Software Design Manual

Department of Applied Mathematics
School of Aeronautical and Space Engineering
Technical University of Madrid (UPM)

Part I

User Manual

Overview

This software, programmed in Python, has a clear objective of providing a new point of view of the source code of Fortran or Python programs. This is aimed to multilayer-based FORTRAN 95 or PYTHON 3.x programs- This means that there are several modules in each layer which call to modules in the lower layers, and which are called from modules in the upper layer. It can handle large amounts of files and directories, so this is suitable for complex projects, like simulations. In python programs it also includes the files of the directory where it is installed.

It is intended to present different graphs with the main information of the structure of the program. For this is necessary for the program to follow some specifications:

For FORTRAN programs

- Every program and module should be in a separated file, whose name should be the same as the module/program name.
- If inside a file there is code after an *endmodule* or an *endprogram* line, it will not be read.
- This program distinguishes between upper case and lower case, so if there is a call to a module or function, both call and subroutine name should be with identical cases.
- Because the same reason as before, all statements and commands proper of FORTRAN should be in lower case.
- All files should avoid the BOM Unicode writing, as it could cause reading problems.

For PYTHON programs

- It only works with any 3.x versions.
- All the used packages should be installed.

Getting started

Firstly, you will need the software involved in the program. This consist of a folder with following items:

- **"bin" folder** Includes all the program files and source code.
- **"doc" folder** Where this document should be included.
- **"graphs" folder** Where the resulting graphs will be written. This folder is necessary for the program to run, even if it's empty.
- **"excludes.ini"** It is a file which holds the "USES" to be excluded from the graphs. It is read and updated by the program. If you open it with the notebook you can read the last session exclusions (as a python list: [*"item*1*"*,*"item*2*"*,])
- **"configuration.ini" file** Here, data referred to the previous session is stored. This data is the last directory and main file selected.
- **"graphviz-2.28.0.msi"** It is compulsory to install this software, since it creates the graphs from DOT language. It only takes a few steps.
- **"RUN.bat"** The BAT file that starts the program.
- **Python Portable** For the use of this version, it is necessary to have the "WinPython" folder, which contains the python portable, in the root folder of the program, the folder that contains the program folder.

For a successful use of the program you must start with the following steps:

1. Before start FortranProgram you should install GraphViz by double-clicking on the "GraphViz.msi" in the folder you have installed FortranProgram. Then follow the instructions.
2. Once you have installed GraphViz, you can start FortranProgram by double-clicking on the **"RUN.bat"** file of the main directory where you have installed it, and after a few moments the interface will pop up.

