Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Integration of FreeFOAM into Salome

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Ivan Farré Vicente

DIRECTOR: Gerber van der Graaf

DATA: 31 de Octubre de 2010

Títol: Integration of FreeFOAM into Salome

Autor: Ivan Farre Vicente

Director: Gerber van der Graaf

Data: 31 de Octubre de 2010

Resum

El objectiu d'aquest estudi es ampliar i adaptar el software CFD (computació de fluids dinàmics) utilitzat per als estudis avançats al camp dels fluids de la sang dins l'equip RMEE de la UPC/EPSEM. El programari CFD FOSS (aplicacions lliures i de codi obert) FreeFOAM és utilitzat per aquest propòsit. FreeFOAM es una branca del extensament utilitzat OpenFOAM CFD. FreeFOAM esta pensat per permetre'ns utilitzar les eines que proveeix OpenFOAM dins del nostre sistema a través de CMake, habilitant-nos a utilitzar FreeFOAM com una aplicació nativa dins del nostre sistema operatiu. FreeFOAM ens proporciona un conjunt d'aplicacions en C++, anomenades 'solvers' que serveixen per simular i solucionar problemes dins del camp de la mecànica de fluids.

Per als estudis de CFD és comú aspirar a obtenir una interfaç d'usuari gràfica per generar geometries complexes. Per exemple les geometries de les arteries utilitzades per l'estudi dels fluids sanguinis fets a la UPC/EPSEM. Encara que OpenFOAM i FreeFOAM són uns dels softwares més avançats dins del programari CFD, sota llicència GPL, una de les seves principals mancances es la falta de un mòdul que permeti visualitzar els resultats durant el pre-processament i el post-processament. Una manera d'obtenir aquesta tipus de mòdul gràfic dins de OpenFOAM/FreeFOAM és fent que les seves llibreries funcionin com scripts Python.

En aquest estudi en centrarem com obtenir els mòduls Python de FreeFOAM. El software Salome té integrat una consola que ens permet accedir a algunes de les seves funcionalitats utilitzant scripts Python. PyFreeFOAM intenta definit un nou nivell de flexibilitat i d'interacció per als usuaris dins d'aquest escenari sobre programari de simulació CFD. El nostre objectiu és utilitzar aquest projecte per la creació d'una eina que permeti al software FreeFOAM utilitzar les seves funcionalitats dins de Salome.

Title: : Integration of FreeFOAM into Salome

Author: Ivan Farré Vicente

Director: Gerber van der Graaf

Date: October, 31th 2010

Overview

The aim of this study is to extend and adapt the CFD (Computational Fluid Dynamics) software used in advanced studies on hemodynamics at the RMEE team at UPC/EPSEM. The FOSS (Free and Open Source Software) CFD software FreeFOAM (Open Field Operation and Manipulation) is used for this purpose. FreeFOAM is a fork of the widely used OpenFOAM CFD software. It is geared towards freeing OpenFOAM from its system dependency (using Cmake), enabling it to run natively on as many operating systems as possible and improved documentation. The software concerns a a C++ toolbox for the development of customized numerical solvers, and pre-/post-processing utilities for the solution of continuum mechanics problems, including computational fluid dynamics (CFD).

For CFD studies, a Graphic User Interface (GUI) program is often desired to create complex geometries. For example geometries of arteries used for hemodynamics (blood flows) studies, done at UPC/EPSEM. Though OpenFOAM/FreeFOAM is one of the most advanced CFD tools under a the FOSS GPL licence, its main drawback is the lack of a Graphic User Interface for integrated displaying pre-processing and post-processing results. In order to get the functionality of the OpenFOAM / FreeFOAM CFD software in some GUI programs, the CFD libraries will need their functionality in Python scripts. This is obtained by the creation of modules that are derived from the libraries and can be loaded in Python scripts or programs.

In this study we will focus to obtain python modules from the FreeFOAM libraries. The SALOME GUI desktop, embeds a Python console allowing the user to access different SALOME functionalities via Python interfaces. PyFreeFOAM intends to define a new level of flexibility and user interaction scenario for numerical simulation software. Our aim is to use this project for the creation of a python front-end for the FreeFOAM CFD software to get its functionality in Salome.

INDEX

INTRODUCTION

Computational fluid dynamics (CFD) is established to be one of the essential part of the aided engineering (CAE) tools and it has been far used for differents purposes. It approach to model fluid flow phenomena allows engineers and technical analysts to have a powerful fluid simulation on their desktop. Computational fluid dynamics is a branch of fluid mechanics that uses numerical methods and algorithms to solve and analyze problems that involve fluid flows.

Nowadays exist more projects that have been developed that allow us to provide simulations with results extremely near the reality. Nevertheless most of them are from private sector and it implies, in most of cases, to acquire expensive licenses. Open source CFD software gives free angles of computation without pay and has free license under the GNU general public license. The open source community follow the exchange of the information regarding to the numerical research for industrial purposes.

This way software like as OpenFOAM or FreeFOAM provide tools for run CFD simulations in various scenarios. For beginner users could be difficulties to use them if they don't familiarize with complex numerical functions and don't have knowledge about C++ code. Therefore we develop python interface, PyFreeFOAM. He enables work from python interpreter with FreeFOAM toolbox.

In first chapter shows a global perspective in history of CFD. We obtain perspective in history about his evolution and differents areas where we can use this kind of software. In second chapter are explained some features of OpenFOAM and FreeFOAM, centred in his installation process. There we could recognize principal differences in related open source projects. After in third chapter we describe the develop and implementation process made to obtain python interface. We gather the work initiated in the Vulashaka project to grab our objectives.

The memory finishes with the conclusions of the project and some propositions of future improvements. And finally the references to the whole documentation consulted.

CAPÍTOL 1. Introduction to CFD Software

This chapter will deal about history of CFD evolution and CFD software features. We describe some differences that exist between open source and commercial software when applied as engineering tool.

1.1. A bit of hystory

Numerical methods were known since the Newton's time in 1700s. It is debatable, who actually performed the earliest CFD calculations, although Lewis Fry Richardson in England (1881-1953) developed the first numerical weather prediction system by dividing physical space into grid cells and using the finite difference approximations of Bjerknes's "primitive differential equations". In 1910, Richardson published 50 pages paper to Royal Society; he performed hand calculations by using human computers, which were iterated 2000 operations per week. His own attempt to calculate weather for a single eight-hour period took six weeks of real time and ended in failure. Since the 1940s, analytical solutions to most fluid dynamic problems, especially those arising in aerodynamics, were readily available for simplified or idealized conditions. The following list shows a bit perspective of progress in history about CFD study[1]:

- 1943 – Finite element analysis (FEA) was first developed by R. Courant, who utilized the Ritz method of numerical analysis and minimization of variation calculations to obtain approximate solutions to vibration systems.

- Around 1960 - First Scientific American articles on CFD was published.

- 1965 - Marker and Cell methods - Harlow & Welch.

- 1965 - Use in research and "grand challenges" (NASA, Los Alamos...).

- 1970 - Finite difference methods for Navier-Stokes.

- 1970 - Finite element methods for stress analysis.

- 1980 - Finite volume methods (Imperial College). At the moment, the Finite Volume Method (FVM) method is the most common standard method in numerical fluid dynamic simulation.

- 1983 - Fluent was launched as commercial CFD soft. Afterwards a number of commercial CFD software growth and fulfilled the competition of CFD market.

- 1985 - Use in "aero" industries (Boeing, General Electric, ...) .

🕐 1995 - Use in "non-aero" industries (General Motor, Ford, Astra, Ericsson...).

