Agenzia nazionale per le nuove tecnologie,
l'energia e lo sviluppo economico sostenibile

MINISTERO DELLO SVILUPPO ECONOMICO

FELMCORE-CATHARE COUPLING ON SALOME PLATFORM

A. Cervone, A. Attavino, D, Cerroni, L. Fancellu,
S. Manservisi

Report RdS/PAR2014/213

FELMORE-CATHARE COUPLING ON SALOME PLATFORM

A. Cervone - ENEA, A. Attavino, D. Cerroni, L. Fancellu, S. Manservisi - CIRTEN, Università di Bologna

Settembre 2015

Report Ricerca di Sistema Elettrico
Accordo di Programma Ministero dello Sviluppo Economico - ENEA
Piano Annuale di Realizzazione 2014
Area: Produzione di energia elettrica e protezione dell'ambiente
Progetto: Sviluppo competenze scientifiche nel campo della sicurezza nucleare e collaborazione ai programmi internazionali per il nucleare di IV Generazione
Linea: Collaborazione internazionale per il nucleare di IV generazione
Obiettivo:Termoidraulica del refrigerante
Responsabile del Progetto: Mariano Tarantino, ENEA

Il presente documento descrive le attività di ricerca svolte all'interno dell'Accordo di collaborazione "Sviluppo competenze scientifiche nel campo della sicurezza e collaborazione ai programmi internazionali per il nucleare di IV Generazione"
Responsabile scientifico ENEA: Mariano Tarantino
Responsabile scientifico CIRTEN: Giuseppe Forasassi

ENEA Ricerca Sistema Elettrico	Sigla di identificazione	Distrib.	Pag.	di
	ADPFISS – LP2 – 106	L	1	94

Titolo

FEMLCORE–CATHARE COUPLING ON SALOME PLATFORM

Descrittori

Tipologia del documento:	Rapporto Tecnico
Collocazione contrattuale:	Accordo di programma ENEA-MSE su sicurezza nucleare e reattori di IV generazione
Argomenti trattati:	termoidraulica del nocciolo, Generation IV reactors

Sommario

In this report we present the development of the FemLCore-Cathare code coupling for the simulation of the dynamics of the primary loop of a lead-cooled nuclear reactor. In the framework of a multi-scale and multi-physic approach we develop a class interface for the FemLCore modules and another for the Cathare system code that allow data transfer with a supervisor function. The numerical coupled solution is obtained by using the defective coupling approach. The problem is solved over a mono-dimensional mesh by using the system code and its solution is then corrected over the overlapping three-dimensional domain. The Cathare code solves and generates boundary conditions for the FemLCore modules in order to receive back the three-dimensional solution corrections in the form of appropriate mass, momentum and energy sources. The input-output formats for meshes and solutions are in MedMem format. Data are exchanged at each time step through a supervisor code written in c++ language. Two coupling tests are performed. In the first test a simple three-dimensional core model reactor and a primary mono-dimensional loop, which consists of a pump and a heat exchanger, are considered. In the second test a three-dimensional CFD-porous reactor with core and lower-upper plenum is coupled with a primary mono-dimensional loop.

Note

Il presente contributo è stato preparato con il contributo del personale ENEA e CIRTEN:
A. Cervone (ENEA)
A. Attavino, D. Cerroni, L. Fancellu, S. Manservisi (CIRTEN - Università di Bologna)
Sigla documento rif.: CERSE-UNIBO RL1361/2015

Copia n. **In carico a:**

2			NOME			
			FIRMA			
1			NOME			
			FIRMA			
0	EMISSIONE	14/09/15	NOME	A. CERVONE	M. TARANTINO	M. TARANTINO
			FIRMA	[illegible]	[illegible]	[illegible]
REV.	DESCRIZIONE	DATA		REDAZIONE	CONVALIDA	APPROVAZIONE

FEMLCORE–CATHARE coupling on SALOME platform

A. Attavino, D. Cerroni, A. Cervone, L. Fancellu, and S. Manservisi

Monday 14th September, 2015

Contents

Introduction

A nuclear plant is a very complex machine and its numerical simulation is nowadays above the power computability of the actual computers. A complete study of this machine is multi-physics and multiscale and every detailed engineering study should be limited to well defined regions and a single-physical phenomenon. If the three-dimensional effects are ignored the components of the nuclear plant can be modeled by simple volumetric balances of energy, momentum and mass. The 0-dimension components can be connected with simple mono-dimensional pipes. The mono-dimensional pipes describe the evolution of the state fields interacting with the external environment by adding external sources in the mono-dimensional conservation equations. The Cathare code and other system codes were developed based on this simple hypotheses [5, 6]. These codes have been developed long time ago and during this period has been continuously updated and verified with many experiments [2, 5].

In many situations the mono-dimensional model cannot explain how the system evolves since bi or three-dimensional effects are important. However the three-dimensional simulation of large o complex parts of the system are not possible. In fact nowadays one can simulate only small parts of the nuclear reactor when realistic boundary conditions are available. The coupling techniques between system codes, which may give appropriate boundary conditions, and three-dimensional simulation gives a good opportunity to explore more complex problems.

In early reports we have developed FemLCore code to simulate three-dimensional nuclear reactor components such as the core the upper and the lower plenum. In the previous report we have simulate the the core with a 3D-porous FemLCore module, the plena with a 3D-CFD FemLCore module and the primary loop with a 1D-libMesh or 1D-CFD FemLCore module. In order to improve the quality of the coupling it is very interesting to couple 3D-porous FemLCore module with a mono-dimensional system code that contains trusty information about nuclear power plant such as the Cathare code. It is even more interesting to explore the coupling where the three-dimensional simulations are added as a correction to the mono-dimensional system simulation so that one can evaluate the need of the use of these expensive CFD computations.

In this report we developed a coupling algorithm on the Salome platform between the FemLCore and Cathare code where the numerical approach is a defective correction approach. This means that we solve the Cathare problem over all the domain overlapping partially this mono-dimensional mesh to the three-dimensional one where the solution is obtained with FemLCore module. The mono-dimensional system is always solved generating the boundary conditions for the CFD three-dimensional code and then the three-dimensional solution gives back corrections in the form of sources in the conservation equations to the Cathare code.

Great difficulties arise from the complexity of combining the three and the mono-dimensional mesh together. The input-output formats for mesh and solutions should be compatible. Data should be exchanged at each time step. The Salome platform gives the help needed since an interface compatible with this platform has been developed by the Cathare research group. The Cathare code is not open source and therefore without data transfer functions would be impossible to get and set data inside the code.

Chapter 1 is devoted to the description of the implementation of the Cathare coupling class. There are two classes that can be used to transfer data from Cathare code to the Salome platform: the *ICoCo* package and the *Problem_Cathare* interface developed in this report. The *ICoCo* interface is provided with the Cathare code and uses the *TriouField* format to exchange data with Salome platform. In this work we use a modified version of the *Problem_Cathare* interface which is equipped with graphical class which allows the the evolution of the fields time by time. In this class the transfer with the Salome platform is based on simple c++ *double* type format.

In Chapter 2 there is a brief description of the implementation of the FemLCore coupling class. In this Chapter we point out the main functions to be used inside the supervisor. Details on volume and surface interfaces are introduced. The cmake file for the compilation of the FemLCore and Cathare coupling classes is briefly described.

Chapter 3 starts with a brief description of a standard project of a user application of the type FemLCore-Cathare. Two coupling tests are presented. Test 1 is performed with a simple three-dimensional core model reactor coupled with the primary mono-dimensional loop which consists of a pump and a heat exchanger. In Test 2 a more realistic three-dimensional reactor with core and lower and upper plenum is coupled with a primary mono-dimensional loop. Details on Cathare and FemLCore meshes and on the supervisor code are shown.

1 Cathare on Salome Platform

1.1 Cathare MESH on Salome

Cathare code is a system code essentially capable of computations over mono-dimensional meshes. The mesh is read by Cathare through a text parser which uses its own library commands. This mesh for mono-dimensional computations consists of several groups of directives that defines geometric modules. It can be generated by using the GUI interface on SALOME platform called GUITHARE.

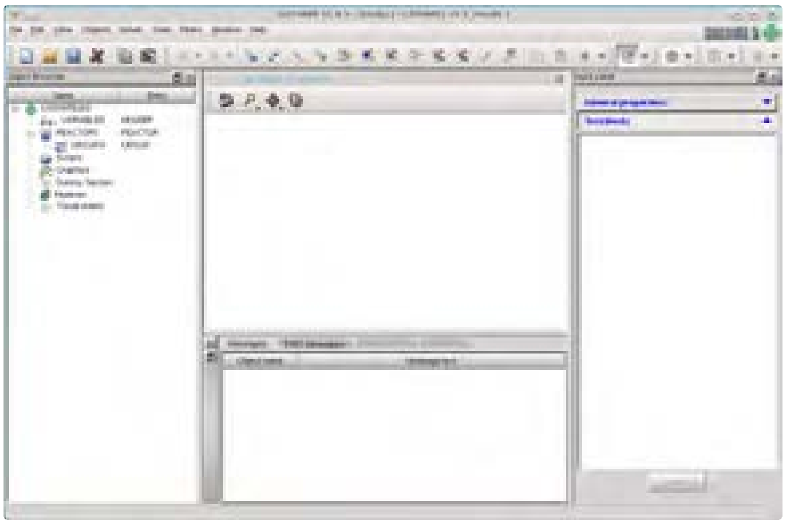


Figure 1.1: GUITHARE main window on Salome 5.

1.1.1 GUITHARE installation

The graphical mesh generator GUITHARE is a GUI written by CEA and EDF, distributed under different forms of licenses. The package is distributed in binary form for different Linux distributions. In this report we use the *open SUSE 13.2* (Harlequin) (x86_64) and therefore the appropriate package is the *GUITHARE_V1.8.5_SUSE9_64bit_20130703.tar.g*
The installation of this binary package presents many problems since the libraries used for the interface GUITHARE are not updated for recent Linux distributions. After performing the untar command of this package the scripts

```
env.csh and  run.csh
```

appear in the main directory. They cannot run in this original form since they are written with dos editors. It is necessary to apply the dos2unix command to both files to delete the *dos* end line control characters. The command for *run.csh* is

```
 dos2unix run.csh .
```

Now the script *run.csh* can run under the following Linux command

```
 ./run.csh .
```

The script cannot open the GUI and an error appears as

```
 error while loading shared libraries: libstdc++.so.5:
   cannot open shared object file: No such file or directory.
```

The library *libstdc++* is the standard C++ library, needed for dynamically linked C++ programs. The version *libstdc++.so.5* is rather old and now the *libstdc++.so.6* is installed in the *OpenSuse 13.2* distribution. In order to install the old library *libstdc++.so.5* one can go to OpenSuse web page. Select *libstdc++33*, the standard C++ shared library, and select the option Click Install on 64 Bit:

```
 devel:gcc    3.3.3    32 Bit    64 Bit 1 Click Install .
```

The old library *libstdc++.so.5* is now installed on the *OpenSuse 13.2* distribution. Even now the script cannot open the GUI and the new following error reads

```
error while loading shared libraries: libpng.so.3:
  cannot open shared object file: No such file or directory.
```

The *libpng.so.3* is an old version of the png package so that in the OpenSuse 13.2 is now installed the library *libpng16.so.16.13.0*. The libraries are not compatible. The library *libpng12.so* is compatible with *libpng.so.3*. By using yast2 the library *libpng12* can be installed and linked properly. The following link commands should be executed

```
ln -s /usr/lib64/libpng12.so   /usr/lib64/libpng.so.3 .
```

Now a new error appears as

```
cannot load library libCathareGUI.so:
  libexpat.so.0 cannot load shared object file
```

In the OpenSuse 13.2 is now installed the *libexpat1.so*. The *libexpat0.so* can be installed by using yast2 together with *libexpat1.so* in this Linux distribution.

Finally the main GUITHARE window appears as shown in Figure 1.1.

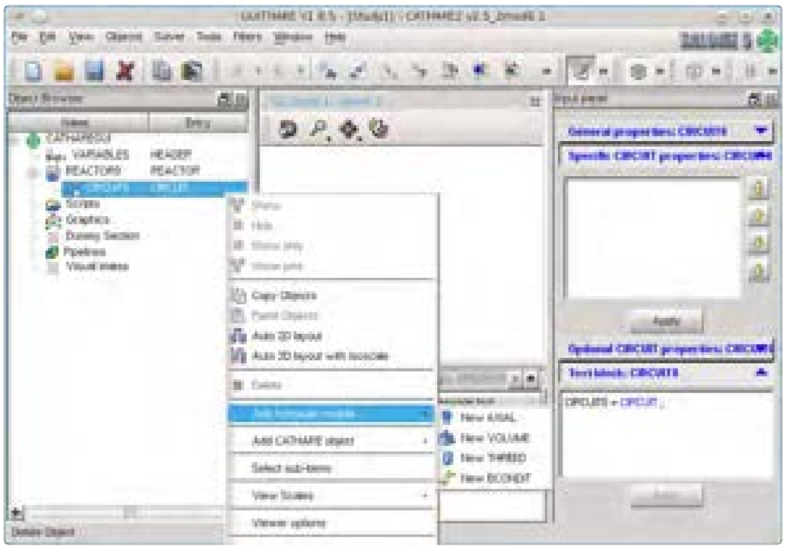


Figure 1.2: Cathare main hydraulic modules.

1.1.2 Cathare modular structure

Information on Cathare code can be found on the Cathare documentation [1, 2, 3, 5, 6, 7]. In the Cathare code several main modules are assembled to model sub-component circuits.

