the necessary classes and functions, using SWIG to incorporate it in Python, choosing dexml for serialisation of metadata and HDF5 for storage of results, a Git project was created to gather all the pieces, register every step of the realisation and facilitate data exchange and work coordination between me and my supervisor.

3.6 Makefile/CMake

To get more experience in handling scientific projects, it was extremely efficient to understand the process of Makefiles and then CMake and apply them on my work.

- Makefiles are a simple way to organize code compilation, through rules specified as as a list of target entries, in order to build executable programs from many modules. It only rebuilds in case of new modules added to the program.
- CMake is a cross platform build system, that automatically generates Makefiles using the CMakeListst.txt file, that specifies the packages, libraries, source files,.. needed to build.

4 Results

4.1 Python/C++ Interface

The user interface consists of:

4.1.1 C++ Library

- C++ Class Record: specifies which results should be stored and in which interval.
- C++ Class Material: contains the characteristics of each material (name, electric permittivity, magnetic permeability).
- C++ Class Region: includes the specifications of each region (name, start, end, material index).
- C++ Class Device: lists the materials and regions applied in the simulation.
- C++ Class Scenario: contains the total time of the simulation, the timestep and the list of the records.
- C++ Class Result: gives a vector of values corresponding to the measured result (electric field, elements of the density matrix)
- C++ Function: Depending on the Device and Scenario, calculates the results.

4.1.2 Interface

SWIG can be incorporated in the build system CMake, which can detect the SWIG executable and many of the target language libraries for linking against and knows how to build shared libraries and loadable modules on many different operating systems. Using a single cross platform file (CMakeLists.txt) and two simple commands: cmake and make, CMake generates native build files such as makefiles, nmake files and Visual Studio projects which will invoke SWIG and compile the generated C++ files into _(module_name).so (UNIX) or _(module_name).pyd (Windows).

4.1.3 Python Testprogram

The testprogram (Skript and Notebook) allows the setup of Materials, Device, Scenario etc., extracts Metadata from an XML File, calls the C++ Function that calculates the required results, displays them and stores them in addition to the Metadata in a HDF5 File.

4.2 Compatibility with Python3

SWIG is compatible with the different versions of Python (Python 2.7 and Python 3.x). To choose a specific version, all that has to be done is to indicate it in the CMakeLists.txt File, which will find the corresponding packages and use them to compile and link.

4.3 SWIG on Windows

Another approach is available on Windows: building the extension module using a configuration file (conventionally called setup.py), it creates an Extension module object using the source code files generated by swig, in addition to the original C++ source and compiles it into a shared object file or DLL (.pyd on Windows), which can be called in Python testprogram.

5 Conclusion

The Python/C++ Interface is estimated to be the most suitable way to combine the efficiency and performance of C++ programming with the flexibility and interactivity of Python. SWIG was implemented to create the wrappers needed to access the C++ Library from Python and so making the interfacing simple and automatic. Nevertheless, there are other possibilities to achieve this goal such as Cython which is a Python like language for writing C/C++ extensions. In addition, further opmisations can be seeked on Windows using Visual Studio.

Bibliography

[1] C. L. Tang, *Fundamentals of Quantum Mechanics: For Solid State Electronics and Optics* (Cambridge University Press, 2009).