🕐 2004 - An open source CFD software (FOAM) was released.

🕐 2006 - Fluent was acquired by ANSYS, Inc. This acquisition makes ANSYS as a strongest computer aided engineering player in numerical FEA and CFD simulation.

🕐 2011 - OpenFOAM was acquired by SGI, Ltd.

1.2. CFD Software

Computers are used to perform the calculations required to simulate the interaction of liquids and gases with surfaces defined by boundary conditions. With high-speed supercomputers, better solutions can be achieved. Ongoing research, however, yields software that improves the accuracy and speed of complex simulation scenarios such as transonic or turbulent flows. Initial validation of such software is performed using a wind tunnel with the final validation coming in flight tests. In all of these approaches the same basic procedure is followed[2].

- During preprocessing
  - The geometry (physical bounds) of the problem is defined.
  - The volume occupied by the fluid is divided into discrete cells (the mesh). The mesh may be uniform or non uniform.
  - The physical modeling is defined – for example, the equations of motions + enthalpy + radiation + species conservation
  - Boundary conditions are defined. This involves specifying the fluid behaviour and properties at the boundaries of the problem. For transient problems, the initial conditions are also defined.
- The simulation is started and the equations are solved iteratively as a steady-state or transient.
- Finally a postprocessor is used for the analysis and visualization of the resulting solution.

Historically, the closed software business model has been most popular since recently. Following this model, software is sold only in binary format without publishing the source code. Software issued under the Free and Open Source Software (FOSS)[3], among GPL, the source code is free available. In case modifications are made in FOSS software, changes in the source code will have to be published as well in case a developer will issue the modified software. So, other users and developers will profit from the changes and may continue to

improve and extend the software. In this way a collaboration of developers may grow for a specific piece of software, all round the world.

Other 'free' software exist that allow to access the source code, but do not require to issue the changes that have been made. These licences are often appreciated by private companies who prefer the closed source software model for their business. An example is the BSD license used in the BSD unix by Apple for theis OSX operating system.

Most of the professional available CFD software is available under closed source licences. Often, very high prices are asked for these programs and it is impossible to understand its working. With this type of software it is impossible to have full access to the code and to modify it to one's needs.

CAPÍTOL 2. FOAM Software

In this chapter we talk about two open source projects developed in CFD area. OpenFOAM, the most used program for open source CFD community, and his fork FreeFOAM. Both has developed to run in Unix/Linux distributions but they have some differences in his installation process.

2.1. OpenFOAM

FOAM was written at Imperial College, London, to develop a more powerful and flexible general simulation platform than the standard at its time, FORTRAN. For a few years FOAM was sold as a commercial code by their company Nabla. However, in 2004 they decided to release the code under GPL and rename it to OpenFOAM[4]. OpenFOAM is written in C++ and uses an object oriented approach which makes it easy to extend.

C++ provide the mechanism to declare types and associated operations that are part of the verbal and mathematical languages used in science and engineering. OpenFOAM classes, have a syntax that closely resemble the partial differential equations being solved. For example the equation:

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \bullet \phi \mathbf{U} - \nabla \bullet \mu \nabla \mathbf{U} = -\nabla p$$

Fig. 2.1.1 Equation

is represented by the code:

```
solve
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - fvm::laplacian(mu, U)
    ==
  - fvc::grad(p)
);
```

OpenFOAM is a C++ toolbox used primarily to create libraries and executables or applications. Two categories of applications exist:

- solvers, that are each designed to solve a specific problem in continuum mechanics.
- utilities, that are designed to perform tasks that involve data manipulation.

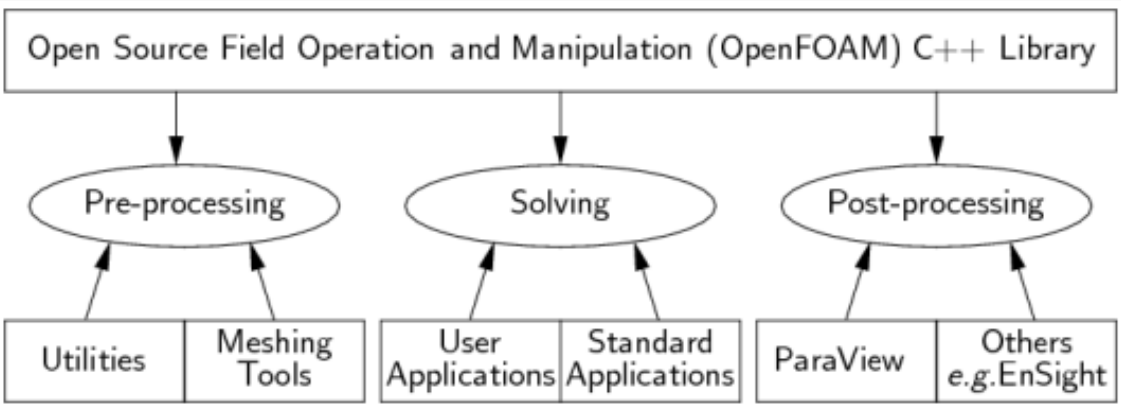

Fig. 2.1.2 OpenFOAM structure

The essence of object-orientation is that the user does not need to have a detailed knowledge of the working of the code; only knowledge of the class existence and its functionality are sufficient for its usage . This, and other requirements, demand that the principal programming language of OpenFOAM uses object-oriented features such as inheritance, template classes, virtual functions and operator overloading. These features are not available in many languages that sense to be object-orientated but actually have very limited object-orientated capability, such as FORTRAN. C++ has got all these features while it also is s widely used with a standard specifications by reliable compilers to produce efficient executables. OpenFOAM uses Wmake to perform the compilation (building, linking) of its C++ source code. Wmake uses similar concepts as other make tools, such as GNU make.

The source code of OpenFOAM is organised the in such a way that each application or library is placed in a separate directory of which its name is that of the application or library. The top level source file takes the application name with the .C extension. For example, the source code for an application called newApp would reside is a directory newApp and the top level file would be newApp.C as shown in figure.

Fig. 2.1.3 OpenFOAM application hierarchy

In case of an application, the Make/options file contains the full directory paths to locate the header files of the required libraries as well as a list of .C source files to be compiled. The list also contains the main .C file and any other source file required to create the specific application which are not included in a class library. For example, users may create a new class or some new functionality to an existing class for a particular application. The full list of .C source files, then, must be included in the file called Make/files .

Apart from the source code that can be compiled by the user, OpenCFD, the provider of OpenFOAM, also offers the software in binary format for the amd64 and i386 architectures. However, these binaries are limited to the Suse and Ubuntu Linux distributions.

2.2. FreeFOAM

FreeFOAM[5] is initiated because of the difficulties and limitations of OpenFOAM during compilation, installation and usage of the software. FreeFoam is a fork of OpenFOAM and is intended to make the software more portable and user-friendly.

In OpenFOAM some steps are needed before compilation and installation. The user should choose a directory location and set some environment variables manually. Some scripts will have to be executed for checking the system and to find required software for building and usage of OpenFOAM. After these steps have been passed successfully, the user may execute the wmake scripts in order to compile OpenFOAM. When finished, the wmake process will have to be invoked again for testing and installation. When all processes have been finished correctly, the user is able to use OpenFOAM. For using OpenFOAM the applications and libraries, specific environment variables will have to be defined in order to locate the libraries and programs. Although these environment variables can be defined automatically using a profile, it is uncommon as a situation may become easily unmanageable on a system using thousands of programs. Therefore, environment variables are eliminated in FreeFOAM. Also an automatic configuration, compilation and installation system is required to eliminate manual intervention. Various tools exist to perform the tasks for configuration, compilation building the software more easily and efficient than the Wmake scripts used in OpenFOAM, like Scons, Jam and Waf and Cmake. FreeFOAM uses the Cmake[6] software to replace the Wmake[7] system used in OpenFOAM.