Different circuits can be connected to assemble the main plant circuit (reactor0). The main modules in Cathare are four: *AXIAL*, *VOLUME*, *THREED* and *BCONDIT*. They can be added to the circuit (called CIRCUIT0) by selecting, in the GUITHARE main window, the *add custom hydraulic module* under CIRCUIT on the right mouse button menu as shown in Figure 1.2. An *AXIAL* module is a mono-dimensional module needed to describe components such as pipes. This module is shown in Figure 1.3. It is defined by two points called junctions ($J1$ and $J2$). On the line between the two junctions a mesh with sub-meshes is constructed. The point partition is defined in the *XAXIS* panel. In Figure 1.3 the mesh consists of two points: AXI11 and AXI12. Points can be added or deleted with the $+$ or $-$ button. The points define an interval which can be discretized in *Nb meshes* defined in the *MESH* panel. In this panel one can define the direction by setting the *cos* cell. By putting 0,1, -1 in the *cos* cell the line is horizontal, upward vertical or downward vertical respectively. In the *GEOMETRY* panel the section and the dimension of the pipe can be controlled. The corresponding command code generated by GUITHARE are as following

```
AXIAL = AXIAL J1 USTREAM J2 DSTREAM ;
AXIAL11 = XAXIS 0. ;
AXIAL12 = XAXIS 1. ;
LAXIA1 = AXIAL11
SEGMENT 10 AXIAL12 COS 0. ;
```

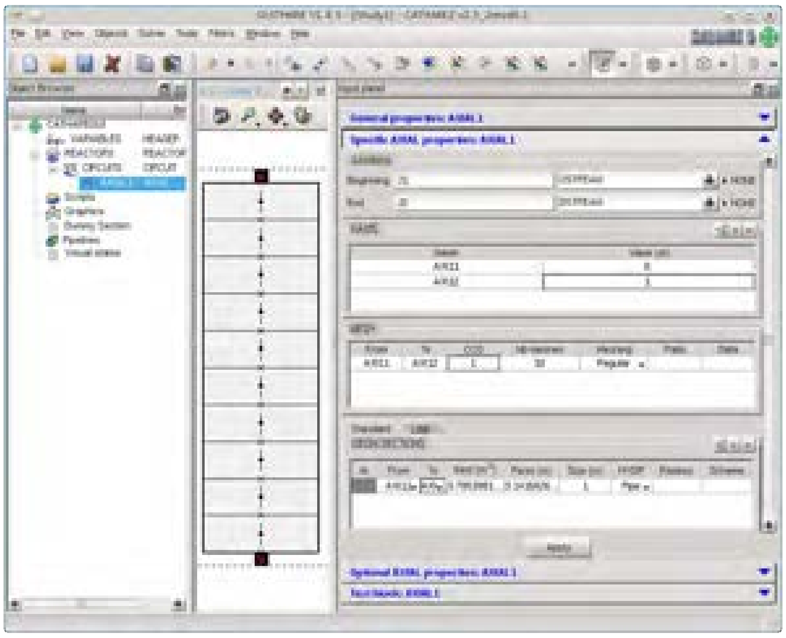


Figure 1.3: Cathare *AXIAL* hydraulic module.

```
MESH AXIAL LAXIA1;
GEOM AXIAL (AXIAL11 AND AXIAL12)
SECT 0.785398 PERI 3.1415926 SIZE 1.;
```

This code defines the *AXIAL* geometric module. The semicolon ends the definition. There are only two points (AXIAL11,AXIAL12) at the mono-dimensional coordinate $x = 0$ and $x = 1$ with one SEGMENT divided into 10 elements in the horizontal direction (cos 0).

A volume module is a zero-dimensional module, which consists of a two-node module used to describe large size plena with several connections, such as pressurizers, steam generator domes and upper and lower plena. The volume module is shown in Figure 1.4. The corresponding code is as follows [3]

```
VOL1 = VOLUME J3 DSTREAM J4 DSTREAM J5 DSTREAM ;
GEOM VOL1
0. 1.
.5 1.
J3 TOP LENGTH .1
SECT 0.007853981633 PERI 0.314159265 SIZE .1 VERTICAL
J4 BOTTOM LENGTH .1
SECT 0.007853981633 PERI 0.314159265 SIZE .1 VERTICAL
J5 BOTTOM LENGTH .1
SECT 0.007853981633 PERI 0.314159265 SIZE .1 VERTICAL;
```

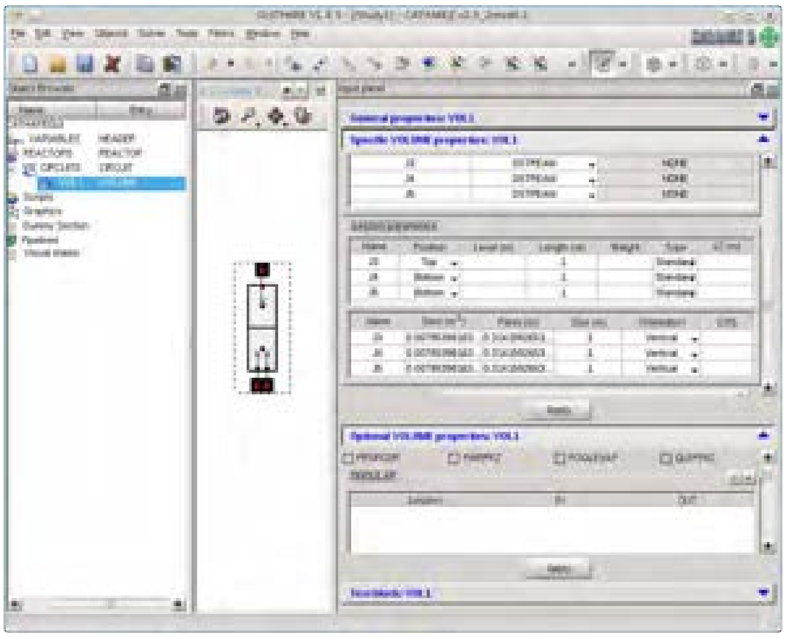


Figure 1.4: Cathare *VOLUME* hydraulic module.

In VOL1 of Figure 1.4 two nodes and three junctions (J3,J4,J5) are defined in the volume and junction panels respectively. For each junctions the geometrical configuration should be defined in the appropriate panel. The junctions can enter into the *VOLUME* horizontally or vertically as specified in the junctions parameter panel of the *VOLUME* module. The BC boundary condition module is used to model the boundary conditions defined by the external components of the reactor or external facilities. Different boundary conditions can be specified. BCONDIT modules may be of two types: internal or external type. In internal type modules variables are defined with respect to time (with the ABTIME directive). In the external type modules variables are not time-dependent but imposed externally (and explicitly) by the user via the input deck. There are three kinds of boundary conditions: inlet, outlet and mixed boundary conditions. For details see [1, 2, 5, 6] The *BCONDIT* module is shown in Figure 1.5. In this Figure a BC3B (internal) boundary condition model is selected. For this boundary conditions an interval of time (REALLIST 0. 100.) must be defined where the state variables should be specified on the boundary. In this case liquid temperature TL is set to (20-20) in the time interval defined by the ABTIME directive. In this case the temperature is fixed to $20C$ during the whole interval of time. In the same way the gas temperature TG and the velocity VL and VG are to be specified. The corresponding code generates by GUITHARE is written as [3, 5]

```
BC = BCONDIT J1 DSTREAM ;
MODEL BC BC3B
TL (REALLIST 20. 20.  )
```

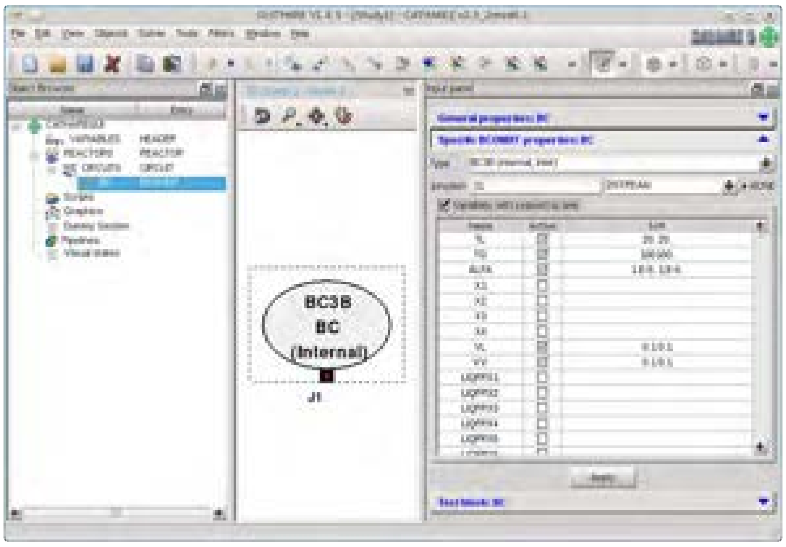


Figure 1.5: Cathare *BCONDIT* hydraulic module.

```
TG (REALLIST 100100.  )
ALFA (REALLIST 1.E-5. 1.E-5.  )
VL (REALLIST 0.1 0.1  )
VV (REALLIST 0.1 0.1  )
ABSTIME (REALLIST 0. 100.  );
```

The *THREED* module attempts to describe multidimensional effects in a vessel.

Together with module there are sub-modules that are selected in an appropriate sub-menus. Each module can have sub-modules. Some popular sub-modules are the *TEE* and the *HEAT EXCHANGER* sub-module The TEE sub-module is used to represent a lateral branch (tee-branch) on a 1D module. Heat exchanger sub-modules can be used between two circuits or between two elements of a circuit [3, 5].

1.2 The Cathare coupling interface on Salome

There is a coupling interface created by the Cathare team. The interface is obtained by creating a c++ wrapper for input and output of the Cathare variables. We simplify this class to obtain a very light a friendly interface on Salome platform. The original wrapper class is called *ICoCo*. For details see [1, 2, 5, 6, 7]. The ICoCo wrapper and a new lighter one, defined through the c++ class *Cathare_problem* are actually present at the same time in this code. The ICoCo wrapper is compiled in the library *libCathare.so* while the class *Cathare_problem* can be used as standard class and compiled together with the main function *main.cpp* (called supervisor). In summary there are two classes that can be used to transfer data from Cathare code to the Salome platform: the *ICoCo* package and the *Problem_Cathare* interface developed in this report.

1.2.1 The *ICoCo* supervisor

The compilation and linking of the *ICoCo* supervisor is not a standard procedure and everything needed for this purpose is written in the Makefile_gad included in the *ICoCo* package. First Cathare and the ICoCo wrapper should be compiled with the environment key $v25_3 set to the Cathare code main directory (. . . /Cathare2/v25_3/mod2.1/). Then one uses the input deck Cathare file **.dat*, creates the final library libCathare_gad.so, uses main.cpp and creates the executable superv with the command

```
DATAFILE=omega.dat make -f $v25_3/ICoCo/Makefile_gad .
```

A datafile used with ICoCo must have certain features in order to be parsed and to allow the control of the time loop by an outside supervisor. The exec block should have a standard look

```
(...)                   <- may include steady state loop
REPEAT BLOCKnn nnn;     <- should be outermost loop
(...)
TIME = TIME + DT;
TRANSIENT CIRCTOT TIME DT;
DT = NEWDT;
TIME = NEWTIME;
(...)
IF ( TIME>TMAX );       <- will be ignored in the supervisor
  QUIT BLOCKnn ;
ENDIF
(...)
END BLOCKnn ;
(...)
```

The supervisor is the C++ file main.cpp. The main.cpp should be written with modular functions and should be easy to read and reuse. The supervisor can be run with the command

```
./superv
```

For details see [7].

1.2.2 The *Problem_Cathare* interface

We are not going to use the *ICoCo* package but mainly the *Problem_Cathare* interface. This interface consists of four basic files with its corresponding headers: *CathareM*, *problem_Cathare*, *SaveCatDataM* and *axial_io*.

The *CathareM.cpp* and *CathareM.h* files contain the c++ wrapping of the FORTRAN main Cathare files. The main functions are

```
bool ecrire(double value, std::string keyword,
                  std::string cname, int imesh, int irad);
bool ecriri(int value, std::string keyword,
                std::string cname, int imesh, int irad);
double value(std::string keyword, std::string cname,
                              int imesh, int irad, int& ivstat);
bool parse_name(std::string name, std::string& keyword,
                  std::string& cname, int& imesh, int& irad);
```

The *ecrire* and *ecriri* functions write the *double* or the *int* variable in the Cathare codes. The variable is defined by four arguments: *std::string keyword* (variable name), *std::string cname*(module name), *int imesh* (mesh element) and *int irad* (angle value). The *parse_name* function is a function that parses the *name* into the four argument *keyword*, *cname*, *imesh* and *irad*. For example if *name*=PRESSURE_AXIAL1_10 then *keyword*=PRESSURE, *cname*=AXIAL1 and *imesh*=10. The *irad* is automatically set to 0. The *ICoCo* library uses the functions through the single *name* argument while the new *Problem_Cathare* class directly by four split arguments.

The *Problem_Cathare* class is the core of the coupling. The main functions of the *Problem_Cathare* class are shown in Table 1.1. We remark that the function

```
getValue_Cathare(string keyword,string cname,int ix, int irad,
                                                   double val)
```

and

```
setValue_Cathare(string keyword,string cname,int ix, int irad)
```

are not contained in the *ICoCo* interface. The *setValue_Cathare* function has four input arguments: *keyword*, *cname*, *imesh* and *irad*. The *getValue_Cathare* function has the four previous mentioned input arguments and one output argument: the output variable is the value read from Cathare. The TrioField variable type is not present in this class while it is the standard type for exchange variable in *ICoCo* interface. The exchange between the Cathare code and the supervisor is done by using simple double type variables.

Cathare code needs to store and retrieve data. In FORTRAN programs the data are inside the *common block data*. The *SaveCatData* is the structure used for saving/restoring the FORTRAN commons (Cathare and Pilot) by using the functions in the *SaveProblemData* class. This class saves only the used portion of the commons and the variables of a *Problem_Cathare* itself. The data structure *SaveCatData* is described in the Table 1.2 where the FORTRAN commons and all the possible variables of a Problem are written. *SaveCatData* class consists of two main functions: save and restore function as shown in Table 1.3.

The *Problem_Cathare* interface includes also the files *axial.cpp* and *axial.h*, that include the class that allows to print in XDMF format. In this way the Cathare solution

return type	function
	start/stop program
bool	initialize()
void	terminate()
void	setDataFile(·);
	interface time step
double	presentTime() const;
double	computeTimeStep(·) const;
bool	initTimeStep(·);
bool	solveTimeStep();
void	validateTimeStep();
bool	isStationary() const;
void	abortTimeStep();
	interface saving
void	save(·,·) const;
void	forget(·,·) const;
void	restore(·,·,·);
	interface FieldIO
vector< *string* >	getInputFieldsNames() const;
void	getInputFieldTemplate(·,·,·,·) const;
void	setInputField(·,·);
vector< *string* >	getOutputFieldsNames() const;
void	getOutputField(·,·) const;
Problem_Cathare class only	**interface FieldIO**
double	getValue_Cathare (·,·,·,·)
void	setValue_Cathare (·,·,·,·,·)

Table 1.1: *Problem_Cathare* class.

can be seen at any time together with the three-dimensional one on the Paraview visualizer. The variables that should be printed and visualized for each axial are defined in a vector of strings: the *varNames* vector. For example if one would like to see the values of the Cathare variables ALFA, GASTEMP, LIQFLOW, LIQH, LIQTEMP, PRESSURE, LIQV one must define the static vector at the top of the file *axial.cpp* as

```
const static std::vector<std::string> varNames= {
  "ALFA",
  "GASTEMP",
  "LIQFLOW",
  "LIQH",
  "LIQTEMP",
  "PRESSURE",
  "LIQV"
```

return	function	description
void	fill_sizes();	
void	save_Cathare()	save Cathare commons
void	save_Pilot();	save here the Pilot commons
void	restore_Cathare();	restore from here the Cathare commons
void	restore_Pilot();	restore from here the Pilot commons
int	Cathare_dsize;	
double*	Cathare_doubles;	
int	Cathare_isize;	
int*	Cathare_ints;	COMMON DIMENSION
int	Cathare_csize;	
char*	Cathare_chars;	
int	Cathare_psize;	
void**	Cathare_pointers;	
int	Pilot_dsize;	
double*	Pilot_doubles;	
int	Pilot_isize;	
int*	Pilot_ints;	

Table 1.2: *SaveCatData* class is the structure used to contain the data commons and the variables of a Problem_Cathare.

return	function	description
void	save (const Problem_Cathare& what)	save data
void	restore (Problem_Cathare& where)	restore data

Table 1.3: *SaveCatData* class functions to extract and store Cathare variables.

```
};
```

Each axial is defined by using the structure *AxialData* as reported in Table 1.4. The

type	data	description
string	name;	component name
int	nCells;	total number of elements
vector<double>	blockCoord;	coordinates
vector<uint>	blockN;	elments in each block
vector<double>	cos;	directions

Table 1.4: *Axial* data structure.