The Interface

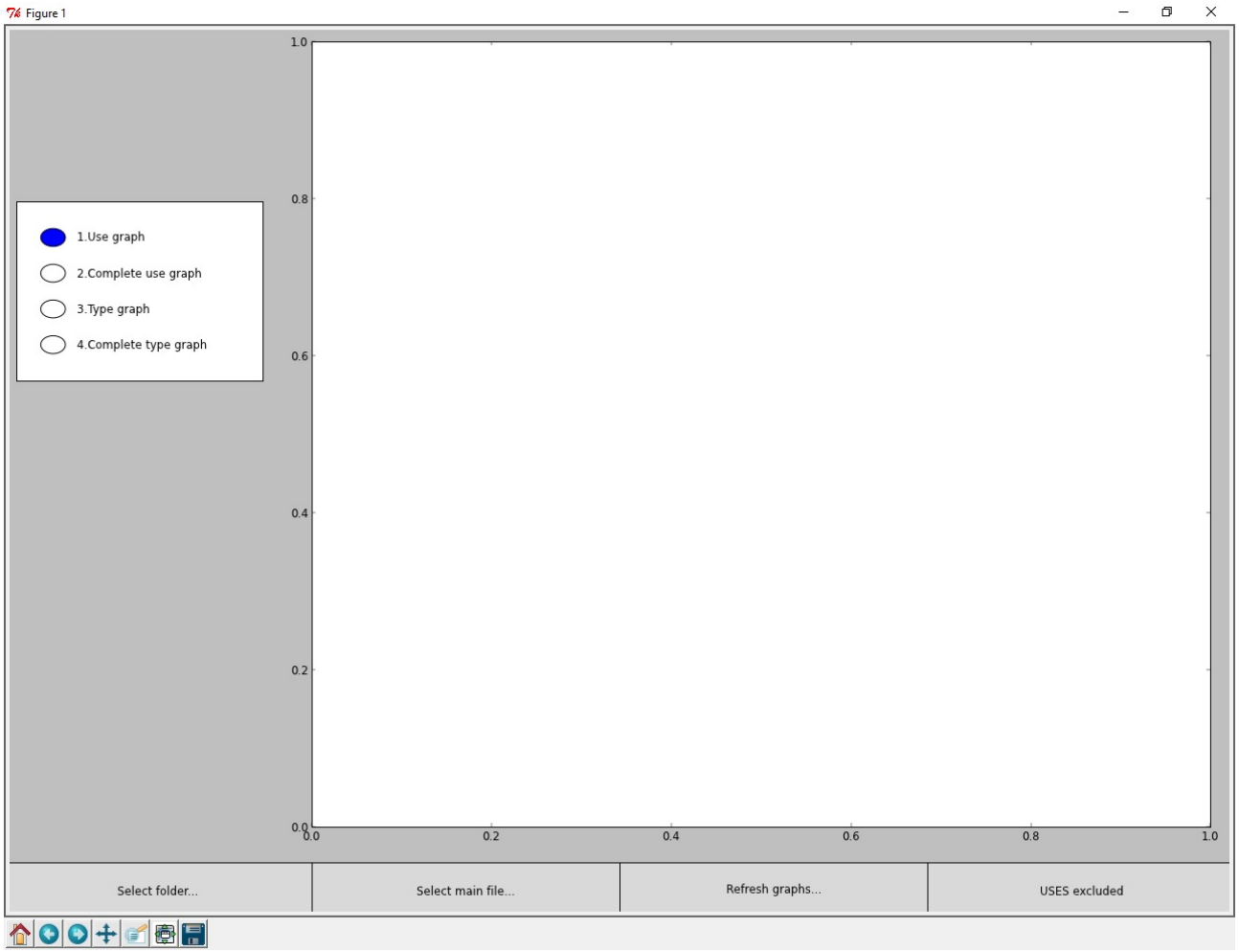


Figure 1: The interface

The interface has a very simple and intuitive use.

First of all, there are four buttons in the lower area.

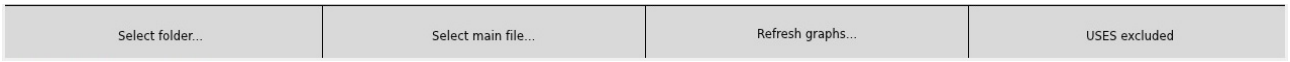


Figure 2: Lower buttons

These are used to select the input directory (**Select folder...** button), to select the main file where it will start reading (**Select main file...** button), to start reading the files (**Refresh graphs...** button) and modify the *exclusion list* (**USES excluded**):

- **Select folder...** From this button you can select the folder that contains all the *.f90* or the *.py* files of source code. In python programs it is not necessary to include the packages installed in the root of python.

- **Select main file...** From here you can select the main file of the program which calls to other modules (which are in other files, but contained in the selected folder). You should select the type of file (.f90 or .py) before looking for it in your PC.
- **Refresh graphs...** Refresh the graphs.
- **USES excluded** In FORTRAN programs: This button opens a window where you can modify the USES (nodes) of the program to be omitted, so these will not appear in the graphs. In PYTHON programs: it opens the same file but in this case there are folders instead of files.f90, which content will be omitted.

