NanoDome

H2020-NMP-2014-two-stage

NMP-20-2014 Widening materials models

Grant Agreement n. 646121

DELIVERABLE 4.2

Interface and Linking Description Document

Contents

1 Introduction

This document is to be intended as an extension of the source code provided for the Deliverable 4.2, concerning the implementation of the libraries for linking/coupling the mesoscale core software with third party CFD simulation environment. This linker is specifically designed for exchanging data between *NanoDome* and *ANSYS FLUENT*.

The linking/coupling process can be summarized in the diagram in figure 1.

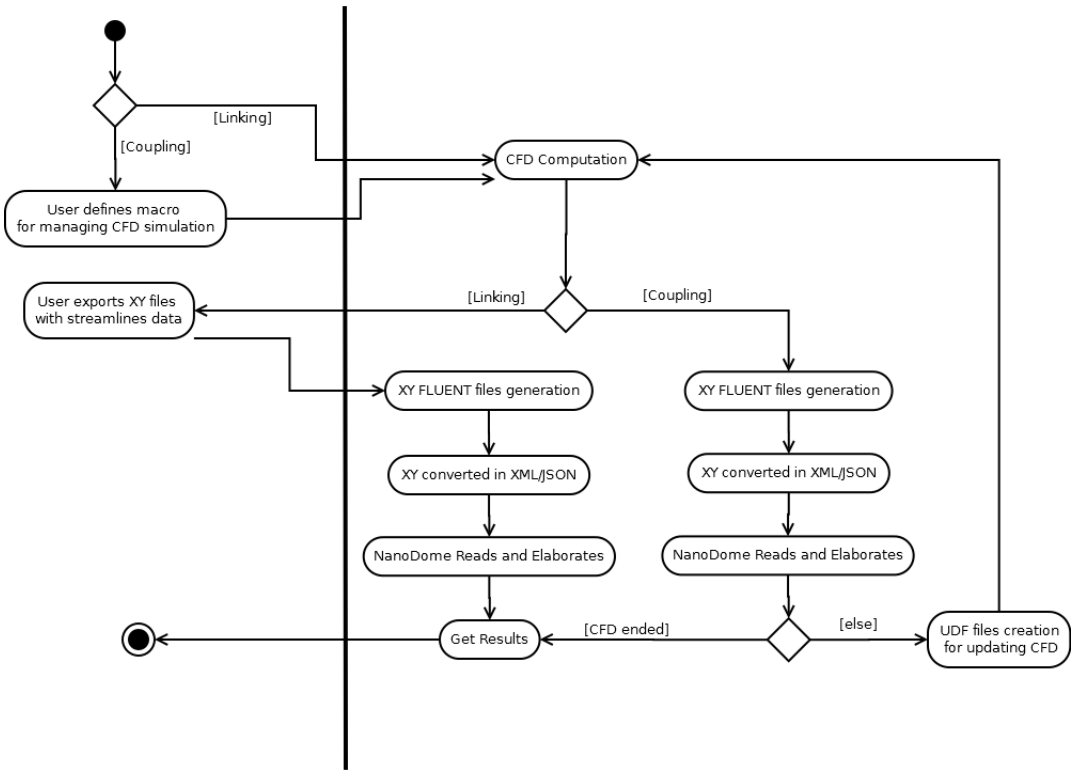

Figure 1: Linking/Coupling workflow

Both processes are based on the creation of a set of XY *FLUENT* files, exporting the features describing the streamlines, as requested in the Deliverable 4.1. The main difference between the two approaches is that for the coupling side, the user creates a macro script for automating the exporting process for the files, the calls to the converter and the mesoscopic simulation executables. Once the mesoscopic simulations completed, is created an UDF file for updating the quantities inside the CFD environment and restart the simulation. For the linking side, once the CFD simulation is completed, the user exports, using the dialog boxes provided by the *FLUENT* UI, the data needed in XY format, then using the

converter, creates the XML/JSON file for feeding *NanoDome*.

We propose a synopsis where we describe at an higher level of abstraction the software behavior and also provide a guide for the utilization by the end user.