Axial class contains an AxialData class structure (called *_data*) a *Problem_Cathare* class pointer (called *_Cathare*) and an *AxialWriteFlag* (*_writeFlag*). The *_data* contains the axial data as in Table 1.4 while the *Problem_Cathare* class pointer allows the data exchange with a FORTRAN code itself. The boolean flag *_writeFlag* set the print in XDMF-HDF5 or VTK format for Paraview visualization [36, 32]. The functions of the *Axial* class are defined in Table 1.5. For each *AXIAL* the *Axial* class reads from the

type	function
void	setName(const std::string name)
void	setNNodes(const int nCells)
void	write(double const time,int const step)
vector<vector<double> >	_getData();
void	_writeHDF5Sequence(·,·,·);
void	writeXDMF(·, ·);
void	writeHDF5(·, ·);
void	writeHDF5mesh() const;
void	writeVTK(·, ·);
void	writeVTKSequence();

Table 1.5: *Axial* class functions.

input Cathare deck file its data and fill *Axialdata* vector. With reference to the following portion of code

```
AXIAL1 = AXIAL1 J1 USTREAM J2 DSTREAM ;
AXIAL11 = XAXIS 0. ;
AXIAL12 = XAXIS 1. ;
LAXIA1 = AXIAL11
SEGMENT 10 AXIAL12 COS 0. ;
MESH AXIAL1 LAXIA1;
GEOM AXIAL1 (AXIAL11 AND AXIAL12)
SECT 0.785398 PERI 3.1415926 SIZE 1.;
```

in the *Axialdata* vector we have the module name *name=AXIAL1*, the total number of elements *nCells*=10, the coordinate vector *blockCoord*=0,0,0,1, *elements in each block* blockN=10; *direction vector* cos=0.

1.3 Cathare post-processing on Salome

There are three ways to see the Cathare results on Salome: the standard Cathare text output, the graphical GUITHARE and on the Salome platform by using Paraview with XDMF-HDF5 format [36, 32, 4].

1.3.1 Cathare text output

The post-processing of the results can use the graphic input data deck and the binary file FORT21 to display the data. The post-processing of results is performed afterward by *postpro.unix*. The command has the following format

```
postpro.unix test-caseg.dat
```

where *test-caseg.dat* is the name of the input deck file. The *postpro.exe* reads the file DICO and generates the ASCII evolution file FORT07 from the result file FORT21 on the basis of user selected variables [1, 2, 4]. If one wants to create and use a local *postpro.exe* instead of the standard one, one must add the second argument Âń mask Âż in *postpro.unix* procedure used.

1.3.2 GUITHARE with FORT21 output file

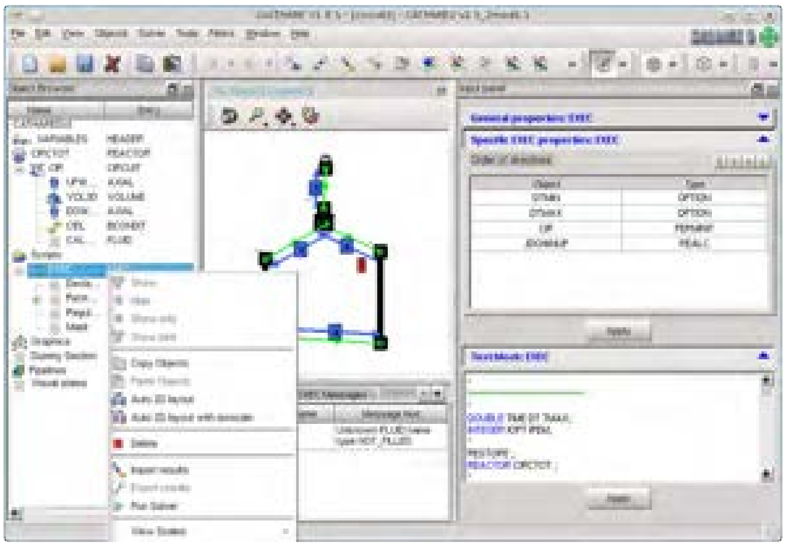


Figure 1.6: GUITHARE import result window.

Post-processing can use a graphic input data deck and the binary file FORT21. In Figure 1.6 the FORT21 is loaded by using the menu of the *exec* menu. The GUITHARE graphical mode is then activated, as in Figure 1.7, and finally the graph is displayed (Figure 1.8) [4].

1.3.3 Paraview XDMF-HDF5

The new *Problem_Cathare* class generates the mesh file *AXIAL_mesh.h5* that contains the coordinates and the connectivity structure. Also the *AXIAL.time.h5* and its driver in XDMF format *AXIAL.time.xmf* are generated. The file in XDMF format *AXIAL.time.FEMuS* can be opened by using the Paraview application as shown in Figure 1.9 [36, 32]. The great advantage of this approach is that the solution fields can be seen during the evolution and not only at the end of the computations.

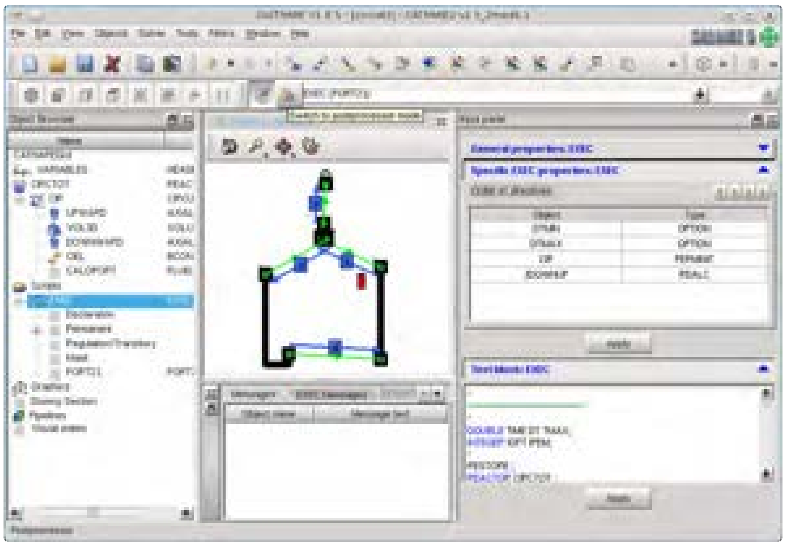


Figure 1.7: GUITHARE graphical mode button.

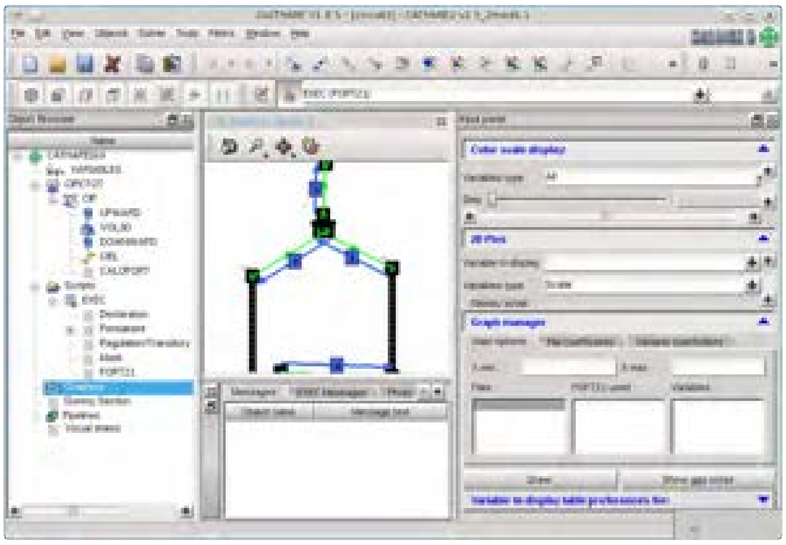


Figure 1.8: Plotting result window.

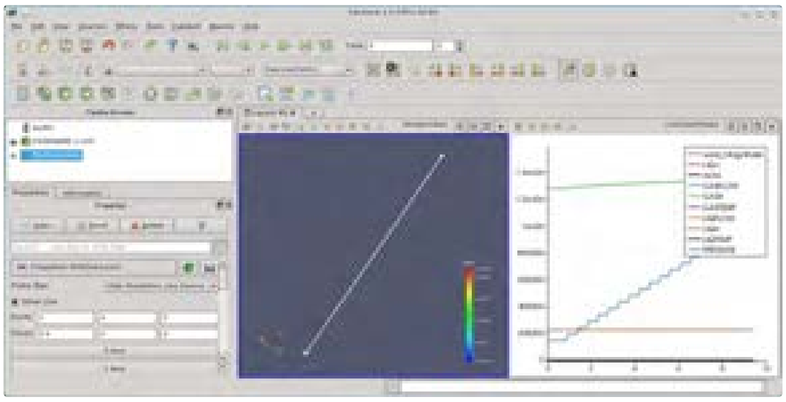


Figure 1.9: Paraview with results in XDMF format.

2 Implementation of FemLCore on Salome Platform

2.1 SALOME-FemLCore interface

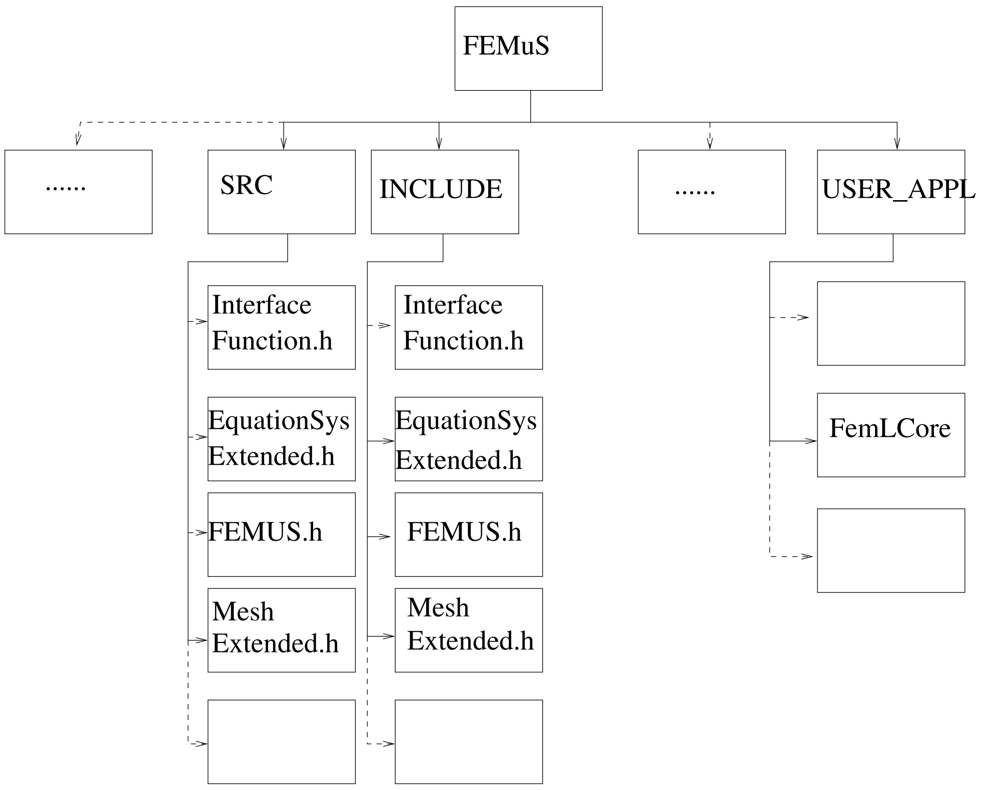


Figure 2.1: Diagram of the *FEMuS* interface class to the Salome Platform.

The FemLCore code implement modules aimed at computation of 3D core thermal-hydraulics of fast nuclear reactors [13, 14, 15, 16]. As shown in Figure 2.1 the FEMuS interface to the Salome platform is organized in four files or classes: *FEMuS*, *EquationSystemsExtendedM*, *ParallelMeshExtendedM* and *LibMeshFunctionM*. All these classes are located in the *src* FEMuS directories. The corresponding headers are located in the *include* FEMuS directory. The FEMuS interface allows to pass commands from MEDMem library to the *EquationSystemsExtendedM* class which is the problem solver function and viceversa [17, 18, 19]. The *FEMuS* class is a c++ class that can com-

Type	name data	description
MGFEMuSInit *	_start;	start function
MPI_Comm	_comm;	communicator
bool	_local_MPI_Init	initial mpi flag
MGUtils *	_mg_utils	param and files
MGFEMap *	_mg_femap	fem map
MeshExtended*	_mg_mesh	FEMuS-mesh
MEDCouplingUMesh *	_med_mesh	Med-mesh
EquationSystemsExtendedM*	_mg_equations_map	system
MGTimeLoop *	_mg_time_loop	transient

Table 2.1: FEMuS data structure.

Type	name data	description
	FEMUS()	constructor
void	terminate()	end
void	init_param(·)	init param
void	init_fem(·)	init fem
void	setMesh()	set mesh
void	setSystem(·,·,·)	set system
const MeshExtended &	get_MGMesh()	get GMesh
const EquationSystExtM&	get_MGExtSystem()	get system
void	solve_setup(·,·)	initial setup
void	solve_onestep(·,·,·,·,·)	transient
void	solve_non_linear_onestep(·,·,·,·,·)	transient
void	solve_control_onestep(·,·,·,·,·)	one time step

Table 2.2: FEMuS interface functions to FemLCore code.

municate data into and from the FemLCore code. It contains data and functions that are show in Table 2.1 and Table 2.3. In Table 2.1 there is the list of data, mainly pointers needed to transfer data from one mesh format to another, namely MGMesh to MED format. The pointers *_mg_mesh* and *_med_mesh* refer to the mesh in their respective MGMesh and MED format. The *FEMuS* class needs information to extract data from these different mesh structures. In particular, the classes *MGUtils*, *MGGeomEL*, *MGFEMap* and *MGFemusinit* must be available inside the FEMUS interface to transfer basic information such as file names, parameters, multigrid level, fem elements, etc. The *MGUtils* contains all the file names and input parameter values. The *MGGeomEL* and *MGFEMap* contain information about the finite element mesh structure and *MGFemusinit* class is the manager class for MPI and multigrid solvers [18, 19].