Apart from the CMake configuration and building system, the source code of FreeFOAM is mostly kept identical to that of OpenFAOM. FreeFOAM libraries and applications are organized using the convention that the source code of each application is placed in a directory whose name is that of the application with the word Foam appended. For example, the source code for a simple solver called T, solving the heat transport problem, would reside is a directory TFoam and the top level file would be Tfoam.C.

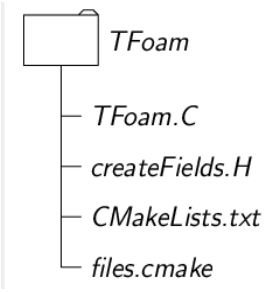

Fig. 2.2.1 FreeFOAM application hierarchy

In addition, the directory contains a CMakeLists.txt file and, optionally, a files.cmake file. When compiling the software using CMake, it is common to do perform the building process in a directory outside of the source tree. This involves that the source tree will stay unaffected. The building process takes place in two stages. First, standard build files are created from configuration files. Then the platform's native build tools are used for the actual building. The build dependencies are found automatically. Configuration settings may be re-defined manually. To start from fresh, a build directory will have to be created and cmake will have to be invoked from within the build directory, pointing to the source code of FreeFOAM, for configuration and compilation by typing for example:

```
$ mkdir build
$ cd build
$ cmake ../$FreeFOAM_HOME
$ make
```

CMake provides a system of messaging that is written during compilation process, most of which are to help debugging. When finished the compilation, “make install” may be invoked to install the FreeFOAM libraries, executables and other required files to the target directories. Finally, the user may run the FreeFOAM applications.

CMake has some additional benefits above the Wmake system:

- Full cross platform.
- Very simple script language that supports control structures (conditional, iterative), regular expressions, eliminating the need of grep+awk+sed+perl, macros (similar to functions) and portable commands for file and directory manipulation.
- Automatic dependency discovery.
- Ability to add custom rules and targets.

So far, an official release package of FreeFOAM has not been issued, yet. However, there are release candidates for the first public release available. FreeFOAM should prove to be more successful, but it's still in development.

CAPÍTOL 3. Python integration of OpenFOAM

In this chapter describe some characteristics of Python language and talk about other open-source project PythonFlu, our base to achieve python modules and libraries of PyFreeFOAM.

Python[8] is one of the most common and popular open source scripting languages. With Python we could split our applications into modules that can be reused in other Python programs. As such, Python has a large library of written code and tutorials. Under its open-source license, it is easy to port to other operating systems, such as Microsoft's Windows, Apple's Mac OS X and all Linux distributions. The Python interpreter is easily extended with new functions and data types implemented in C or C++. Python is also suitable as an extension language for customizable applications.

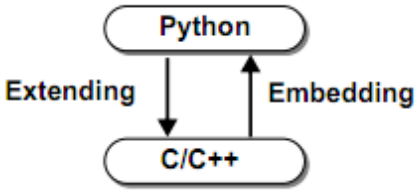

Fig. 3 Python/C++ diagram

Python is a scripting language that has many advantages over other scripting languages. Programs made with this language are generally expected to run slower than C++ programs, but they also take much less time to develop. Python programs are typically 5-10 times shorter than equivalent C++ programs. This difference can be attributed to Python's built-in high-level data types and its dynamic typing. For example, a Python programmer wastes no time declaring the types of arguments or variables. Other advantage is that Python's run time must work harder than C++'s. For example, when evaluating the expression “a+b”, it must first inspect the objects “a” and “b” to find out their type, which is not known at compile time. It then invokes the appropriate addition operation, which may be an overloaded user-defined method. C++, on the other hand, can perform an efficient integer or floating point addition, but requires variable declarations for “a” and” b”, and does not allow overloading of the “+” operator for instances of user-defined classes. For these reasons, Python could be considered better language. While C++ is better characterized as a low-level implementation language. In fact, the two together make an excellent combination. Components can be developed in C++ and combined to form applications in Python. These comparisons concentrate on language issues only. In practice, the choice of a programming language is often dictated by other real world constraints such as cost, availability, training, and prior investment, or even emotional attachment.

In our mind our initial objective was wrap our-self all FreeFOAM source files to obtain his respectively python modules. We spend more time without significant advance trying to made interface files for each one header file. We heard about

other open-source project with similar goals, PythonFlu. We proceed to analyze software and contact with Alexey Petrov, principal developer of this project, to get help and guidance. He give us the directions to understand the effort of pythonFlu. He reference us that they were necessary two developers, himself and other for one year to wrap OpenFOAM. In the next part tell some properties of PythonFlu program.

3.1 PythonFlu

The principal objective of PythonFlu[9] comes from the need to make OpenFOAM calculation environment more interactive and more automated at the same time. Work with OpenFOAM solvers could be hardest because his C++ code definition needs to be compiled before we could run them. And this steep should be repeated for every modifications that we have made.

PythonFlu is the continuation of Vulashaka[10] project. The developers of this project works in Pebble Bed Modular Reactor from South Africa. They developed some applications to work with OpenFOAM libraries under Python modules. Some of his components are:

- Ifoam (Interactive FOAM): Interactive calculation and integration framework for OpenFOAM.
- pyFoam[11]: Python front-end to OpenFOAM, supports IFoam.
- unv2foam: Extends OpenFOAM ideas UnvToFoam utility, introducing embedded capabilities necessary for embedding (available as C++ function).
- foam2vtk: Memory based conversion of FOAM objects into VTK objects for display.
- foam2med: Allows translation of OpenFOAM data into MED format to enable integration with SALOME.
- confFoam: Common configuration package for OpenFOAM development based on automake tools.

At February 2010 they had been wrapped about 40% of the OpenFOAM C++ API into Python modules[10]. The Valushaka project is still an unfinished project. Before this PythonFlu follow Vulashaka purposes, obtain python front-end for OpenFOAM framework. Some of PythonFlu features are:

- Decrease the learning curve by substituting the initial OpenFOAM C++ interface with an easy to learn Python one.
- Keeps the same performance .
- Enables definition of OpenFOAM pure Python classes derived from the corresponding C++ source files. Allows you to define custom "physical models", for example.

PythonFlu allow us to obtain python modules and libraries of OpenFOAM using Swig. Swig[12], Simplified Wrapper and Interface Generator, is a code development tool to provide automatic wrapping of libraries written in C/C++. Python, provide some mechanism for making calls to functions written in C or

C++. However, this almost always requires the user to write special wrapper functions that provide the union between the scripting language and the C functions. This works by taking the declarations found in C/C++ header files and using them to generate the wrapper code that scripting languages need to access the underlying C/C++ code. Swig automates this process by generating wrapper code from a list of ANSI C function and variable declarations. As a result, adding scripting languages to C applications can become an almost trivial and automated work. Swig allows to connect programs written in C and C++ with scripting languages, such as Perl, Python, Ruby, and Tcl.