2 Software Synopsis

This software is a file converter from the XY *ANSYS FLUENT* data exporting format to XML and JSON schemes. The file format for exchanging data from/to CFD designed for *NanoDome* is described in the Deliverable 4.1. The data extraction for the linking/coupling by means of the streamlines inside the CFD simulation in the FLUENT environment has been designed exploiting the native input/output functions provided by the UI of the software. We implemented a file converter instead of a software module inside the *FLUENT* environment for these reasons:

1. The *FLUENT* source code is not very well documented, making the identification of the proper data structures for this task extremely difficult.

2. There are not specific data structures describing the streamline itself. The data relative to the streamlines are created, once requested, on the fly by the software itself.

3. This approach also gives us the assurance that the data retrieved are correct and consistent.

2.1 Source Code and Classes Description

In this subsection we provide information regarding the source code organization and implementation. Following the UML schema in figure 2, the main class is the `Converter` class that uses the `XY_parser` class for extracting the raw data stored in the XY files.

Once retrieved the raw data, defined in the `Raw_data` class, we store it in memory exploiting the data structures described by the `streamline` class and by means of the `streamlinefileXML` and `streamlinefileJSON` classes, that encapsulate the XML (tinyxml2) and JSON (Simple JSON) files management libraries, we create and store the output files.

In detail, the behavior of the software is described in the UML Activity Diagram in figure 3.

For the Microsoft Windows version, we exploited an ad-hoc implementation of the `dirent.h` library, native in Unix, for retrieving the file names inside a folder. This has been done for using the the same code and function calls in both the environments. At compiling time, using preprocessor directives, the code

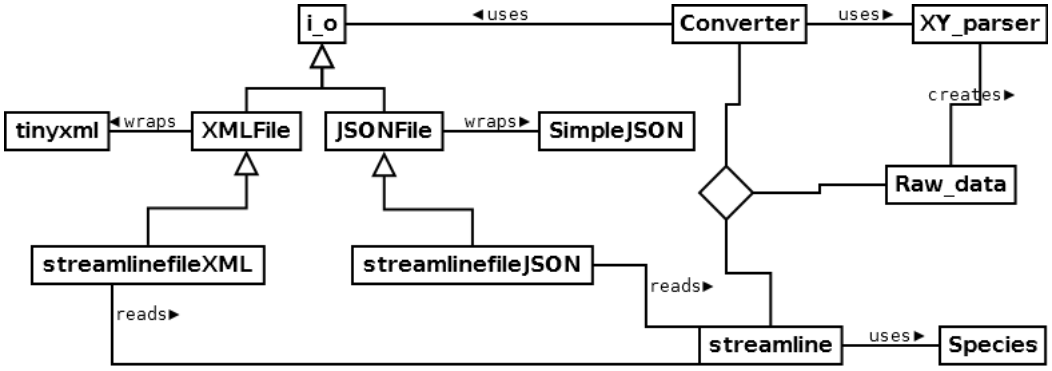

Figure 2: Linker Class Diagram

understands the environment in which is compiled and automatically turn on the proper lines of code (`XY_parser` class).

The detailed description of the attributes and methods implemented in the classes is described in the Doxygen documentation provided with the code.

3 Compilation and Utilization Details

The package is divided in several folders:

- `JSON`: containing the JSON library source code.
- `tinyxml2`: containing the XML library source code.
- `win_dir_libs`: containing the library source code for the `dirent.h` windows porting.

3.1 Linux Environment

For compiling the code is sufficient to have the latest version g the GNU (`g++`) compiler or the Intel (`icpc`) compiler. For creating the executable, is sufficient to go inside the main folder of the source code and type the following command in the terminal:

```
g++(icpc) -std=c++11 *.cpp JSON/*.cpp tinyxml2/*.cpp
-o exeName
```