The *FEMuS* class contains the FemLCore commands and the SALOME platform data structure. Each code interface implemented on Salome platform must have similar

Type	name data	description
void	setMedMesh(·)	set Med mesh
void	init_interface(·,·,·,·,·,·)	init interface
void	init_interface(·,·,·,·,·,·,·)	init interface
void	setAnalyticSource(·,·,·)	set anal field
void	setFieldSource(·,·,·)	set num field
void	write_Boundary_value(·,·,·,·,·)	set bd value
void	setAnalyticBoundaryValues(·,·,·)	set anal bd value
double	getAvOnBoundary_nodes(·,·,·)	get av
MEDCouplingFieldDouble *	getValuesOnBoundary(·,·,·,·)	get field

Table 2.3: FEMuS interface functions to Salome platform.

commands to run the code from the same supervisor program *main.cpp*. As shown in Table 2.2 the functions that start and stop the code are the constructor *FEMUS(MPI_Comm comm)* and the function *terminate()*. These functions take into account the start and stop of all MPI parallel processes of the code. The initialization of the parameters and the FEM elements for the FemLCore project are obtained by the following functions

```
void init_param(MGUtils &mgutils);
void init_fem(MGGeomEl & mggeomel,MGFEMap & mgfemap);
```

The system and the mesh can be set by the following three functions

```
void setSystem(const std::vector<NS_FIELDS> & pbName);
void setMesh();
void setMedMesh(const std::string & meshFile);
```

The type of the problem is defined by using the members of the *NS_FIELDS* enumeration shown in Table 2.4 [18, 19]. For example, a vector pbName defined as pb-

name	number	fem order	description
NS_F	0	2–1	Navier-Stokes
NSX_F	0	2	x-Navier-Stokes
NSY_F	1	2	y-Navier-Stokes
NSZ_F	2	2	z-Navier-Stokes
P_F	3	1	projection Pressure
T_F	4	2	Temperature
K_F	5	2	Turbulence K
EW_F	6	2	Turbulence W

Table 2.4: NS_FIELDS enumeration structure.

Name[0]=NS_F, pbName[1]=T_F defines a problem which consists of the Navier-Stokes and energy equations. The last functions in Table 2.2 are used to assemble and solve the system. We have

```
void solve ()
void solve_setup (int &t0, double &time)
void  solve_onestep (const int &t0, const int &t_step,
                const int &print_step, double &time, double &Dt)
```

The *solve_setup* initializes system at $t = 0$. The *solve_onestep* function controls a single time step of the problem while *solve()* can be used to solve more time steps at once. The Table 2.2 contains functions that run the FemLCore code. The Table 2.3 contains functions that can handle data in and out from the Salome platform. All these functions need the MedMem libraries. In order to exchange the data first we have the data exchange interface. It must be initialized before one can use it. For this reason we have different interface initialization functions

```
void init_interface(
  const int interface_name,         ///< label name
  int interface_id,                 ///< Med-MG group name (
  int order_cmp,                    ///<  order
  const std::string & medfile_name, ///< medfile name
  bool on_nodes=true,               ///< nodes or elements
  const int index_medmesh=0         ///< med-mesh index
  ) ;
void init_interface(
   const int interface_name,        ///< label name
  int interface_id,                 ///< Med-MG group name
  int order_cmp,                    ///<  order (in)
  const std::string & medfile_name, ///< medfile name
  bool on_nodes=true,               ///< nodes or elements
  const int index_medmesh=0         ///< med-mesh index
  ) ;
```

Usually one can use a line as

```
 init_interface(101,100,2,"file_med_mesh")
```

to initialize the interface, where 102 is the name on the interface (or id name), 100 is the group mesh that characterizes this surface geometry, 2 is the order of interpolation needed (in this case quadratic) and *file_med_mesh* is the name of the med file that contains the mesh.

The interfaces can be adapted to volume and surfaces. For volumes we have

```
void setFieldSource (const MEDCouplingFieldDouble *f)
void setAnalyticSource (const std::string &f)
```

The first function sets the numerical *MEDCoupling* field into the volume interface while the second transforms the analytical expression before it passes the field. The most common data transfer interfaces are boundary surfaces. The boundary should be controlled for input and output. For input we have the following interface functions

```
void write_Boundary_value(
    int id_boundary_name,
    std::string mgsystem_name,
    int n_cmp,
    int first_cmp =0
);
void setAnalyticBoundaryValues(
    int interface_name,
    int n_cmp,
    const std::string & f
);
void setFieldBoundaryValues(
    int interface_name,
    int n_cmp,
    const ParaMEDMEM::MEDCouplingFieldDouble * bcField
);
```

The most useful function is the *write_Boundary_value* since this function write the values stored in the interface *id_boundary_name* directly in the *x_old* solution vector. In this case one does not need to pass through the assembly function. The solution values can be transferred only on nodes. For output we have

```
double getAvOnBoundary_nodes(
  int  name,
  const std::string &system_name,
  int  first_cmp
);
  ParaMEDMEM::MEDCouplingFieldDouble *
                                getValuesOnBoundary(
    int interface_name,
    const std::string & systemName,
    int n_cmp,
    int first_cmp=0
  );
```

The first one returns the average value on the surface which is usually what one wants on 2D/1D interface surfaces.

The transfer of data is based on the *InterfaceFunctionM* class. In Table 2.5 there are the data to be transported in/out the Salome platform and in Table 2.6 the function that allows to perform such an exchange. The *_mesh_mg* pointer points to the mesh

type	name	description
const MeshExtended *	_mesh_mg	MGMesh
MEDCouplingUMesh *	_support_med	MEDCoupling mesh
MEDCouplingFieldDouble *	_field	field
int	_n	nodes
int *	_map_med	map [i]− >MGMesh
int *	_map_mg	map [i]− >MEDCoupling

Table 2.5: *InterfaceFunctionM* data structure.

type	name	description
void	eval(·, ·, ·)	evaluation
int	get_n()	get dim
int *	get_map_mg()	get MG map
int *	get_map_med()	get med map
MEDCouplingFieldDouble *	getField(·)	get field
MEDCouplingUMesh *	getSupport()	get interf mesh

Table 2.6: *InterfaceFunctionM* functions.

in FEMuS format while *_support_med* contains the mesh in MEDMem format. The solution of the internal or external problem in MEDMem format are stored in the field *_field*. The *_field* pointer may refer to the part of the mesh (*_support_med*) where the data are to be transferred. In fact if the function is associated with a part of the boundary, the MED mesh and the solution mus be relative only to this part. The *_map_mg* and *_map_med* are the maps to the FEMuS to the MEDMem mesh format for the nodes, faces and elements respectively. The function of the *InterfaceFunctionM* class is used to evaluate the field and maps the node and element connectivity between coupling codes. Each code must have a MEDMem interface in order to exchange data through the supervisor *main.cpp* during execution.

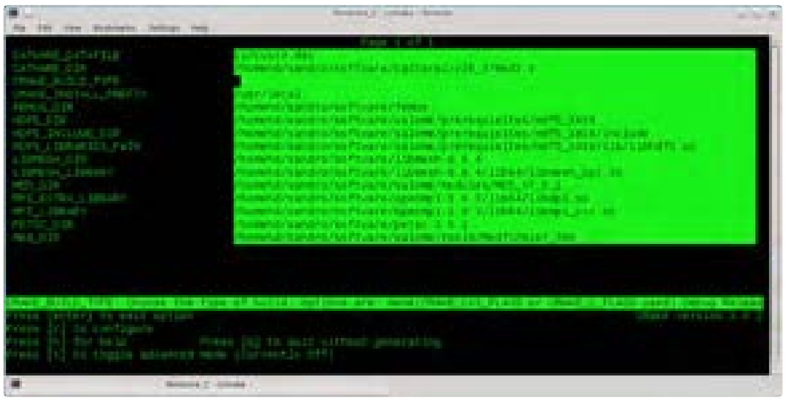
2.2 Compiling and running the coupling modules

The compiling and linking all the coupling modules together with the supervisor program is a complex task. In order to facilitate that we create the makefile by using CMake software. CMake allows us to control the library and include paths. We can start such graphical setting with

```
 ccmake  ./SRC
```

The Figure 2.2 shows this graphical environment. We start with

```
cmake_minimum_required(VERSION 2.8)
project(ICoCoTest CXX Fortran)
```



```
[illegible]
```

Figure 2.2: ccmake environment.

name	open-source	description
MPI	y	Parallel lib
PETSC	y	solver lib
HDF5	y	data compression lib
med	y	mesh and data file format
MED	y	coupling lib
FEMuS	y	FemLCore lib
CATHARE	n	Cathare lib

Table 2.7: Project library dependence.

include	lib	description
$CMAKE_SOURCE_DIR	project	./SRC
$CMAKE_BINARY_DIR/DATA	project	./DATA
$CMAKE_SOURCE_DIR/cathare_problem	CATHARE	c++ wrapper
$LIBMESH_INCLUDE_DIRS	LIBMESH	FindLIBMESH.cmake
$HDF5_C_INCLUDE_DIR	HDF5	FindHDF5.cmake
$PETSC_INCLUDE_DIRS	PETSC	FindPETSC.cmake
$MED_INCLUDE_DIRS med	MED	FindMED.cmake
$med_INCLUDE_DIRS	med	Findmed.cmake
$FEMUS_INCLUDE_DIRS	FEMUS	FindFEMUS.cmake

Table 2.8: Project include files.

set the c++ compiler

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}
                          -Wall -Wpedantic -g -std=c++11")
```

and set CMAKE_MODULE_PATH where the cmake module instruction are kept

```
 set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} "$
                              {CMAKE_SOURCE_DIR}/cmake")
```

For this code we need the MPI, PETSC, HDF5, LIBMESH, med, MED, FEMuS and Cathare libraries [30, 31, 33, 35, 38, 34, 29, 1, 3]. In Table 2.7 there is a brief description of these needed libraries.

The *include_directories* directive lists the include directories as

```
include_directories(
 lists in Table
)
```

Now we should generate two dynamic libraries: Cathare_DATAFILE.so and femus.so.

a. **Construction of the library Cathare_DATAFILE.so**

The source files are set by

```
file(GLOB_RECURSE Cathare_problem
        ${CMAKE_SOURCE_DIR}/Cathare_problem/*.cpp)
```

We plan to generate a dynamic library with the name of the input deck file. The file dat for the geometry and data in introduced as

```
set(Cathare_DATAFILE "circuit3.dat"
                     CACHE FILEPATH "Cathare input file")
string(REGEX REPLACE "\\.dat$"
               "" Cathare_DATANAME ${Cathare_DATAFILE})
```

The dynamic library for Cathare (library ICOCO_LIB) and the pilot file are set

```
set(ICOCO_LIB "${Cathare_DATANAME}.so")
if(Cathare_VERSION STREQUAL v25_3)
  set(ICOCO_PILOT "PILOT_${Cathare_DATANAME}.f")
elseif(Cathare_VERSION STREQUAL v25_2)
  set(ICOCO_PILOT "PILOT.f")
endif()
```

Once the dynamic library with Cathare source code is defined together with the include directories we can start to generate the Cathare environment. The procedure consists of several steps:

a1. **Creating links in the main directory.** The command that makes the links is

```
  $v25_3/unix-procedur/vers.unix
```

where $v25_3 is the environment variable pointing in the working the directory set by export command. In cmake language this is obtained by

```
include_directories(${Cathare_DIR}/lib)
include_directories(${Cathare_DIR}/ICoCo/API)
```

```
add_custom_command(
  OUTPUT CATHAR.f cathar_omp.unix
     cathar.unix DICO FASTSIZE.f
     incline.unix postpro.unix read.unix _version
  COMMAND ${Cathare_DIR}/unix-procedur/vers.unix
)
```

The *include_directories* command adds the binary tree to the search path for include files while the *add_custom_command* creating links in the main directory.

a2. **Creating reader file.** The command that creates reader file is

```
$v25_3/unix-procedur/vers.unix reader
```

The cmake commands to write are as follows

```
set(reader_files reader/DICO
         reader/READER.f reader/_version)
add_custom_target(clean_Cathare
  COMMAND ${CMAKE_COMMAND} -E remove_directory reader
)
add_custom_command(
  OUTPUT reader/ ${reader_files}
  COMMAND ${CMAKE_COMMAND} -E make_directory reader
  COMMAND ${Cathare_DIR}/unix-procedur/vers.unix reader
  DEPENDS ${CMAKE_BINARY_DIR}/cathar.unix
  COMMENT "creating reader files..."
)
```

a3. **Adding ICoCo mask reader files.** We need to add the reader file to the (ICOCO_SRC_MASK) variable. The command to be used is the same as before so we can write

```
file(GLOB mask_reader_files
       ${Cathare\_DIR}/ICoCo/mask_reader/*)
foreach(file IN LISTS mask_reader_files)
  get_filename_component(filename ${file} NAME)
  add_custom_command(
    OUTPUT reader/${filename}
    COMMAND ${CMAKE_COMMAND} -E
             create_symlink ${file} reader/${filename}
    DEPENDS reader/DICO reader/READER.f reader/_version
    COMMENT "adding ICoCo mask reader files..."
  )
  list(APPEND reader_files reader/${filename})
endforeach()
```

```
set(ICOCO_SRC_MASK FASTSIZE.f)
```

a4. **Running Cathare preprocessor.** The command that creates the reader file is

```
.${CMAKE_SOURCE_DIR}/${Cathare_DATAFILE}
                                      mask | tee reader.out
```

therefore we write

```
${CMAKE_SOURCE_DIR}/${Cathare_DATAFILE}
                                   mask | tee reader.out
add_custom_command(
  OUTPUT FAST.H C2.INIT ${ICOCO_PILOT}
                          reader.out reader/reader.exe
  COMMAND ${CMAKE_COMMAND} -E remove -f PILOT.f
  COMMAND ./read.unix ${CMAKE_SOURCE_DIR}/
                 ${Cathare_DATAFILE} mask | tee reader.out
  COMMAND ${CMAKE_COMMAND} -E
                   rename PILOT.f ${ICOCO_PILOT}
  DEPENDS read.unix ${reader_files}
          ${CMAKE_SOURCE_DIR}/${Cathare_DATAFILE}
  COMMENT "running Cathare preprocessor..."
)
```

a5. **Generating source files.** The command that generates source files is