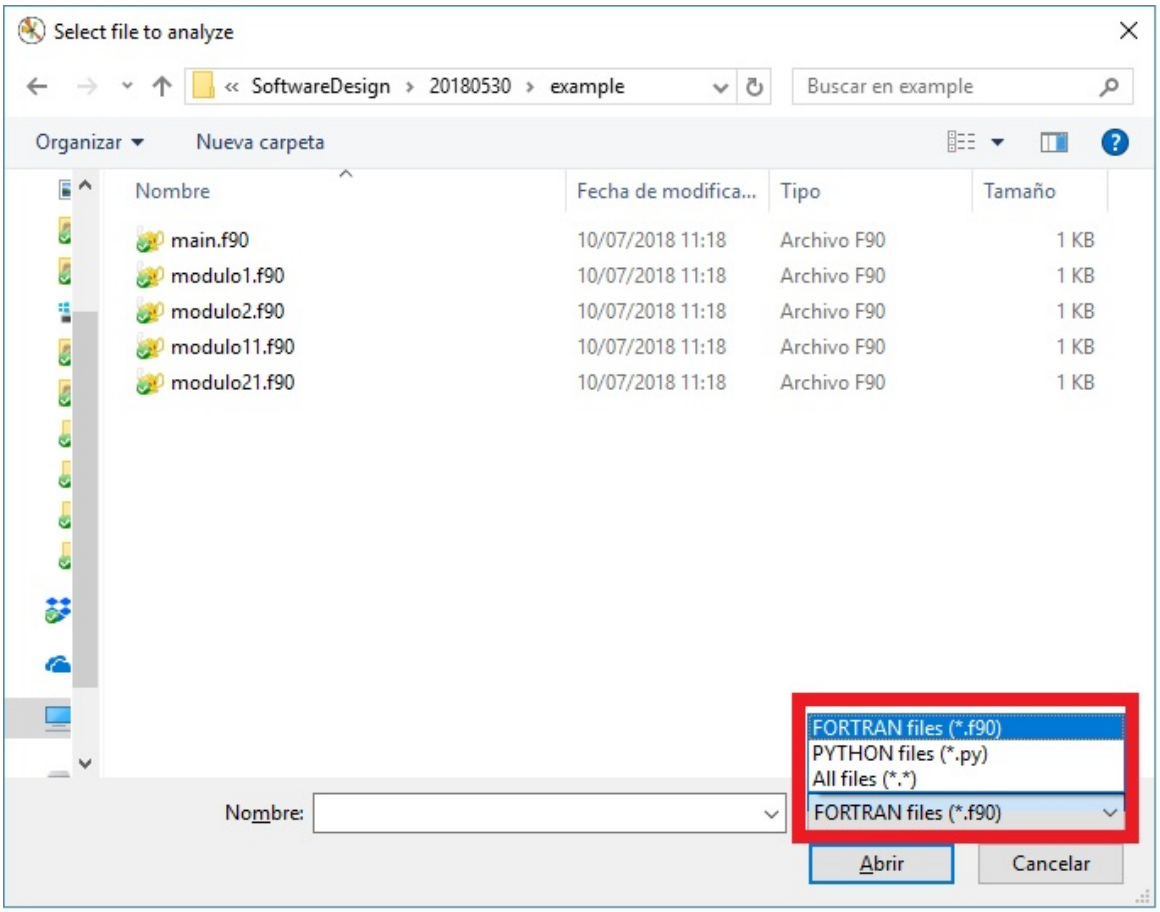


Figure 3: Window that pop ups when you are going to select the main file. At the right lower corner you can select whether to see FORTRAN files or PYTHON files.

Note that: The data referred to the selected directory and main file is stored in a file called *configuracion.ini*. So, if you run the program after a previous session, *configuracion.ini* will be read, and you will find that the directory and main file of the previous session is selected. Then, if you wanted to analyze the same directory and folder, you could click on "**Refresh Graphs...**" directly without selecting directory and folder again.