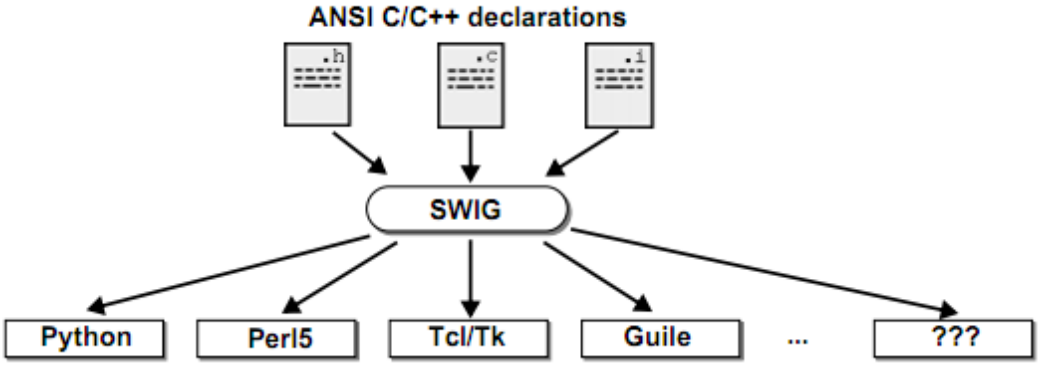

Fig. 3.1 Swig conversions codes

The ease of using Swig makes it possible for scientists to build applications out of components that might not have been considered earlier. Nonetheless Swig have some limitations. Swig it's limited to understand complex data types. The user should write special functions to extract information from structures even though passing complex data types.

CAPÍTOL 4. Development and implementation

In this chapter shows steeps in development and implementation to achieve python front-end for FreeFOAM using Cmake. The result of this work has received the name of PyFreeFOAM[14].

The build procedure of the pythonFlu package is based on GNU autoconf and make utilities. As like as OpenFOAM, the pythonFlu installation process carry some steps of environment configuration previous to compilation. ConfFlu is common package configuration that has been used for pythonFlu. This package provides, with .m4 scripts, the environment variables for compilation process and check if system have third-party packages required to use pythonFlu.

We follow the wheel of FreeFOAM and use CMake to build and compile source files and obtain libraries and python modules. Furthermore CMake allow to use of third-party tools, like as Swig, to compile the application.

4.1 PyFreeFOAM

The motivation of the work presented here is to provide functionality of the FreeFOAM library in the Python language. Our work consist in wrap the FreeFOAM source files using interface files developed in PythonFlu project. However, Pythonflu will need adaptions for wrapping the FreeFOAM libraries. The following steps describe the procedure followed to obtain PyFreeFOAM:

A. Renamed files to allow use them with CMake
B. Edit the interface file and add Swig directives.
C. Write patch files to do wrapping for each version of program.
D. Run Swig, compile and link into a Python extension module.
E. Rewrite the solver application's in python language to run with python module compiled in steep D.

A. *Renamed files to allow use them with CMake*

In pythonFlu the interface files have “.cxx” extension instead of “.i”, as is common usage by Swig. PythonFlu developers prefer to behave their text editor (Emacs) in C++ mode without changing the editor's configuration settings. As this is, to our opinion, a weak argument and is very confusing when researching the code. In addition, the configuration and building software CMake will interpret the interface file wrongly. Therefore, the interface files have been renamed with extension “.i”. We didn't renamed “.cpp” files to “.i” extension although they are interface files too. These interface files are invoked and included by the main interface files and, therefore would not be compiled as standalone.

Also there are files with “.hh” extension. These files are used to overcome the OpenFOAM "standalone header usage" problem. When an OpenFOAM header is taken separately, for swigging or compilation for example, it is not 'self sufficient': it requires to include some additional header files to be completely defined and processed.

B. *Edit the interface file and add Swig directives*

We have made modifications in interface files to compile with FreeFOAM. We have adapted them to the structure of FreeFOAM to provide access to files that are different name or place that in OpenFOAM. One example of this is SymmTensor header file. In OpenFOAM we can find this file under this name “SymmTensor.H” whereas in FreeFOAM this file has named with underscore at final, “SymmTensor_.H”. Our work around was to include these lines in the SymmTensor interface file:

```
#if FOAM_NOT_BRANCH( __FREEFOAM__ )
%include <SymmTensor.H>
#else
%include <SymmTensor_.H>
#endif
```

We repeat this process for all files that have different name of location in FreeFOAM with regard to OpenFOAM. We add “__FREEFOAM__” as one more possible value of “FOAM_NOT_BRANCH” variable too, this variable is needed for PythonFlu to recognize which version of FOAM we are compiling.

C. *Write patch files to do wrapping for each version of program.*

At this point the source code of PythonFlu is adapted to work with FreeFOAM. We need create patch files for the wrapping procedure. This files are located in the “Foam/patches” directory. This patch directory contains changes of files to be patched for each version of OpenFOAM or FreeFOAM. Also patches are found for FreeFOAM in “1.7.1-free” and “1.5-free” sub-directories. These sub-directories contain files that are invoked recursively. Since FreeFOAMrc5 all patches will have to be applied. From older versions of FreeFOAM only patches files in 1.5-free sub-folder are needed. This is because FreeFOAMrc5 corresponds with OpenFOAM1.7.1 version. In older versions of FreeFOAM we could find his corresponds with OpenFOAM in his README or Release Notes files. For this patching we look the method used in patched files of OpenFOAM subfolders. We adapt them to FreeFOAM structure and according as when tried to compile them and we could detect errors in swigging process.

Patches are used to overcome existing Swig syntax analysis limitations. Swig fails when parsing some advanced C++ semantic construction like as "template friend functions". For example in the patch file Field.H we can see the following:

```
#ifndef Swig
friend Ostream& operator<< <Type>
(Ostream&, const Field<Type>&);
friend Ostream& operator<< <Type>
(Ostream&, const tmp<Field<Type> >&);
#endif
```

Another example of can be found in sphericalTensor.H patch file:

```
#ifndef Swig
// Identity tensor
static const sphericalTensor I(1);

static const sphericalTensor oneThirdI(1.0/3.0);
static const sphericalTensor twoThirdsI(2.0/3.0);
#else
// Identity tensor
static const sphericalTensor I;

static const sphericalTensor oneThirdI;
static const sphericalTensor twoThirdsI;
#endif
```

By this patch, we feed to Swig the C++ construction which can be processed properly, as Swig cannot process the original C++ definition. In this way the value "sphericalTensor I" can be used in the Python module. So, we made a trick by letting Swig to wrap this value properly and use the original C++ definition at run-time.

D. *Run Swig, compile and link into a Python extension module.*

We have developed a common CMake structure to PythonFlu, now renamed for ours PyFreeFOAM. In the main directory we include a CMakeList.txt file. This file checks if Python, Swig and FreeFOAM are still properly installed on the computer system and provides the required environment variables in order to compile PyFreeFOAM using the FreeFOAM source files. Each piece of required software (libraries, programs like compiler etc) is checked in a modular way using CMake modules. The following lines the in CmakeLists.txt main file shows since it is easy to verify if Python, Swig and FreeFOAM has installed in the system:

```
#Python enviroment
FIND_PACKAGE(PythonLibs)
INCLUDE_DIRECTORIES(${PYTHON_INCLUDE_DIR})

#Swig enviroment
FIND_PACKAGE(Swig 1.3.40 EXACT REQUIRED)
INCLUDE(${Swig_USE_FILE})

#FreeFOAM enviroment
FIND_PACKAGE(FreeFOAM REQUIRED)
INCLUDE(${FOAM_USE_FILE})
```