```
  ${Cathare_DIR}/ICoCo/bin/reader2 ${ICOCO_PILOT}
```

therefore

```
if(Cathare_VERSION STREQUAL v25_3)
  set(ICOCO_SRC_GAD INIGAD.f MANAGAD1.f MANAGAD2.f
            TERMIGAD.f SAVEGAD.f RESTGAD.f SETPREST.f)
elseif(Cathare_VERSION STREQUAL v25_2)
  set(ICOCO_SRC_GAD INIGAD.f MANAGAD1.f
                                      MANAGAD2.f TERMIGAD.f)
endif()
add_custom_command(
  OUTPUT ${ICOCO_SRC_GAD} MANAGAD.H
  COMMAND ${Cathare_DIR}/ICoCo/bin/reader2 ${ICOCO_PILOT}
  DEPENDS ${ICOCO_PILOT}
)
```

a6. **Generating the library Cathare_DATANAME.so**

The library $Cathare_DATANAME.so is generated from $ICOCO_SRC_GAD and $ICOCO_SRC_MASK. Therefore first we define $ICOCO_SRC_GAD and $ICOCO_SRC_MASK as

```
 add_library(${Cathare_DATANAME}
   SHARED
   ${ICOCO_SRC_GAD} ${ICOCO_SRC_MASK}
)
```

and then $Cathare_DATANAME.so as

```
# add  libraries for ${Cathare_DATANAME}.so
target_link_libraries(${Cathare_DATANAME}
   m     # system libm.so
 ${Cathare_DIR}/ICoCo/lib/libCathare_base.so
 ${Cathare_DIR}/ICoCo/lib/libCathare.so
   )
```

b. Generating the library femus.so

First we must collect the file to put in the library from the main FEMuS and coupling class directories (src, contrib/matrix, SRC/cathare_problem)

```
file(GLOB_RECURSE FEMUS_src ${FEMUS_src_DIR}/*.C)
file(GLOB_RECURSE FEMUS_matrix
                             ${FEMUS_CONTRIB_DIR}/matrix/*.C)
file(GLOB_RECURSE CATHARE_problem
         ${CMAKE_SOURCE_DIR}/cathare_problem/*.cpp)
```

The sheared library can be build as

```
add_library(femus
  SHARED
  ${FEMUS_src} ${FEMUS_matrix} ${CATHARE_problem}
)
```

The supervisor main.cpp can be compiled with the following cmake directive

```
add_executable(main
    main.cpp
    list of file in  supervisor in Table
)
```

where the list of files can be found in Table 2.9. The supervisor is linked with the libraries in Table 2.10 as