Then, at the upper left corner we have a list of output graphs. After we first refresh the graphs, the graphs will be available. We can choose between a use's graph, a complete use's graph, a type graph, and a complete type graph:

- **Use's graph**
- **Complete use's graph**
- **Type graph** (Only available for fortran programs)
- **Complete type graph** (Only available for fortran programs)

These will be explained later

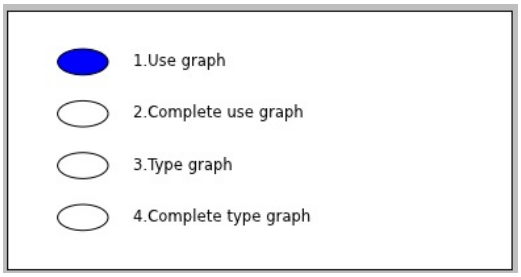


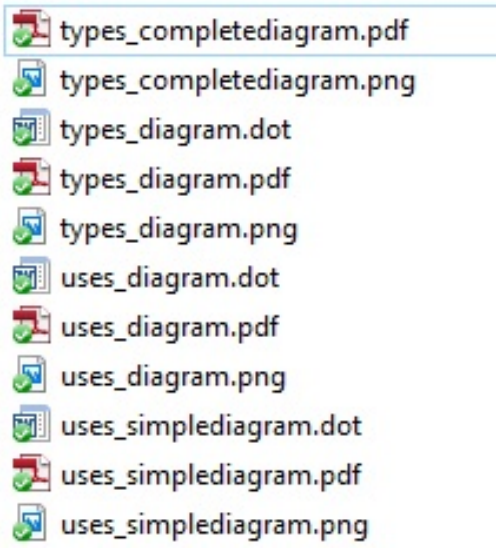
Figure 4: List of available graphs. Only the two first are works with python graphs.

The main area, which is in blank at the beginning, is where the graphs are showed. After reading the files, the first of the graphs will be presented here automatically. If there is any type of error in any graph, the graphs will not show up directly, and only by pressing the buttons of the list will pop up the ones successfully made.

Finally in the lower left corner there are some buttons related to the graph handle, which are not relevant for the use of the SoftwareDesign program, as the results are stored in the **"graphs" folder**. This buttons could be used to zoom or move the graphs in the display area, or to save the selected graph in a specific format.

Graphs

As we said, the result are four graphs for FORTRAN programs and two graphs for PYTHON programs, which you can see with the buttons of the left in the white area. Apart from that, the graphs are saved in the folder *graphs*, in the same directory as the SoftwareDesign program.



Each graph is saved in three different formats: JPG, PDF and DOT. DOT is the lenguage which is used by the GraphViz software, a sub-program that creates the graphs.

- **Use graph:** Files are arranged with main file in the upper layer, followed by the modules in the second layer, which are used with the *use* statement by the main file. Down them are the modules used by the modules from the second layer.
- **Complete use graph:** This one is the same as the previous one, but with extra information within the labels. This includes functions, subroutines and types held in each file.
- **Type graph:** Types are arranged as some of them are extended from others located in the same or in other file, or because inside of some of them there are other types as a part of the main. Labels of types are inside square labels that indicates where are they contained. Only for FORTRAN programs.
- **Complete type graph:** This one is the same as the previous one, but is included in the label the part of the code that defines each type. Only for FORTRAN programs.

If reached some point it is found that there is a use to a module that isn't found, it will be represented as an elliptic label, instead of a box with rounded corners.

Examples

This is the result of the analysing the folder ”Example”, selecting ”main.f90” as the main FORTRAN file.