for lauching the converter:

```
./exeName in_dir out_dir (x/j/xj)
```

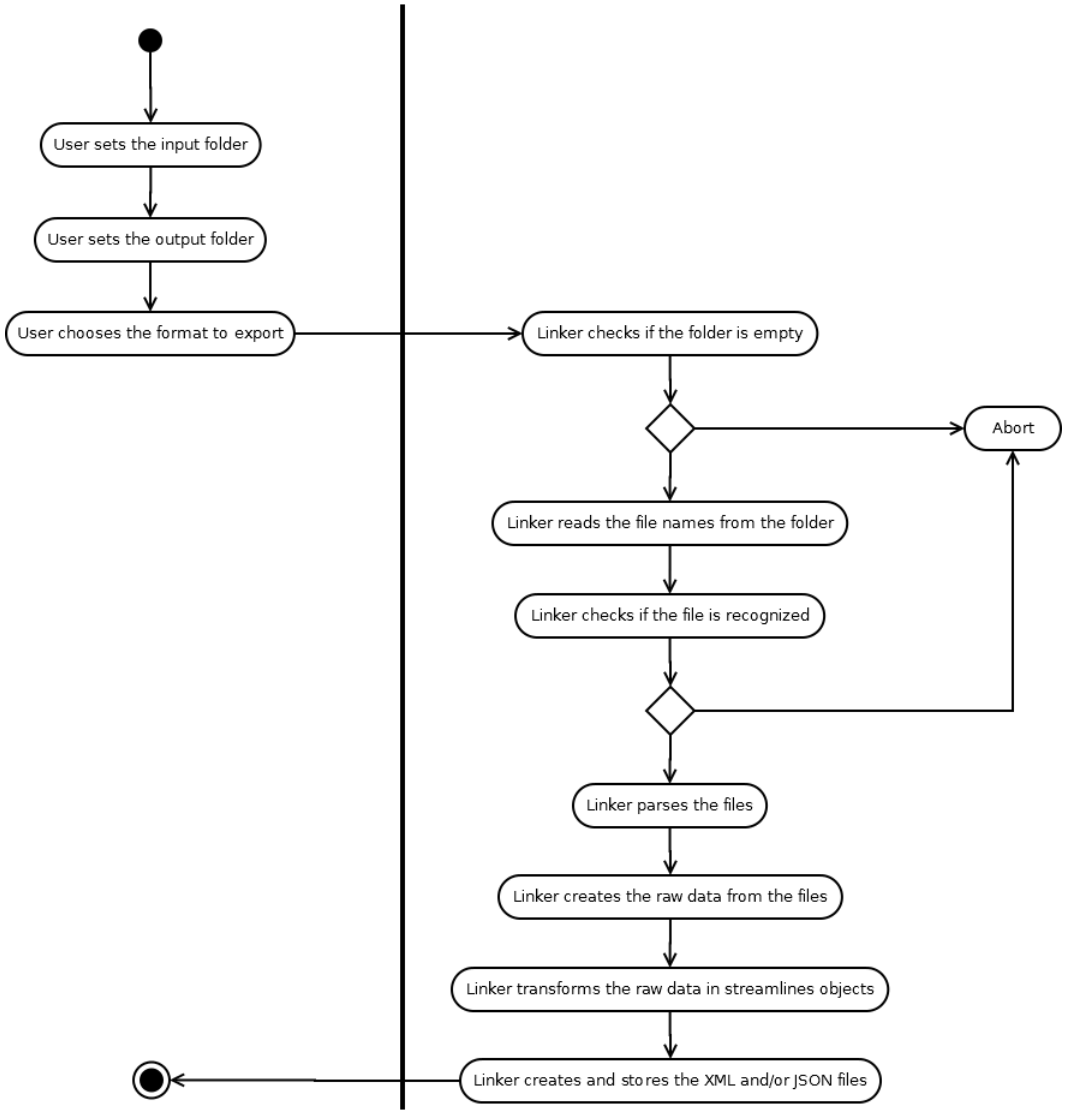

Figure 3: Linker Activity Diagram

For example, if we want to create an instance of the linker called `FLUENT_Link` that reads FLUENT files from the directory `source` and creates both XML and JSON files in the `nanodome_link` folder we can type:

```
g++ -std=c++11 *.cpp JSON/*.cpp tinyxml2/*.cpp -o FLUENT_Link
./FLUENT_Link source nanodome_link xj
```

The options for the generation of the files are:

- x: for generating only the XML file.
- j: for generating only the JSON file.
- xj: for generating both JSON and XML files.

3.2 Windows Environment

Using the command is the same procedure of the previous section, but adding in the set of file to compile also the ones in the `win_dir_lib` folder.

Using *Visual Studio*, is sufficient to create a blank Visual C/C++ project adding the files inside the package, keeping attention to add also the ones in the `win_dir_lib` folder