```
 target_link_libraries(main
  list of libraries to link in Table
)
```

name	type	description
main.cpp	directory	supervisor
MGSolverNS.C	SRC	Navier-Stokes
MGSolverT.C	SRC	Temperature
User_NS.C	SRC	Bc Navier-Stokes
User_T.C	SRC	Bc Temperature
ReactData.C	SRC	interface N
axial_io.cpp	cathare_problem	graphical
CathareM.cpp	cathare_problem	interface C
Problem_cathare.cpp	cathare_problem	interface C
SaveCatDataM.cpp	cathare_problem	interface C

Table 2.9: Table of files to include in the supervisor.

name	library	description
$MPI_CXX_LIBRARIES	MPI	parallel
$PETSC_LIBRARIES	PETSC	algebra
$HDF5_hdf5_LIBRARY_RELEASE	HDF5	data store
$HDF5_LIBRARIES	HDF5	data store
$MED_LIBRARIES	MED	coupling
gfortran	GFORTRAN	fortran
femus	FEMUS	FemLCore
$CATHARE_DATANAME	CATHARE	cathare

Table 2.10: Table of linked libraries.

3 Numerical results for FemLCore-Cathare coupling

3.1 Introduction

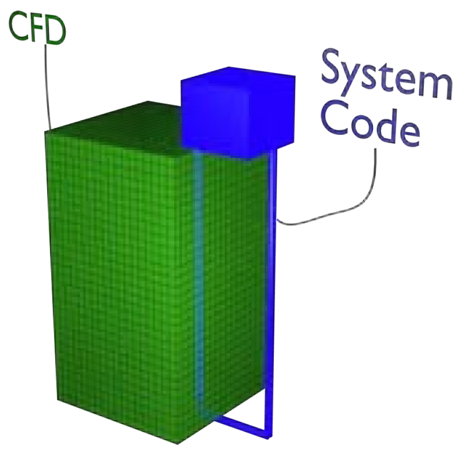


Figure 3.1: Sketch of the FEMuS(3D)-Cathare(1D) coupling.

In this section we discuss the FemLCore-Cathare coupling. In the previous sections we have briefly introduced the c++ interfaces of the Cathare and the FemLCore to the common platform. Any other code with an appropriate interface can interchange data with this thermohydraulics system. For example we remark that a three-dimensional neutronics code can couple to FemLCore-Cathare system to give the appropriate heat volume source to the 3D-porous model of the reactor. Since in the present analysis we are not interested in the neutronics coupling we have used a neutronics algebraic module that store static values of the heat sources through a MedMem c++ interface. In a similar way a simplified volume interface for pressure losses is introduced to define its values for the 3D-porous model of the reactor. Both interfaces may be coupled with more realistic three-dimensional codes.

In this section we illustrate how this coupling can be implemented in a c++ code when

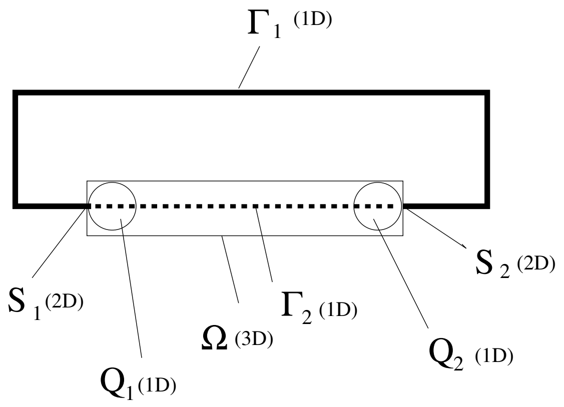


Figure 3.2: Sketch of the defective method approach for numerical coupling.

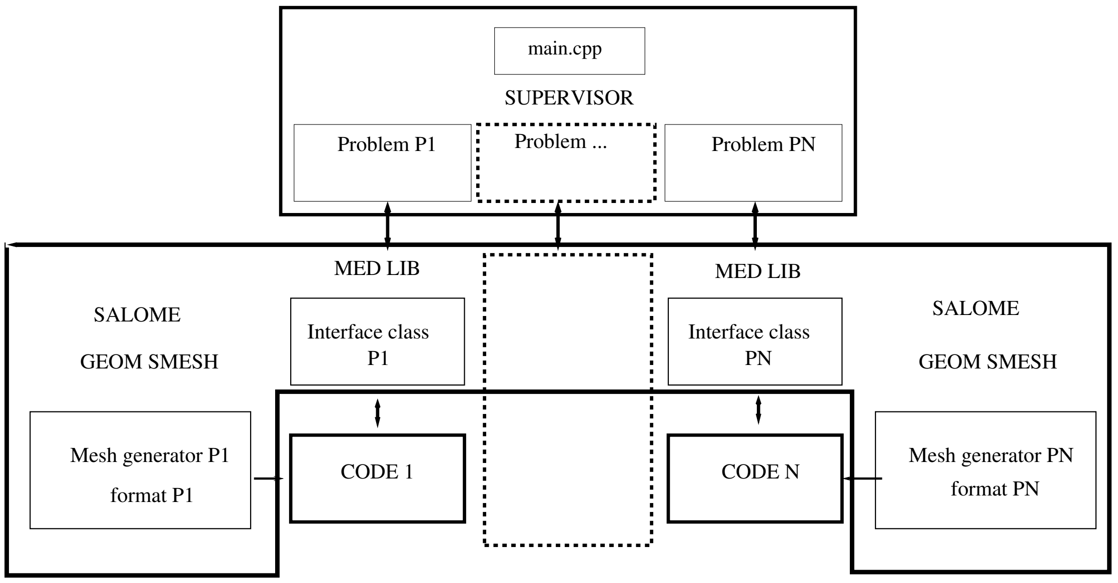


Figure 3.3: Sketch of the numerical coupling approach on Salome platform.

each code is available in the form of C++ dynamic library. For Cathare we assume that Cathare is contained in the *Cathare_base.so* dynamic library while FemLCore is in the *femus.so* dynamic library.

The coupling is geometrically defined in Figure 3.1. The three-dimensional domain Ω, which consists of the reactor plena and core, is connect with a mono-dimensional domain Γ that represents the primary loop. The primary loop consists of almost two mono-dimensional pieces: one external to the reactor (Γ_1) and one internal (Γ_2). The mesh of internal mono-dimensional part Γ_2 overlaps over the three-dimensional domain Ω to close the mono-dimensional circuit $\Gamma_1 \cup \Gamma_2 = \Gamma$. In the primary loop Γ_1 one can find the pump and the heat exchanger. In the junction between the external Γ_2 and internal

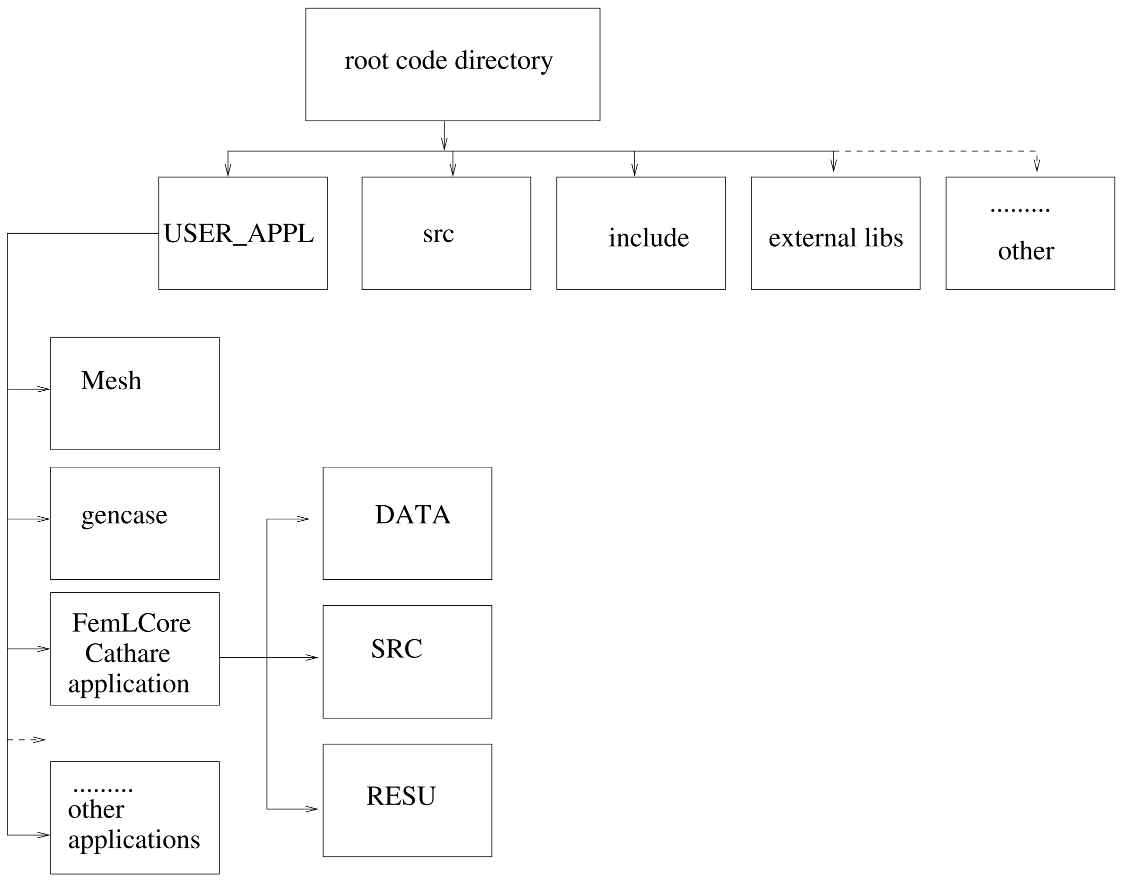


Figure 3.4: Project directory tree.

Γ_1 mono-dimensional mesh there are two surface interfaces S_1 and S_2 for the 3D and 1D modulus connection. One volume interface V_1, needed for the heat and pressure sources, is set in the core of the three-dimensional domain Ω.

The numerical coupling method adopted here, and shown in Figure 3.2, can be classified as a defective method. The mono-dimensional system equation is solved along the entire domain over a mono-dimensional circuit Γ. The mono-dimensional circuit gives the boundary conditions (over S_1 and S_2) to the three-dimensional problem and the three-dimensional code corrects the system solutions through the appropriate sources Q_1 and Q_2 in the momentum and energy equations. The sources, located in the overlapping region Γ_2, are feedback control devices that increase or decrease the source intensity based on the variable state matching at the 1D-3D interface S_1 and S_2.

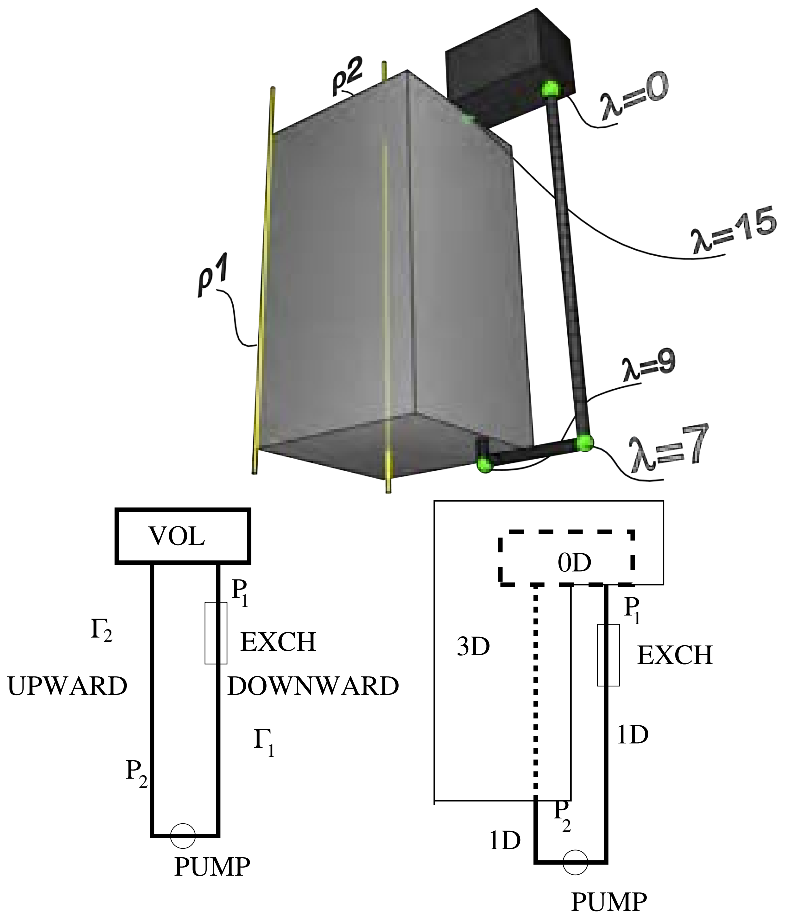


Figure 3.5: Test 1. Sketch of the mono-dimensional circuit (on the right) and the coupling 3D-1D dimensional domain geometry (on the left).

3.2 Test 1 for FemLCore-Cathare coupling

3.2.1 General description

In this test we couple a simplified geometry of a model reactor described by the 3D-porous FemLCore porous module with a mono-dimensional primary loop defined through an input deck of the Cathare code. In Figure 3.5 a sketch of the coupling between the mono and three-dimensional domain geometry is shown. The three-dimensional domain Ω represents the whole reactor. Ω consists of a rectangular parallelepiped of dimension 3.6m $\times$3.6m $\times$6.6m wrapped by six surfaces that defined the boundary $\partial\Omega$. The bottom and the top are considered the inlet and the outlet of the reactor, respectively. The mono-dimensional domain Γ, which represents the primary loop consists of three pieces: a point volume (VOL), Γ_2 (UPWARD) and Γ_1 (DOWNWARD). The point volume VOL and the mesh of the UPWARD axial branch Γ_2 overlap with the the three-dimensional region Ω.

Γ_1 exits from the reactor Ω and moves downward then horizontally and finally vertically to enter again in the tree-dimensional region Ω. In the mono-dimensional circuit the fluid exits from the point volume VOL enters the axial DOWNWARD, flows into the UPWARD branch and returns to the volume point VOL. In the three-dimensional region Ω the the fluid flows from the bottom to the top.

3.2.2 FemLCore 3D mesh and geometry

First we build the geometry and the mesh for the three-dimensional domain Ω. Since we plan to perform fluid dynamic computations with the Finite Element method the mesh should be constructed with quadratic HEX27 elements The FEM basis are of the Taylor-Hood type: the velocity field is quadratic while the pressure fields is supposed to be linear.

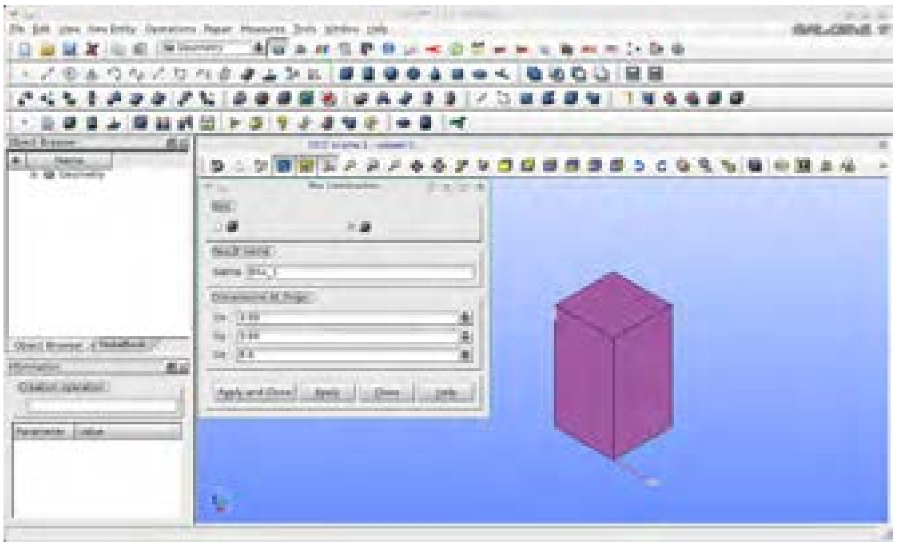


Figure 3.6: Test 1. The simplified three-dimensional reactor geometry with the GEOM module.

The project is developed under the Salome platform and therefore we can use the appropriate module to complete this task and generate the mesh in MED format. The FemLCore code is integrated with the Salome platform and can read input mesh in this format. We open the platform and select the GEOM module in order to draw the required geometry. As shown in Figure 3.6 we draw a rectangular parallelepiped geometry $3.6 \times 3.6 \times 6.6$. The FemLCore needs to mark the surfaces where special operations should be performed. We have boundary conditions and coupling interfaces of 3D-1D, 1D-1D, 3D-3D type. In Figure 3.7 we create the geometry groups for boundary conditions and coupling interfaces. The *create group* function of the GEOM module is used. Then we open the SMESH module of the Salome Platform and create the mesh. As shown in Figure 3.8 the hypothesis construction, based on simple uniform subdivisions,

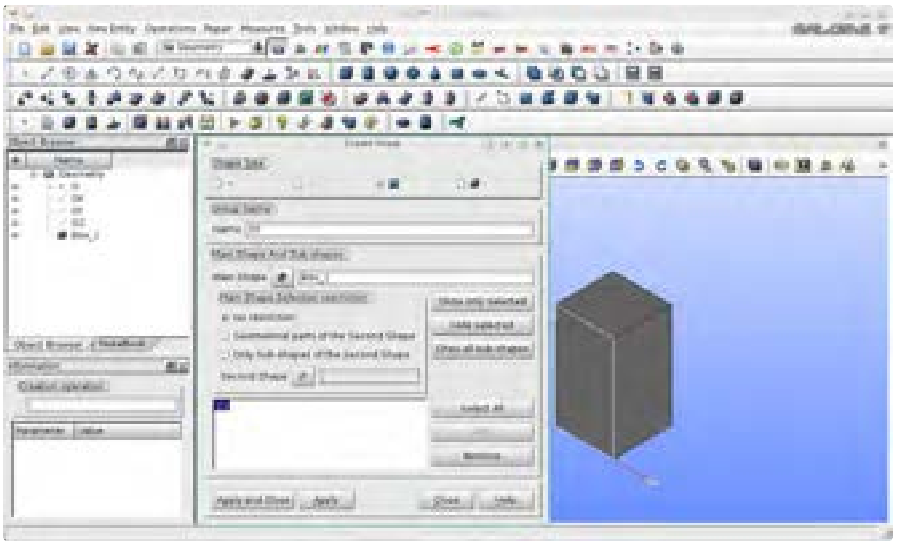


Figure 3.7: Test 1. Group creation for boundary conditions and coupling interfaces with the GEOM module.

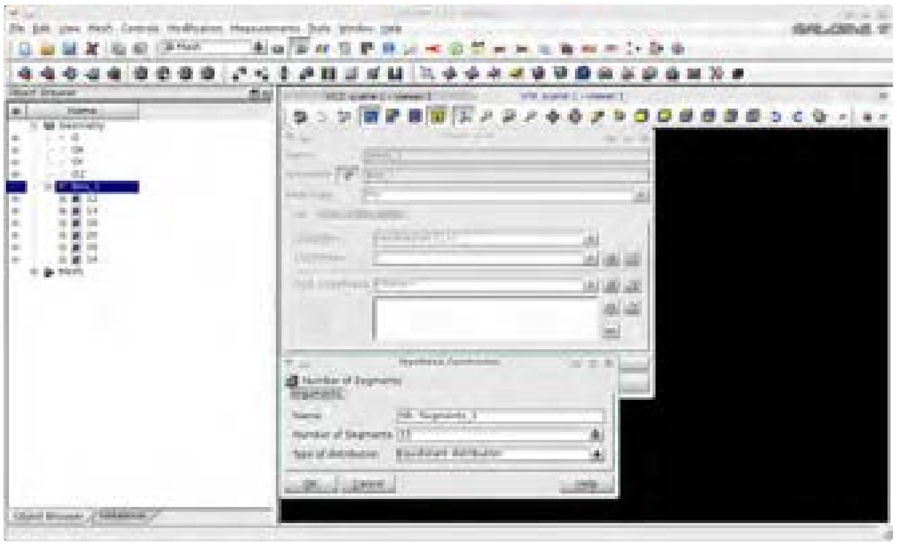


Figure 3.8: Test 1. Mesh creation with the SMESH module.

is selected. In Figure 3.9 the mesh is generated with the function *generate* of the SMESH module. Since the elements generated by default are linear (HEX8) the appropriate *from linear to quadratic* function is applied to transform them in quadratic ones. We recall that the velocity field should be quadratic, in order to satisfy the BBL condition, and therefore the element should be of HEX27 type. The coupling interfaces and boundary condition surfaces must have a local mesh. The local mesh is necessary to perform any

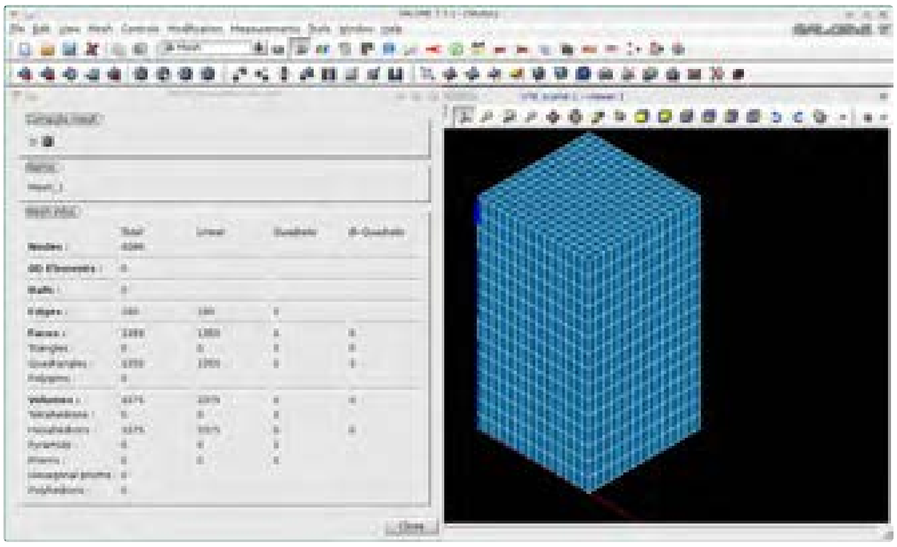


Figure 3.9: Test 1. Mesh generation with quadratic element HEX27.

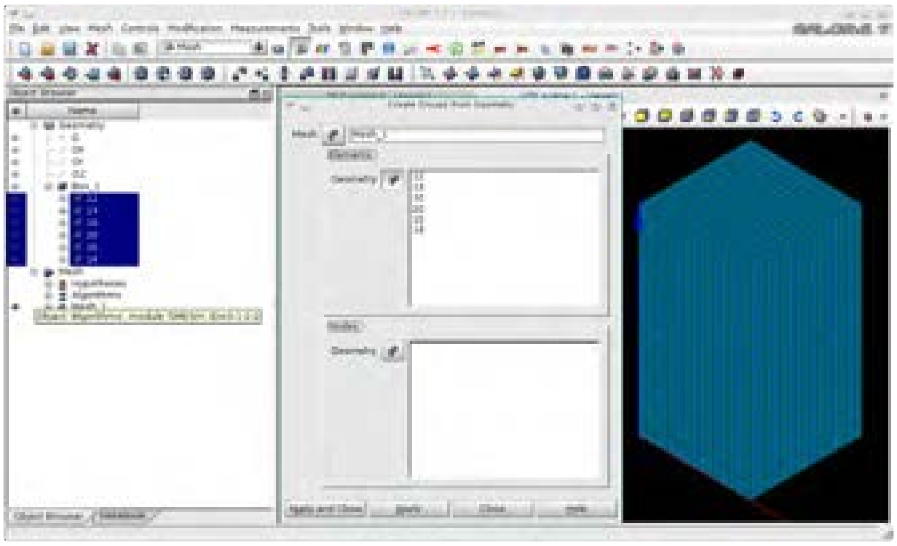


Figure 3.10: Test 1. The mesh group creation.

operations such as averaging, integration and differentiation. Instead of creating new groups we can import and adapt the geometric groups defined by the GEOM module. In Figure 3.10 we use the function import group from the SMESH module to perform this copy operation. The labels of the surfaces marked for boundary conditions and coupling interfaces are defined as follows. The inlet and the outlet of the reactor are labeled with 20 and 10 respectively. The reactor lateral surface are marked by 12,13, 14 and 15. The internal volume needed is label by 2. We recall that a number greater than 9 indicates a

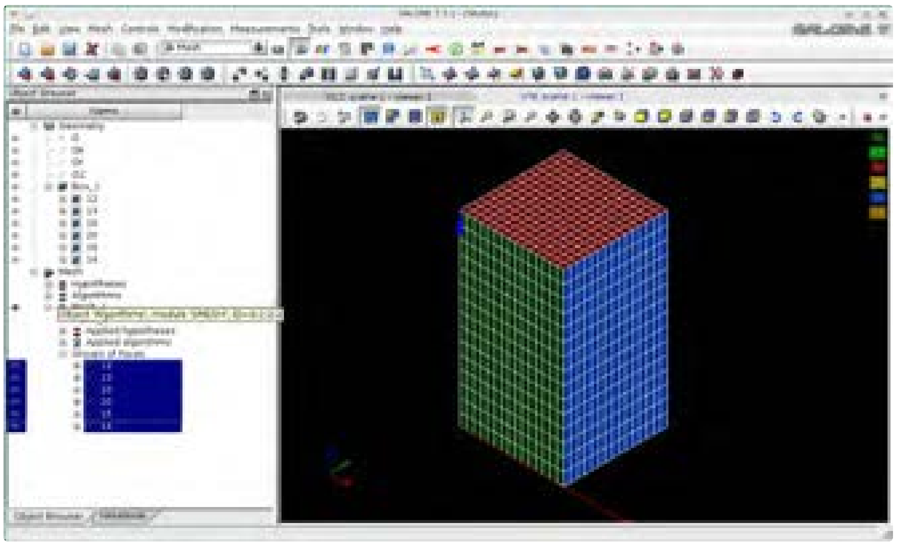


Figure 3.11: Test 1. Coarse basic mesh for the three-dimensional reactor region.

surface and less than 10 a volume by assumption. In Figure 3.11 the coarse basic mesh for the three-dimensional reactor region is shown. We recall that the coarse basic mesh must adapt over the assembly boundary. Mesh with elements that have surfaces between core assemblies is not allowed. A mesh refinement can be easily performed by using the FEMuS utility *gencase* in order to improve resolution inside the assembly grid.

3.2.3 Mono-dimensional primary loop mesh

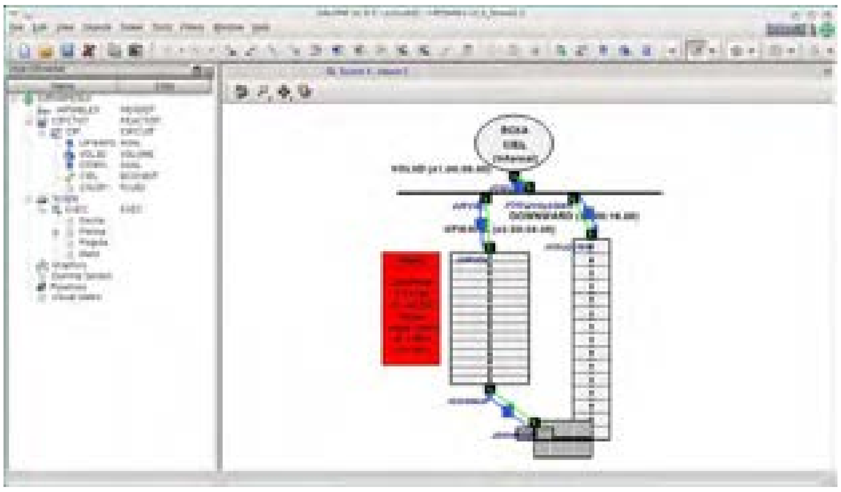


Figure 3.12: Test 1. Sketch of the primary loop components.

The mono-dimensional mesh is defined through the so-called Cathare *input deck* file, which is in this case the file *circuit_test1.dat*. The mesh can be generated by using GUITHARE, the graphical user interface on Salome platform. An overall sketch is shown in Figure 3.12. As said before in the mono-dimensional circuit the fluid exits from the point volume VOL enters the axial DOWNWARD, flows into the UPWARD branch and returns to the volume point VOL.

The mono-dimensional mesh is defined in the *circuit_test1.dat* produced by GUITHARE. In Figures 3.13–3.16 details on the circuit components and its mesh are shown. The directive defining this circuit in the input deck are as follows