If the related software are not installed we will receive a error message. Then we define specific variables used later in pyfoamUtilities.cmake file that we will need to set appropriate environment.

```
SET (PYFOAM_INCLUDE_DIR ${FOAM_INCLUDE_DIRS} CACHE
STRING "Foam Path." FORCE)
SET (PYFOAM_LIBRARY_DIRS ${FOAM_LIBRARY_DIRS} CACHE
STRING "Foam Libs." FORCE)
SET (PYFOAM_BRANCH FREEFOAM CACHE STRING "Choose the
type of Foam OpenSource distribution." FORCE)

SET (PYFOAM_VERSION 010701 CACHE STRING "Choose the
corresponding version of OpenFOAM used. Could be
\"010701\",\"010700\",\"010600\",\"010500\"." FORCE)

SET (PYFOAM_FLAGS -D__FOAM_VERSION__=${PYFOAM_VERSION}
-D__FOAM_BRANCH__=__${PYFOAM_BRANCH}__)
SET (PYFOAM_INCLUDE_DIRS -
I${PYFOAM_INCLUDE_DIR}/finiteVolume
-I${PYFOAM_INCLUDE_DIR}/sampling
-I${PYFOAM_INCLUDE_DIR}/basicThermophysicalModels
-I${PYFOAM_INCLUDE_DIR}/meshTools
-I${PYFOAM_INCLUDE_DIR}/incompressibleLESModels
-I${PYFOAM_INCLUDE_DIR}/incompressibleRASModels
-I${PYFOAM_INCLUDE_DIR}/compressibleLESModels
-I${PYFOAM_INCLUDE_DIR}/compressibleRASModels
-I${PYFOAM_INCLUDE_DIR}/radiation
-I${PYFOAM_INCLUDE_DIR}/incompressibleTransportModels
-I${PYFOAM_INCLUDE_DIR}/interfaceProperties
-I${PYFOAM_INCLUDE_DIR}/dynamicFvMesh
-I${PYFOAM_INCLUDE_DIR}/dynamicMesh
-I${PYFOAM_INCLUDE_DIR}/randomProcesses
-I${PYFOAM_INCLUDE_DIR}/OpenFOAM
-I${PYFOAM_INCLUDE_DIR}/OSspecific)

ADD_DEFINITIONS(${PYFOAM_FLAGS} ${PYFOAM_INCLUDE_DIRS})

foreach(f ${FOAM_DEFINITIONS})
SET (SwigGING_FLAGS ${SwigGING_FLAGS} -D${f})
endforeach(f)
```

The last configuration in main CMakeList.txt file is set the target folder of installation.

```
#Install on PYTHON_LIBRARIES - path to the python library
for default
SET (CMAKE_INSTALL_PREFIX ${PYTHON_LIBRARIES}/dist-
packages CACHE STRING "Default installation path." FORCE)

INSTALL (DIRECTORY ${CMAKE_BINARY_DIR}/Foam DESTINATION
${CMAKE_INSTALL_PREFIX}
FILES_MATCHING PATTERN "*.so")
INSTALL (DIRECTORY ${CMAKE_BINARY_DIR}/Foam DESTINATION
${CMAKE_INSTALL_PREFIX}
FILES_MATCHING PATTERN "*.py")

INSTALL (DIRECTORY "${PROJECT_SOURCE_DIR}/Foam"
DESTINATION ${CMAKE_INSTALL_PREFIX}
FILES_MATCHING PATTERN "*.py")
```

We should take in account that because usually, the common user on a Unix/Linux system, installation of Python modules in the standard system's directory hierarchy is not allowed due to security reasons. Therefore the environment variable “PYTHONPATH” is defined and points to a directory within the users directory hierarchy ($HOME). Then, a Python script that intends to load a module, include the value $PYTHONPATH within the search list for locating and loading the required module. We need know this if we place the target folder installation in other directory that not corresponds with standard location for third-party Python modules. The directory to install the Python modules can be defined within the CMake system during configuration.

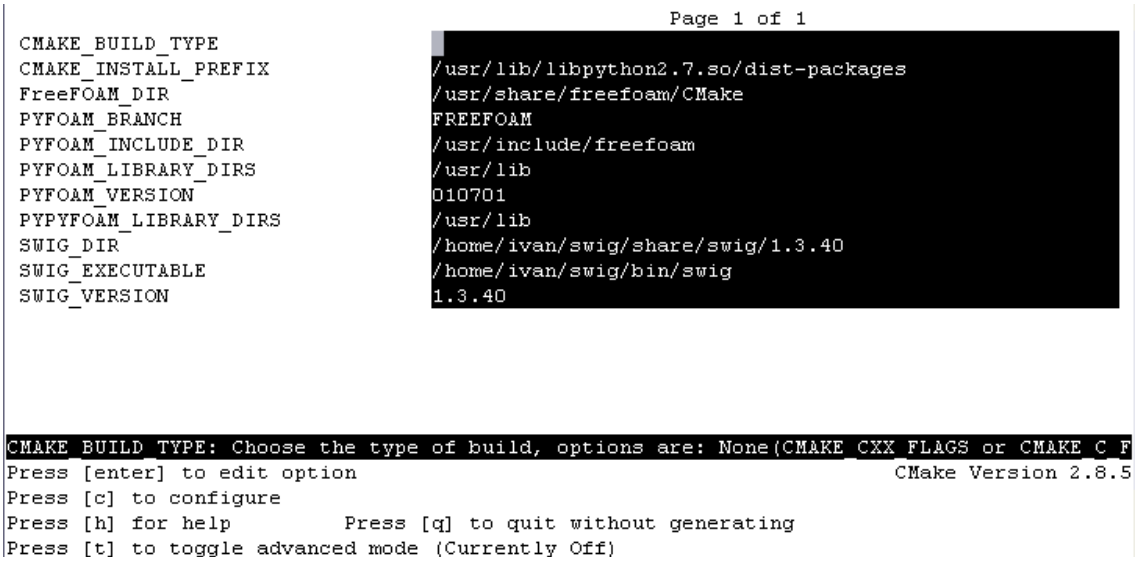

Fig. 4.1 CMake main menu

Each sub-directory of PyFreeFOAM contains a CmakeLists.txt file. This includes the commands to build and install the module. Long commands that are repeated several times, for example for each module, have been defined as a function in the file Cmake/pyfoamUtilities.cmake file under the main directory of the PyFreeFOAM project. The function is than called from the CmakLists.txt of the module. This is one example of one of them:

```
#Reset variables
#############################################################
SET(INTERFACE)
SET(INTERFACE_FILE)
SET(MODULE_NAME)
#############################################################
#Touch all interface files
#############################################################
FILE(GLOB INTERFACE *.i)
#############################################################
#Get all files from directory
#############################################################
FOREACH(item ${INTERFACE})
STRING(REPLACE "${CMAKE_CURRENT_SOURCE_DIR}/" "" item
${item})
IF(item)
LIST(APPEND INTERFACE_FILE ${item})
ENDIF(item)
ENDFOREACH(item ${INTERFACE})
#############################################################
#Replace INTERFACE variable to obtain module name
#############################################################
FOREACH(item ${INTERFACE_FILE})
STRING(REPLACE ".i" "" NAME ${item})
IF(NAME)
LIST(APPEND MODULE_NAME ${NAME})
#Give values to obtain python modules
pyfoam_add_module(${NAME})
# message("Module name: ${MODULE_NAME}")
# message("item: ${item}")
# message("--------------------")
ENDIF(NAME)
ENDFOREACH(item ${INTERFACE_FILE})
#############################################################

add_subdirectory (OpenFOAM)
add_subdirectory (finiteVolume)
add_subdirectory (meshTools)
add_subdirectory (randomProcesses)
add_subdirectory (sampling)
add_subdirectory (transportModels)
add_subdirectory (turbulenceModels)
```