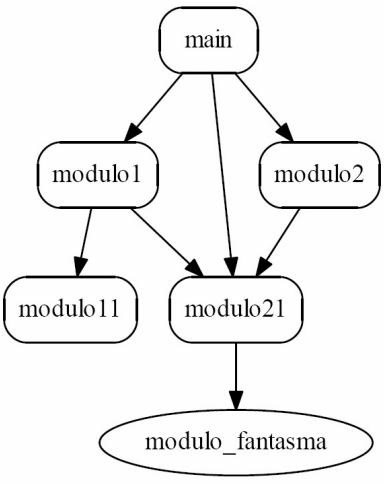


Figure 5: Use Graph

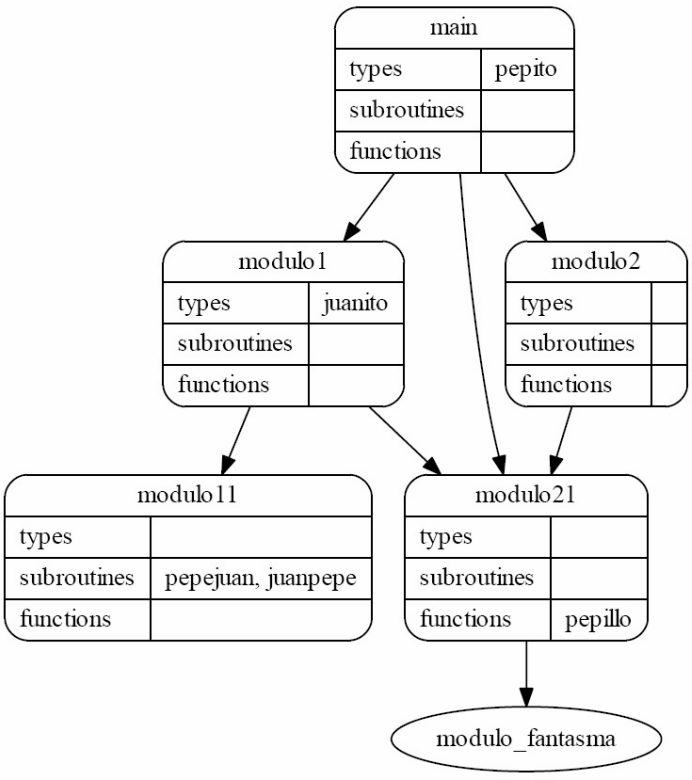


Figure 6: Complete Use Graph

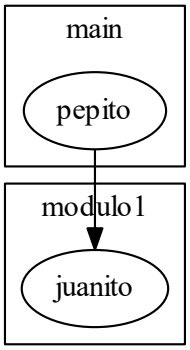


Figure 7: Type Graph

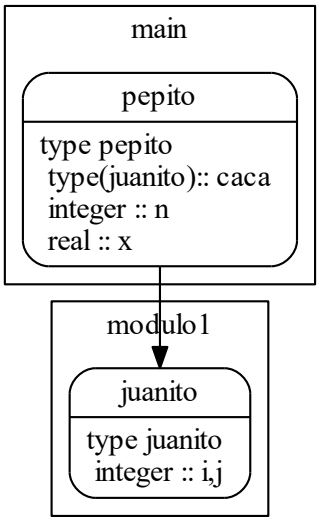


Figure 8: Complete Type Graph

This is the result of the analysing the folder ”Example”, selecting ”main.py” as the main PYTHON file.

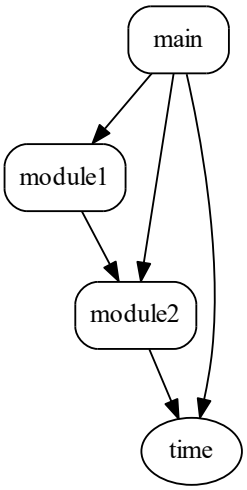


Figure 9: Complete Type Graph

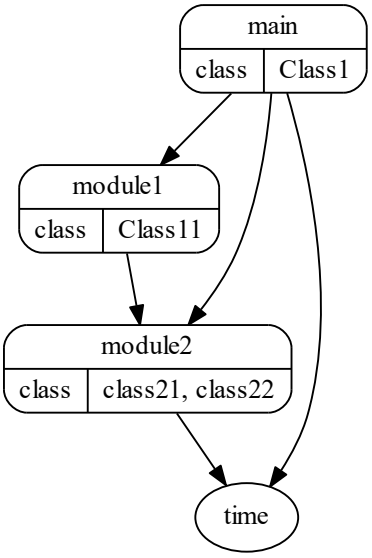


Figure 10: Complete Type Graph

Part II

Developer Guidelines

Overview

This program is writen in python 3.2, and it is written as a project in Visual Studio 2017. In order to execute it from Visual Studio, you need to have installed the following packages:

- **matplolib**
- **PILLOW**
- **PyDot**

The project consists in three files: One common file that defines the interface (*main.py*), one that handles fortran code (*fortran.py*) and the last one that handles python code (*python.py*). The *fortran.py* code and the *python.py* code use the library PyDot for managing graphs.

PyDot Library

This is a library downloaded from GitHub, which author is Ero Carrera. It is a python interface to the GraphViz software, which makes the graphs. More information and installation details are explained in the web page, but in this case you will not need to install it, as it comes with the Software Design source files.

This file, **pydot.py**, offers you a few defined classes related to elements of graphs, with which you can build the graph. These are the more relevant:

- **class Graph** Basic element that contains the rest of the classes.
- **class Dot** It is based on the graph, and represents a code writen in dot language
- **class Subgraph** Class representing a subgraph in Graphviz's dot language.
- **class Cluster** Class representing a cluster in Graphviz's dot language.
- **class Node** It creates a node, and you must add it to a graph, subgraph, cluster or dot class.
- **class Edge** It creates a node between two nodes, and it must be added to the graph, subgraph, cluster or dot class.

Main.py

The *main.py* file uses the **matplotlib** library and its widgets, and it interacts with external files, like the configuration file (which saves the previous session parameters), or the list of exclussions *excludes.ini*. This code also defines the GUI itself.

Both files *fortran.py* and *python.py* have a dynamic of searching in the files key words of the code, and building the structure of the program in dictionaries and tuples.

Fortran.py

This module is responsible of creating an internal structure by reading the FORTRAN code, and later create the graphs with the **PyDot** library.

This module uses mainly the following packages: **re** (Regular Expressions, this is very important because it contains many useful tools for handling text), **os** (for looking for files) and **tkinter** (Windows that asks for specific files of directories).

In *fortran.py*, first searches the USE statements from the main file, passing through all files and building a tuple of the structure. This action is carried out mainly by the function *__get_project_dependencies* and *__get_filedependencies* inside the *fortran.py* code. Later, it searches for the code that defines the types, and it builds again the structure of the types tree. For more details, there are comments in the code itself.

Python.py

This module is responsible of managing the PYTHON code by reading the code files and creating an internal structure. Later, it uses the **PyDot** library for generating the graphs.

After the main file is chosen , the program knows that the file is a python code. The main difference with *fortran.py* is that it searches libraries in the python directory.

As *fortran.py*, it uses packages like **re** and **os**

Example of using the PyDot library

The following is a simple python code that imports the PyDot library. It consists of a graph with three nodes connected, two of which are in a subgraph:

```
from pydot import *

if __name__ == '__main__':


grafico= Dot(graph_name="Prueba",graph_type="digraph")


main_node=Node(name="Nodo1", shape="Mrecord", label="Nodo principal")
sec_node=Node(name="Nodo2", shape="record", label="Nodo secundario")
ter_node=Node(name="Nodo3", shape="",         )

label="{N|P}"
sec_node.set_label(label)

main_node.set_label("{Nodoprincipal|"+label+"}")


grafico.add_node(main_node)
grafico.add_node(sec_node)
grafico.add_node(ter_node)

edge=Edge(main_node,sec_node)
edge2=Edge(main_node,ter_node)

grafico.add_edge(edge)
grafico.add_edge(edge2)


zona=Cluster("zona1",label = "Zona 1")

zona.add_node(main_node)
zona.add_node(sec_node)

grafico.add_subgraph(zona)


grafico.write_pdf('prueba.pdf')
```

Example code that uses the PyDot library

The following is the result of running the previous code (prueba.pdf):