```
CIR     = CIRCUIT UPWARD VOL3D DOWNWARD CIEL CALOPORT;
CIRCTOT = REACTOR CIR ;
```

There is only one circuit (labeled by CIR) which consists of 4 modules: UPWARD, VOL3D, DOWNWARD, CIEL. The UPWARD and DOWNWARD are axial mono-dimensional modules. VOL3D is a 0-dimensional volume module and CIEL is a BCONDIT module which is used to define pressure boundary condition in VOL3D. The key word CALOPORT is needed to identified the fluid. We define

```
CALOPORT = FLUID LEAD ;
```

at the beginning of the file to have lead as a coolant fluid. In Figure 3.13 Cathare *VOLUME* module (VOL3D) is shown. Here we report the corresponding Cathare command lines for this *VOLUME* component

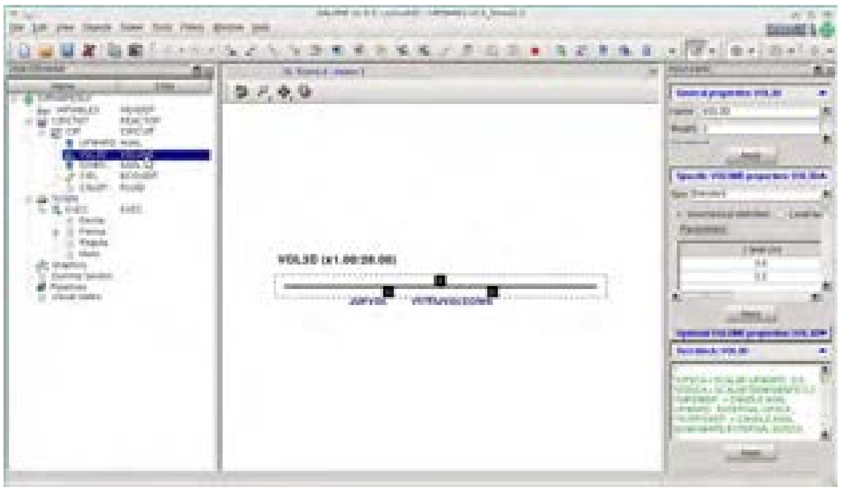


Figure 3.13: Test 1. The *VOLUME* module (VOL3D).

```
VOL3D = VOLUME JUPVOL USTREAM JVOLDOWN DSTREAM JCIEL DSTREAM ;
GEOM VOL3D
0.0  2.14
```

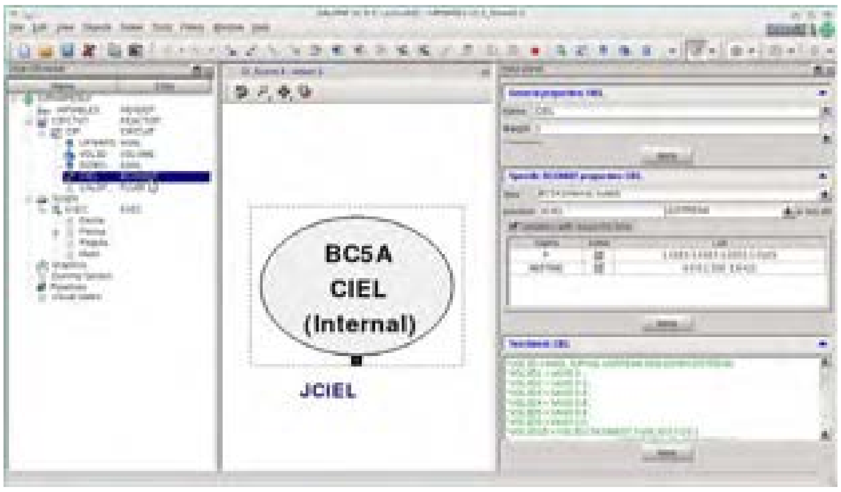


Figure 3.14: Test 1. The *BCONDIT* CIEL module.

```
0.2  2.14
JUPVOL   BOTTOM LENGTH 0. SECT .125 PERI .628 SIZE .2 VERTICAL
JVOLDOWN BOTTOM LENGTH 0. SECT .125 PERI .628 SIZE .2 VERTICAL
JCIEL    TOP    LENGTH 0. SECT .125 PERI .628 SIZE .2 VERTICAL;
```

VOL3D is a *VOLUME* module (0-dimensional) with 3 junctions JUPVOL (USTREAM), JVOLDOWN (DSTREAM) and JCIEL (DSTREAM). The USTREAM and DSTREAM label indicates the circuit orientation. The geometry of VOL3D is defined by the key GEOM. We have two points at level $z = 0.0$ and $z = 2.0$m with constant section of $2.14m^2$. The other three lines define the geometry (length,section,perimeter,size) of the junctions inside the *VOLUME* module. The junction direction is set vertical. The junction JCIEL is connect to the *BCONDIT* CIEL module. The *BCONDIT* CIEL module is described by

```
CIEL = BCONDIT JCIEL USTREAM ;
MODEL  CIEL BC5A
       P    REALLIST 1.01D5 1.01D5  1.01D5  1.01D5
 ABSTIME    REALLIST  0.0  0.1   100.   1.E+11 ;
```

The CIEL is a BCONDIT module with a junction JCIEL (USTREAM). The boundary condition model for CIEL is BC5A. The BC5A type of boundary conditions is an external outlet where one must specify only the pressure P. The pressure P is defined over three intervals of time $[0, 0.1]$, $[0.1, 100]$ and $[100, 1.E{+}11]$. In these three intervals the pressure is assumed to be constant at 1.01×10^5bar by the REALLIST command. In Figure 3.15 Cathare *AXIAL* DOWNWARD module is shown. The corresponding Cathare lines for this component are

```
DOWNWARD = AXIAL JVOLDOWN USTREAM JDOWNUP DSTREAM;
```

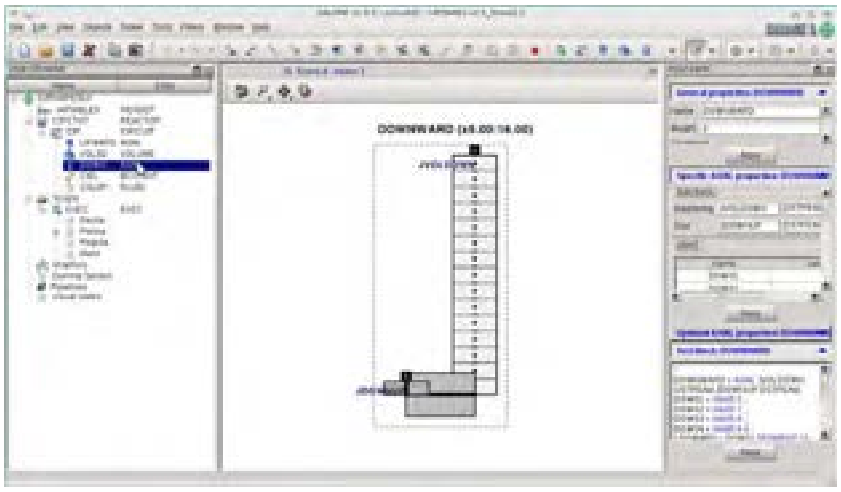


Figure 3.15: Test 1. The *AXIAL* DOWNWARD module.

```
DOW01 = XAXIS 0. ;
DOW02 = XAXIS 7. ;
DOW03 = XAXIS 9. ;
DOW04 = XAXIS 9.4 ;
LDOWARD = DOW01
SEGMENT 15 DOW02 COS -1.
SEGMENT 20 DOW03 COS  0.
SEGMENT 5 DOW04 COS  1. ;
MESH DOWNWARD LDOWARD;
GEOM DOWNWARD
(DOW01 AND DOW04) SECT 0.125 PERI 0.628 SIZE  0.2 ;
```

DOWNWARD is an *AXIAL* module with two junctions: JVOLDOWN (USTREAM) and JDOWNUP (DSTREAM). This component consists of 4 points: DOW01 ($x = 0$), DOW02 ($x = 7$), DOW03 ($x = 9$) and DOW04 ($x = 9.4$). It stars at the point DOW01 with 3 segments with 15 20 and 5 elements, respectively. The orientation of these three segment is defined by the cos directive: downward vertical for $COS = -1$, horizontal for $COS = 0$ and upward vertical for $COS = 1$. The section, perimeter and size of the pipe are define with the key directive GEOM. If the size is given the section and the perimeter can be automatically computed. In Figure 3.16 Cathare *AXIAL* UPWARD module is shown. Here we report the corresponding Cathare line for this component

```
UPWARD = AXIAL JDOWNUP USTREAM JUPVOL DSTREAM;
UPW01 = XAXIS 9.4;
UPW02 = XAXIS 16.0;
LUPWARD = UPW01 SEGMENT 15 UPW02 COS 1.  ;
MESH UPWARD LUPWARD;
```

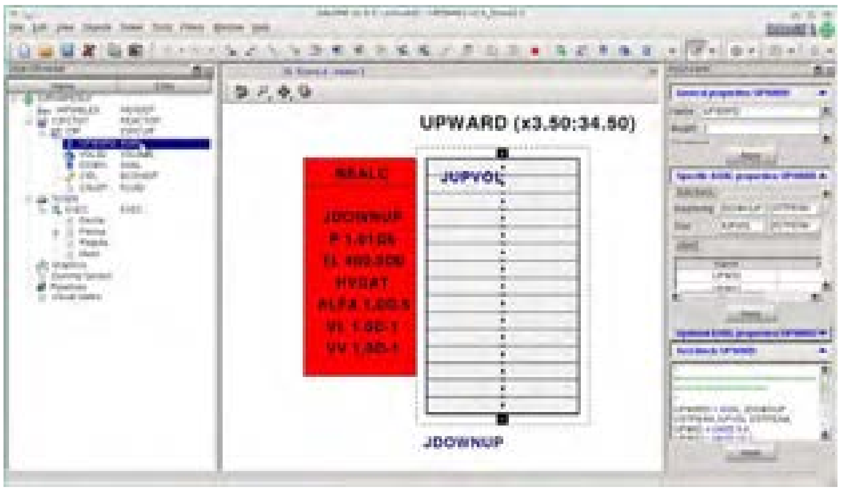


Figure 3.16: Test 1. The *AXIAL* UPWARD module.

```
GEOM UPWARD
(UPW01 AND UPW02) SECT 0.125 PERI 0.628 SIZE  0.2 ;
```

As written above the UPWARD component is an AXIAL connected between two junctions: JDOWNUP (USTREAM) and JUPVOL (DSTREAM). The USTREAM and DSTREAM label indicates the circuit orientation. The components has two points: UPW01 ($x = 9.4$) and UPW02 ($x = 16$). We remark that we have kept a unique coordinate system along the line for the UPWARD and DOWNWARD branch. There a unique segment with 15 element in the upward vertical ($COS = 1$) direction. The geometry is the same as the other *AXIAL* one.

The initial condition in the junction JDOWNUP is given as

```
 REALC JDOWNUP
   P      1.01D6
   TL     400.0D0
   HVSAT
   ALFA   1.0D-5
   VL     1.0D-1
   VV     1.0D-1;
GOPERM;
```

The Cathare code is a two-phase code and even if we use only fluid coolant initial and boundary conditions for the other phase should be given. In this case we assume gas with the same temperature and velocity with gas fraction irrelevant (ALFA=1.0D-5). The computation direction is fixed by