In pyfoamUtilities.cmake file we configure the steeps for build and compile every interface file that the previous CmakeList.txt file related collect from PyFreeFOAM directories.

```
SET (LIBRARIES ${PYTHON_LIBRARIES} ${PYFOAM_LIBS})
SET (Swig_DIRECTORIES ${PYFOAM_INCLUDE_DIRS} -I/usr/include -
I/usr/include/c++)
SET (FLAG "\"-I.\" \"-D
DIRECTOR_INCLUDE=<${modulename}PYTHON_wrap.h>\"")

# General Options to swigging
# -c++ Enable C++ processing
# -MM List dependencies, but omit files in Swig librar
# -w508 Declaration of 'XXX' shadows declaration accessible via
'YYY"
# -w317 Specialization of non-template 'XXX'
# -w312 Nested class not currently supported (ignored)
# -w509 Overloaded method 'XXX' is shadowed by 'YYY'
# -w503 Can't wrap 'XXX' unless renamed to a valid identifier
# -w462 Unable to set dimensionless array variable
# -w473 Returning a pointer or reference in a director method
is not recommended

SET (Swig_PARAMETERS -c++ -MMD ${FOAM_DEFINITIONS} -
D__restrict__ -DOpenFOAM_EXPORTS ${PYFOAM_FLAGS} -
I${COMMON_DIR}/Foam/patches/r1.7.1-free -
I${COMMON_DIR}/Foam/patches/r1.5-free ${Swig_DIRECTORIES}
-w508 -w312 -w509 -w503 -w462 -w473 -module ${modulename})
```

We define in pyfoam_add_module function the properties, required to Cmake, of interface files needed in compilation process.

Using Cmake we could run two modules that give us the availability to use Swig of a simple form. Cmake includes a modules that supports the generation of SWIG wrapper libraries. The SWIG package defines the following macros, SWIG_ADD_MODULE and SWIG_LINK_LIBRARIE:

- **Swig_ADD_MODULE**: Define swig module with given name and specified language
- **Swig_LINK_LIBRARIES:** The libraries that have to be linked to the module

With few lines of code we set the properties to specify special behaviors of Swig process for all modules that Cmakifles.txt passed to pyfoamUtilities.cmake file.

```
SET_SOURCE_FILES_PROPERTIES(${modulename}.i PROPERTIES
CPLUSPLUS ON)

SET_SOURCE_FILES_PROPERTIES(${modulename}.i PROPERTIES
LANGUAGE CXX)

SET_SOURCE_FILES_PROPERTIES(${modulename}.i PROPERTIES
Swig_FLAGS "${Swig_PARAMETERS}")

SET_SOURCE_FILES_PROPERTIES(${modulename}PYTHON_wrap.cxx
PROPERTIES COMPILE_FLAGS "${FLAG}")

Swig_ADD_MODULE(${modulename} python ${modulename}.i
${modulename}.hh)

Swig_LINK_LIBRARIES(${modulename} ${LIBRARIES})
```

C. *Rewrite the solver application's in python language to run with python module compiled in steep D.*

After compilation and installation all python modules and libraries are copied to target directory. This should be accessible from python interpreter. We could use the environment variable PYTHONPATH[13] related above. A solver provided by FreeFOAM can now be executed from python script or interactively from a Python console. As an example we will show of the FreeFOAM solver ico, called icoFoam in the OpenFOAM project following the nomenclature explained in chapter 2.1. The first lines of icoFoam.C are:

```
#include "setRootCase.H"

#include "createTime.H"

#include "createMesh.H"
```

Do the same as the following python lines:

```
from Foam.OpenFOAM.include import setRootCase
args = setRootCase( argc, argv )
from Foam.OpenFOAM.include import createTime
runTime = createTime( args )
from Foam.OpenFOAM.include import createMesh
mesh = createMesh( runTime )
```

For each module, an identical procedure will have to be performed:

- parsing and analyzing command-line arguments
- creating Time and fvMesh objects

The next definition in icoFoam.C file another important clarification in solver definition introduced by pyFreeFOAM:

```
#include "createFields.H"
```

in comparison with the referenced PyFreeFOAM line :

```
transportProperties, nu, p, U, phi, pRefCell,
pRefValue = createFields( runTime, mesh )
```

PyFreeFOAM clearly states that "createFields" functionality

- defines the following variables: transportProperties, nu, p, U, phi, pRefCell, pRefValue
- in order to do its job it requires Time and fvMesh objects as its arguments

As the result, you can clearly see that these magic transportProperties, nu, p, U, phi, pRefCell, pRefValue variables come exactly from "createFileds" function and that "createTime" and "createMesh" should be strictly called before it, to provide necessary input arguments -runTime and mesh.

A solver equation in pyFreeFOAM may be written as:

```
from Foam import fvm UEqn = (fvm.ddt(U) + fvm.div(phi,U) -
fvm.laplacian(nu,U))

from Foam import fvc

from Foam.finiteVolume import solve

solve(UEqn == -fvc.grad(p))
```

Compared to the C++ code in FreeFOAM:

```
fvVectorMatrix UEqn(fvm::ddt(U) + fvm::div(phi,U) -
fvm::laplacian(nu,U));

solve( UEqn == -fvc::grad( p ) );
```

There are other two important points :

- C++ namespaces (like fmv and fvc) in pyFreeFOAM are reflected into the corresponding Python modules (Foam.fvm and Foam.fvc)
- If we need to use a function we need to load the corresponding library first. To call solve, which is defined in fintiteVolume library, you need to load it first by executing "from Foam.finiteVolume import solve"

These points show that in PyFreeFOAM there is no distinction between the library and namespace notions.

We can see that pyFreeFOAM code is very similar to the related FreeFOAM code. Some pyFreeFOAM C++ differences, like usage of namespaces, includes and libraries. To be able to program solvers in python terms we need have knowledge of the FreeFOAM API.

CAPÍTOL 5. Conclusions

The open source OpenFOAM CFD code are growing very fast, has a good prospect in the future. Using CFD open source software we don't should spend money in licenses. We could said the same about FreeFOAM. Maybe in short time, open source CFD will have same peer to peer capability against commercial CFD software like as ANSYS. However, scientific efforts are required when using open source programs compared to the commercially available programs. Working with open source CFD codes faces less support, difficultness in numerical coding and lack of proper manuals, so that greater resources are required to familiarize the users in the programs.

Therefore the availability of work from python with this CFD software might be one of the solutions to solve these limitations. Tools like Swig make it possible to call functions and access data structures in C/C++ directly from a Python script. In other words, now we can almost automatically build a complete Python interface to existing FreeFOAM libraries. This is ideal for prototyping and testing as there is no waiting time for compilation and linking when programming in Python. It is possible to do this without losing the efficiency that allows us C++ code. Also we must think that the curve of learning in Python is exponential and to do scripts with this it is much easier and carries less time that develop software with C++. FreeFOAM and OpenFOAM have a great documentation and we could address our problems with them in CFD online Forums.

On the other hand we could not have proved these Python scripts with Salome. In the moment to carry out the tests, Salome's available version was incompatible with our system Debian. His installation was implying uninstall some packages that could to do the unstable operating system. We develop this project under Debian SID. This distribution will never get released; instead, packages from it will propagate into testing and then into a real release. Debian SID distribution is subject to massive changes and in-place library updates. This can result contains packages that cannot be installed due to missing libraries, dependencies that cannot be fulfilled.

In future lines we could port PyFreeFOAM to work under OpenFOAM, now we only could obtain Python modules and libraries from FreeFOAM. Other improvement to take in account is that the existing interface files of PyFreeFOAM only could be used with Swig 1.3.40 version. Is availability new version, the Swig 2.0.4 has the latest version released at this time.

At finally, the use of CMake can make us think that in a future we could port open source CFD software to other platforms such as Windows.

BIBLIOGRAPHY

[1] FLUENT, A. *A Brief History of CFD*. 2009:
http://coe.uncc.edu/~pramapra/Teaching/CFD_intro.pdf

[2] Computational fluid dynamics:
http://en.wikipedia.org/wiki/Computational_fluid_dynamics
http://www.cfd-online.com/Wiki/General_CFD_FAQ

[3] FOSS:
http://en.wikipedia.org/wiki/Free_and_open_source_software

[4] OpenFOAM:
http://www.openfoam.com

[5] FreeFOAM:
http://freefoam.sourceforge.net/

[6] CMake from Kitware INC.:
http://www.cmake.org/
http://www.elpauer.org/stuff/learning_cmake.pdf

[7] Wmake:
http://www.openwatcom.org/

[8] Python:
http://www.python.org

[9] PythonFlu:
http://openfoamwiki.net/index.php/Contrib_pythonFlu

[10] Vulashaka project:
http://www.openfoamworkshop.org/2009/4th_Workshop/0_Feature_Presentatio
ns/OFW4_2009_Gschaider_PyFoam.pdf
http://www.cfd-online.com/Forums/openfoam/74841-can-someone-tell-me-
more-about-vulashaka.html

[11] PyFOAM:
http://sourceforge.net/apps/mediawiki/pyfoam/index.php?title=Main_Page

[12] Swig:
http://www.swig.org
http://www.swig.org/papers/PyTutorial98/PyTutorial98.pdf
http://www.dabeaz.com/SwigMaster/SWIGMaster.pdf

[13]Pythonpath variable
http://docs.python.org/tutorial/modules.html

[14]PyFreeFOAM repository
https://github.com/ivan7farre/PyFreeFOAM

ANNEXE I. CMakeList.txt main file

```
# include utilities
include(${CMAKE_SOURCE_DIR}/CMake/pyfoamUtilities.cmake)
include(${CMAKE_SOURCE_DIR}/CMake/PYFOAMDetermineArch.cmake)
#set(FreeFOAM_DIR "/usr/local/share/freefoam/CMake")
#Define PROJECT_SOURCE_DIR
SET (COMMON_DIR ${PROJECT_SOURCE_DIR})

# Make sure that Python (developement version, with Python.h header file) is
# installed (tested with 2.4, 2.6.2)
# Please follow instructions from http://www.python.org/ web-site.
# find python interpreter
# find_package(PythonInterp REQUIRED)
## From http://www.cmake.org/Wiki/CMake_FAQ
FIND_PACKAGE(PythonLibs)
INCLUDE_DIRECTORIES(${PYTHON_INCLUDE_DIR})

# Make sure that SWIG is installed (tested with 1.3.36 - 1.3.40 )
#  Please follow instructions from http://www.swig.org/ web-site.
# From http://www.cmake.org/Wiki/CMake_FAQ:
#FIND_PACKAGE(SWIG REQUIRED)
FIND_PACKAGE(SWIG 1.3.40 EXACT REQUIRED)
INCLUDE(${SWIG_USE_FILE})

#FreeFOAM enviroment
FIND_PACKAGE(FreeFOAM REQUIRED)
INCLUDE(${FOAM_USE_FILE})
#message ("FOAM_INCLUDE_DIRS: ${FOAM_INCLUDE_DIRS}")
#message ("FOAM_LIBRARY_DIRS: ${FOAM_LIBRARY_DIRS}")

SET (PYFOAM_INCLUDE_DIR ${FOAM_INCLUDE_DIRS} CACHE STRING
"Foam Path." FORCE)
SET (PYFOAM_LIBRARY_DIRS ${FOAM_LIBRARY_DIRS} CACHE STRING
"Foam Libs." FORCE)
SET (PYFOAM_BRANCH FREEFOAM CACHE STRING "Choose the type of
Foam OpenSource distribution." FORCE)

## Gerber: FreeFOAM uses FOAM_UPSTREAM_VERSION_FULL, but is not
public available. Contact Michael.
SET (PYFOAM_VERSION 010701 CACHE STRING "Choose the
corresponding version of OpenFOAM used. Could be
\"010701\",\"010700\",\"010600\",\"010500\"." FORCE)

SET (PYFOAM_FLAGS -D__FOAM_VERSION__=${PYFOAM_VERSION} -
D__FOAM_BRANCH__=__${PYFOAM_BRANCH}__)
SET (PYFOAM_INCLUDE_DIRS -I${PYFOAM_INCLUDE_DIR}/finiteVolume -
I${PYFOAM_INCLUDE_DIR}/sampling -
```

```
I${PYFOAM_INCLUDE_DIR}/basicThermophysicalModels -
I${PYFOAM_INCLUDE_DIR}/meshTools -
I${PYFOAM_INCLUDE_DIR}/incompressibleLESModels -
I${PYFOAM_INCLUDE_DIR}/incompressibleRASModels -
I${PYFOAM_INCLUDE_DIR}/compressibleLESModels -
I${PYFOAM_INCLUDE_DIR}/compressibleRASModels -
I${PYFOAM_INCLUDE_DIR}/radiation -
I${PYFOAM_INCLUDE_DIR}/incompressibleTransportModels -
I${PYFOAM_INCLUDE_DIR}/interfaceProperties -
I${PYFOAM_INCLUDE_DIR}/dynamicFvMesh -
I${PYFOAM_INCLUDE_DIR}/dynamicMesh -
I${PYFOAM_INCLUDE_DIR}/randomProcesses -
I${PYFOAM_INCLUDE_DIR}/OpenFOAM -
I${PYFOAM_INCLUDE_DIR}/OSspecific)
ADD_DEFINITIONS(${PYFOAM_FLAGS} ${PYFOAM_INCLUDE_DIRS})

foreach(f ${FOAM_DEFINITIONS})
   SET (SWIGGING_FLAGS ${SWIGGING_FLAGS} -D${f})
endforeach(f)


#Install on PYTHON_LIBRARIES - path to the python library for default
SET (CMAKE_INSTALL_PREFIX ${PYTHON_LIBRARIES}/dist-packages
CACHE STRING "Default installation path." FORCE)

INSTALL (DIRECTORY ${CMAKE_BINARY_DIR}/Foam DESTINATION
${CMAKE_INSTALL_PREFIX}
      FILES_MATCHING PATTERN "*.so")
INSTALL (DIRECTORY ${CMAKE_BINARY_DIR}/Foam DESTINATION
${CMAKE_INSTALL_PREFIX}
      FILES_MATCHING PATTERN "*.py")

INSTALL (DIRECTORY "${PROJECT_SOURCE_DIR}/Foam" DESTINATION
${CMAKE_INSTALL_PREFIX}
      FILES_MATCHING PATTERN "*.py")

add_subdirectory (Foam)
```

ANNEXE II. CMakeList.txt file locate in subfolders

```
cmake_minimum_required(VERSION 2.8.0)
project (pyFluFreeFOAM)
cmake_minimum_required(VERSION 2.8.0)
project (pyFluFreeFOAM)

#Reset variables
#############################################################
###########
SET(INTERFACE)
SET(INTERFACE_FILE)
SET(MODULE_NAME)
```

```
#####################################################################
###########

#Touch all interface files
#####################################################################
###########
FILE(GLOB INTERFACE *.i)
#####################################################################
###########

#Get all files from directory
#####################################################################
###########
FOREACH(item ${INTERFACE})
  STRING(REPLACE "${CMAKE_CURRENT_SOURCE_DIR}/" "" item ${item})
  IF(item)
    LIST(APPEND INTERFACE_FILE ${item})
  ENDIF(item)
ENDFOREACH(item ${INTERFACE})
#####################################################################
###########

#Replace INTERFACE variable to obtain module name
#####################################################################
###########
FOREACH(item ${INTERFACE_FILE})
  STRING(REPLACE ".i" "" NAME ${item})
  IF(NAME)
    LIST(APPEND MODULE_NAME ${NAME})
    #Give values to obtain python modules

    pyfoam_add_module(${NAME})
#    message("Module name: ${MODULE_NAME}")
#    message("item: ${item}")
#    message("--------------------")
  ENDIF(NAME)

ENDFOREACH(item ${INTERFACE_FILE})
#####################################################################
###########


add_subdirectory (OpenFOAM)
add_subdirectory (finiteVolume)
add_subdirectory (meshTools)
add_subdirectory (randomProcesses)
add_subdirectory (sampling)
add_subdirectory (transportModels)
add_subdirectory (turbulenceModels)
```

ANNEXE III. pyfoamUtilities.cmake file

```
function(pyfoam_add_module modulename)

FIND_PACKAGE(PythonInterp)
FIND_PACKAGE(PythonLibs)
INCLUDE_DIRECTORIES(${PYTHON_INCLUDE_DIR}
${CMAKE_CURRENT_SOURCE_DIR} ${COMMON_DIR})

SET(PYFOAM_LIBS ${PYFOAM_LIBRARY_DIRS}/libfiniteVolume.so
${PYFOAM_LIBRARY_DIRS}/libsampling.so
${PYFOAM_LIBRARY_DIRS}/libbasicThermophysicalModels.so
${PYFOAM_LIBRARY_DIRS}/libmeshTools.so
${PYFOAM_LIBRARY_DIRS}/libincompressibleLESModels.so
${PYFOAM_LIBRARY_DIRS}/libincompressibleRASModels.so
${PYFOAM_LIBRARY_DIRS}/libincompressibleTurbulenceModel.so
${PYFOAM_LIBRARY_DIRS}/libcompressibleLESModels.so
${PYFOAM_LIBRARY_DIRS}/libcompressibleRASModels.so
${PYFOAM_LIBRARY_DIRS}/libcompressibleTurbulenceModel.so
${PYFOAM_LIBRARY_DIRS}/libradiation.so
${PYFOAM_LIBRARY_DIRS}/libincompressibleTransportModels.so
${PYFOAM_LIBRARY_DIRS}/libinterfaceProperties.so
${PYFOAM_LIBRARY_DIRS}/libtwoPhaseInterfaceProperties.so
${PYFOAM_LIBRARY_DIRS}/libdynamicFvMesh.so
${PYFOAM_LIBRARY_DIRS}/libdynamicMesh.so
${PYFOAM_LIBRARY_DIRS}/librandomProcesses.so
${PYFOAM_LIBRARY_DIRS}/libspecie.so
${PYFOAM_LIBRARY_DIRS}/libOpenFOAM.so
${PYFOAM_LIBRARY_DIRS}/libtriSurface.so
${PYFOAM_LIBRARY_DIRS}/libsurfMesh.so
${PYFOAM_LIBRARY_DIRS}/libdecompositionMethods.so
${PYFOAM_LIBRARY_DIRS}/liblagrangian.so
${PYFOAM_LIBRARY_DIRS}/libLESdeltas.so
${PYFOAM_LIBRARY_DIRS}/libLESfilters.so
${PYFOAM_LIBRARY_DIRS}/libscotchDecomp.so
${PYFOAM_LIBRARY_DIRS}/plugins1/libmpiPstream.so)

SET (LIBRARIES ${PYTHON_LIBRARIES} ${PYFOAM_LIBS})
SET (SWIG_DIRECTORIES ${PYFOAM_INCLUDE_DIRS} -I/usr/include -
I/usr/include/c++)
SET (FLAG "\"-I.\" \"-D
DIRECTOR_INCLUDE=<${modulename}PYTHON_wrap.h>\"")

#General Options
# -c++   Enable C++ processing
# -MM    List dependencies, but omit files in SWIG librar
# -w508  Declaration of 'XXX' shadows declaration accessible via 'YYY"
```

```
# -w317  Specialization of non-template 'XXX'
# -w312  Nested class not currently supported (ignored)
# -w509  Overloaded method 'XXX' is shadowed by 'YYY'
# -w503  Can't wrap 'XXX' unless renamed to a valid identifier
# -w462  Unable to set dimensionless array variable
# -w473  Returning a pointer or reference in a director method is not recommended

SET (SWIG_PARAMETERS -c++ -MMD ${FOAM_DEFINITIONS} -D__restrict__ -DOpenFOAM_EXPORTS ${PYFOAM_FLAGS} -I${COMMON_DIR}/Foam/patches/r1.7.1-free -I${COMMON_DIR}/Foam/patches/r1.5-free ${SWIG_DIRECTORIES} -w508 -w317 -w312 -w509 -w503 -w462 -w473 -module ${modulename})

SET_SOURCE_FILES_PROPERTIES(${modulename}.i PROPERTIES CPLUSPLUS ON)
SET_SOURCE_FILES_PROPERTIES(${modulename}.i PROPERTIES LANGUAGE CXX)
SET_SOURCE_FILES_PROPERTIES(${modulename}.i PROPERTIES SWIG_FLAGS "${SWIG_PARAMETERS}")
SET_SOURCE_FILES_PROPERTIES(${modulename}PYTHON_wrap.cxx PROPERTIES COMPILE_FLAGS "${FLAG}")

SWIG_ADD_MODULE(${modulename} python ${modulename}.i ${modulename}.hh)
SWIG_LINK_LIBRARIES(${modulename} ${LIBRARIES})

endfunction()
```

ANNEXE IV. sphericalTensor.H patch file

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2010 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
```

```
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Typedef
    Foam::sphericalTensor

Description
    SphericalTensor of scalars.

SourceFiles
    sphericalTensor.C

\*---------------------------------------------------------------------------*/

#ifndef sphericalTensor_H
#define sphericalTensor_H

#include <OpenFOAM/SphericalTensor_.H>
#include <OpenFOAM/contiguous.H>

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

typedef SphericalTensor<scalar> sphericalTensor;
#ifndef SWIG
// Identity tensor
static const sphericalTensor I(1);

static const sphericalTensor oneThirdI(1.0/3.0);
static const sphericalTensor twoThirdsI(2.0/3.0);
#else
// Identity tensor
static const sphericalTensor I;

static const sphericalTensor oneThirdI;
static const sphericalTensor twoThirdsI;
#endif


//- Specify data associated with sphericalTensor type are contiguous
template<>
inline bool contiguous<sphericalTensor>() {return true;}
```

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

// ************************ vim: set sw=4 sts=4 et: ************************ //
```

ANNEXE V. SymmTensor interface file (S_ymmTensor.i)

```
//---------------------------------------------------------------------------
#ifndef SymmTensor_i
#define SymmTensor_i


//---------------------------------------------------------------------------
%module "Foam.src.OpenFOAM.primitives.S_ymmTensor";
%{
  #include "Foam/src/OpenFOAM/primitives/S_ymmTensor.hh"
%}


//---------------------------------------------------------------------------
%import "Foam/src/OpenFOAM/primitives/vector.i"

%import "Foam/src/OpenFOAM/primitives/S_phericalTensor.i"

#if FOAM_NOT_BRANCH( __FREEFOAM__ )
%include <SymmTensor.H>
#else
%include <SymmTensor_.H>
#endif


//---------------------------------------------------------------------------
#endif
```