```
 PERMINIT CIR UPWARD VOL3D DOWNWARD  CIEL ;
```

We start from the UPWARD to VOL3D and to DOWNWARD. The CIEL BCONDIT is set at the end as usual.

3.2.4 Fuel distribution and pressure losses

The fuel and the pressure loss distribution are fundamental for the correct computation of the pressure and temperature fields. In this case we assume simple analytic distribution.

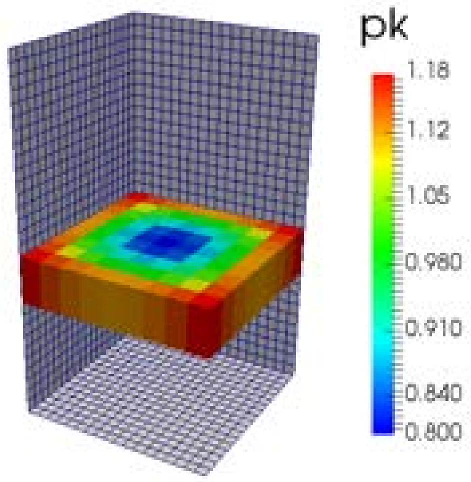


Figure 3.17: Distribution of the peak factor *pk* for fuel assembly.

In Figure 3.17 we show the distribution of the peak factor for fuel assembly for Test 1.

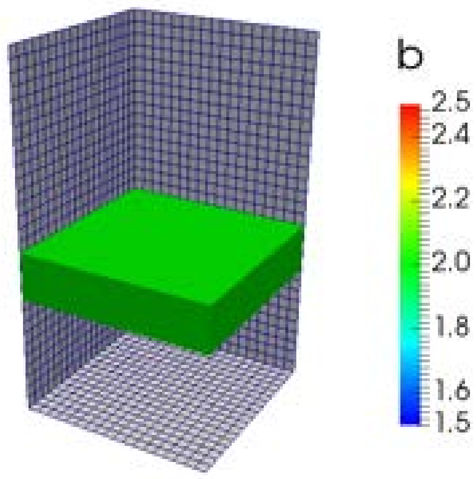


Figure 3.18: Distribution of the peak factor *b* for pressure losses.

The peak factor p_T is the ratio between the average source intensity and the assembly value. The average volumetric source q_{av} is set to

$$q_{av} = 1.176156 \times 10^8 \tag{3.1}$$

The expression of the peak factor p_L for pressure losses is set constant along the z direction while over the plane is distributed as follows

$$p_T(x, y) = 1.2 - .4 * x * (x - 3.66) * y * (y - 3.66) * 16/167.9616\,. \tag{3.2}$$

The distribution is in parabolic form. In order to have maximum intensity on the edges of the reactor and the distribution bounded between 0.8 and 1.2 we subtract the parabolic distribution to the maximum peak value 1.2.

In a similar way in Figure 3.18 the distribution of the peak factor b for pressure losses are reported. The accidental pressure losses are assumed to be constant in a well specified vertical region. There is no particular physical sense for this assumption. The purpose of this choice is only to set enough intensity on a local region to that they can be seen in the the pressure solution when compared with gravity pressure losses.

3.2.5 Coupling FemLCore-Cathare on Salome platform for Test 1

The project should be written in the *USER_APPL* directory of the FEMuS code. This code has a directory structure similar to projects developed on Salome platform module and therefore we can reuse this structure again even for the Cathare input/output. The directory structure is shown in Figure 3.4. All the necessary files with the exception of the mesh file are contained inside the application directory *FemLCore_cathare1*. The input data files are in the *DATA directory*. The basic c++ files of the FemLCore code such as the Navier-Stokes and Energy equation solver together with Cathare links and FORTRAN files can be found in the *SRC* directory. The CFD three-dimensional mesh is in the *MESH* directory while the *cathare.dat* file defining the mono-dimensional geometry is stored in the *SRC* directory. The execuTable program is generated in this main directory while all the results both from FemLCore and Cathare are written in the *RESU* directory. The coupling code starts three problems: P (FemLCore), C (Cathare) and N problem (neutronics or heat sources).

The P problem is solved by the 3D-porous module of the FemLCore code. The *P problem* is defined by two parameter files: *parameters.in* and *param_files.in*. The *MGUtils* class is the class that controls the parameter reading from the appropriate files stored in the *DATA* directory. Through the *MGUtils* class the parameters, the mesh and the basic structure directory name can be loaded. The coarse mesh is generated by the SALOME platform as discussed in Section 3.2.2. In this case we use a coarse mesh in MED Format contained in the file *reactor.med* The multigrid and multiprocessor mesh is generated by the *gencase* application by using the LIBMESH library [37, 38]. In order to assemble the discrete matrix of the Navier-Stokes and energy system with the Finite element method we need linear, quadratic and piecewise basis functions together with the topology of the appropriate geometric element. The appropriate *MGFE* and *MGGeomEl* classes must be initialized before the construction of the P problem. In order to

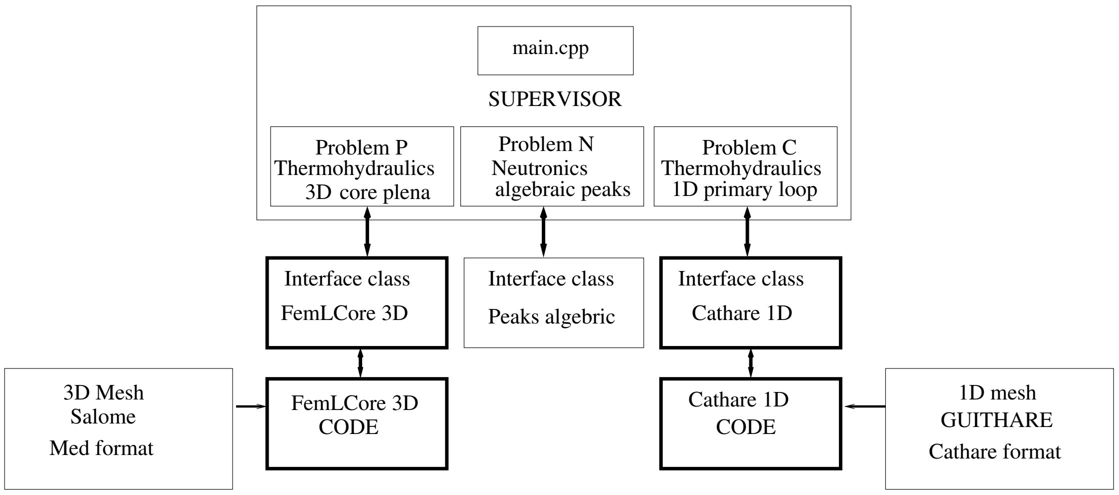


Figure 3.19: Test 1. Schematic of the coupling computations

specify the equation system to be solved the *std::vector<FIELDS> myproblemP* must be filled. The dimension is 2 if Navier-Stokes system (NS_F) and energy equation (T_F) are to be solved. Once the utility classes (mgfemap(*MGFE*),mgutils[0](*MGUtils*), mggeomel(*MGGeomEl*)) are initialized the P problem can be set by the following standard code lines

```
FEMUS P;                          // constructor
P.init_param(*mgutils[0]);        // init parameter
P.init_fem(*mggeomel,*mgfemap);// init fem
// setting mesh ----------------------------------------
P.setMesh();                      // set MGmesh
// setting system --------------------------------------
P.setSystem(myproblemP,0,2);      // set system 2 cell vector
// setting inital condition at t=0 -----------------------
P.solve_setup(itime_0,time);      // initial time loop (t=0)
```

The FemLCore interface to Salome platform is called FEMuS. Each command needed to construct the problem is contained in this class. The parameters and the FEM element approximations are linked to the FEMuS interface and then to the FemLCore core class. The mesh in *MED* format is loaded in the FemLCore mesh structures. Two copies of the same mesh in different formats are allocated in the interface FEMuS class. This allows the transfer of data from the code to the Salome platform and viceversa. The function *setSystem(myproblemP,0,2)* initializes the system NS_F and T_F and allocates two piecewise-constant external fields. These external fields can be used by a neutronic code with the appropriate Salome interface. In order to exchange data from the three-dimensional simulation to the mono-dimensional one we need to define interface in the P Problem. The interfaces are created by writing

```
P.init_interface(10,10, 2,"mesh_file");//  top temperature
```

```
P.init_interface(33,10, 1,"mesh_file");//  top pressure
P.init_interface(11,20, 2,"mesh_file");//  bottom temperature
P.init_interface(21,20, 2,"mesh_file");//  bottom velocity
P.init_interface(31,20, 1,"mesh_file");//  bottom pressure
```

From Figure 3.20 we can see the location of the groups defined from mesh generator. With the first line the group 10 is associated with top surface which is the outlet of the reactor. However on the top surface we need to transfer data for the temperature and pressure fields. Therefore the interface with name 10 is created on the mesh group 10 for quadratic temperature field and the interface with name 33 is created on the mesh group 10 for linear pressure field. It is important to remark that the creation of the interface generates a local mesh which is quadratic and all the linear functions must be adapted over a coarser mesh. The bottom surface is the inlet of the reactor where we need to exchange temperature, velocity and pressure. For this reason three interfaces labeled 11, 21, 31 are defined over the same mesh geometric group 20.

The generation of the heat source is called *N Problem*. Since we are not interested in neutronic computations, the interface with a neutronic code is substituted by a simple interface that generates a semi-analytic heat source profile. This simple interface is defined by the *ReactData* class. In order to define the *N Problem* we write

```
ReactData *N;
N = new ReactData();
N->datagen(P.get_MGExtSystem());
```

N is therefore the interface that controls the data flow for the heat and pressure loss sources inside and out of the reactor.

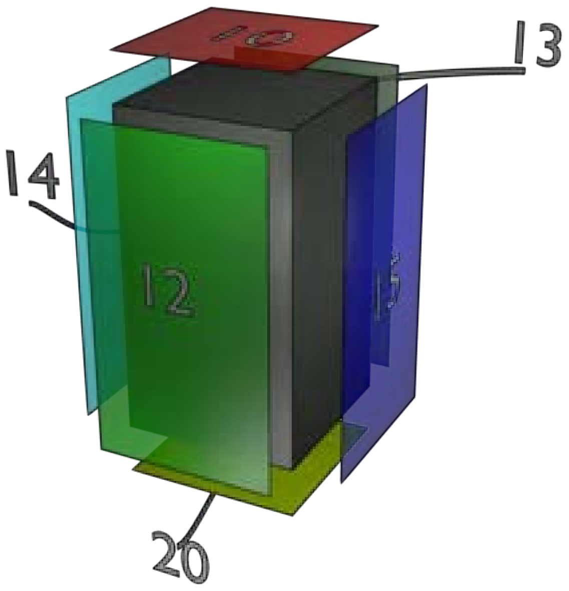


Figure 3.20: Test 1. The surface labels for boundary conditions and coupling interfaces.

The *C problem* is a Cathare problem. The interface class is called *Problem_Cathare*. The initialization of this class can be done simply by

```
Problem_Cathare* c = new Problem_Cathare();
c->initialize();
```

The interface points for the Cathare problem can be defined as in Table 3.1. At the the

location	name	component	position (ix)	rad (irad)
inlet	point 1	DOWNWARD	40	0
outlet	point 2	DOWNWARD	1	0
pump	pump pt	DOWNWARD	25	0
exchanger	heat_ex pt	DOWNWARD	25	0

Table 3.1: Test 1. Cathare key interface points.

point 1 located at the inlet, with coordinates $x = ix\_1$ of the *DOWNWARD* module we can get all the variables with the following Cathare interface command

```
rho_1=c->getValue_Cathare("LIQDENS","DOWNWARD",ix_1,0);
f_1=c->getValue_Cathare("LIQFLOW","DOWNWARD",ix_1,0);
P_1=c->getValue_Cathare("PRESSURE","DOWNWARD",ix_1,0);
h_1=c->getValue_Cathare("LIQH","DOWNWARD",ix_1,0);
T_1=c->getValue_Cathare("LIQTEMP","DOWNWARD",ix_1,0);
v_1=c->getValue_Cathare("LIQV","DOWNWARD",ix_1,0);
```

With *LIQDENS*, *LIQFLOW*, *PRESSURE*, *LIQH*, *LIQTEMP* and *LIQV* as arguments this function returns the value of density, mass flow, pressure, enthalpy, temperature and velocity, respectively.

The pump and the heat exchanger are set in the *DOWNWARD* module. The pump model, which is located at the coordinate $x = ix\_p$, is very simple. In the pump point we impose a constant pressure with

```
c->setValue_Cathare("DPLEXT","DOWNWARD",ix_p,0,DP);
```

where *ix_p* and *irad* are defined as in Table 3.1. $DP = .75 \times 10^5$Pa is the pump gain in pressure. The heat exchanger is located on the *DOWNWARD* module at the point with coordinates $x = ix\_h$, see Table 3.1. The heat exchanger can be modeled by the following code lines

```
T_ex=c->getValue_Cathare("LIQTEMP","DOWNWARD",ix_h, 0);
h_ex=c->getValue_Cathare("LIQH","DOWNWARD",ix_h, 0);
h_ex1 = h_ex -w_ex*(cp*(T_ex-T_ex2));
c->setValue_Cathare("ENTLEXT","DOWNWARD",ix_h,0,h_ex1);
```

The cooling temperature T_{ex2} is set to $400C$. The enthalpy h_{ex} is determined iteratively to the corresponding value of the cooling temperature T_{ex2}. The constant w_{ex} is the feedback constant or the underelaxation constant in the iterative algorithm.

At the point 1 and point 2 defined in Table 3.1 we have the 3D/1D interfaces. At the point 2 the three-dimensional state must pass the temperature value to the Cathare code. The mono-coordinate of the Cathare code of the point 2 is labeled ix_2. In order to set the temperature we use a feedback algorithm. Let T_{outr} be the average temperature that flows out from the three-dimensional reactor and t_{21}/h_{21} be the temperature/enthalpy on the other side of the interface on the Cathare mono-dimensional mesh (point 2). The temperature t_{21} is corrected until it reaches T_{outr} by these code lines

```
t_21=c->getValue_Cathare("LIQTEMP","DOWNWARD",ix_2,0;
h_21=c->getValue_Cathare("LIQH","DOWNWARD",ix_2, 0;
h_12 = h_21-(cp*(t_21-T_outr));   // hentalpy correction
c->setValue_Cathare("ENTLEXT","DOWNWARD",ix_2, 0,h_12);
```

The pressure coupling is obtained by imposing the three-dimensional average pressure drop $P[0] - P0[0]$ to the mono-dimensional part of the circuit between the point 1 and point 2. In this way we correct the mono-dimensional drop with the correct one. We write the following code lines

```
 press_p2=c->getValue_Cathare("PRESSURE","DOWNWARD",ix_2,0);
 press_p1=c->getValue_Cathare("PRESSURE","UPWARD",ix_1,0);
 dp_ioo=c->getValue_Cathare("DPLEXT","UPWARD",2,0);
 dp_io=dp_ioo-0.5*(fem_rho*(h*g+averageP[0]-averageP0[0])-
                                (press_p1-press_p2));
 dp_io=dp_ioo;
 c->setValue_Cathare("DPLEXT","UPWARD",2,0,dp_io);
```

On the inlet section of the reactor we need to set temperature and velocity field. Since we do not know the field distribution at the reactor we assume a constant distribution. The temperature T_1 from the point 1 of Cathare should be imposed to the inlet reactor. We write

```
// Temperature coupling  T_1(Cathare) with interface 11(bottom)
std::ostringstream a; a<< T_1;
P.setAnalyticBoundaryValues(11,1, a.str().c_str());
P.write_Boundary_value(11,"T",1);// write inside x_old sol
```

and for the velocity field we write

```
// velocity coupling  v_1(Cathare) with interface 21 (bottom)
std::ostringstream vel; vel<< "IVec*0.+JVec*0+KVec*"<< v_1;
P.setAnalyticBoundaryValues(21,3, vel.str().c_str());
P.write_Boundary_value(21,"NS0",3); // write inside x_old
```

The boundary conditions for temperature and velocity are imposed directly by using the FEMuS interface functions *write_Boundary_value* directly on the old solution vector *x_old*.

The reactor temperature T_{outr} on the outlet is compute as average on the surface 10 by using the FEMuS interface command *getAvOnBoundary_nodes* as

```
T_outr=P.getAvOnBoundary_nodes(10, "T",0);
```

In a similar way the pressure at the outlet $P[0]$ and at the inlet $P0[0]$ can be computed with the same function as

```
P[0] = P.getAvOnBoundary_nodes(31, "NS0",DIMENSION);
P0[0] = P.getAvOnBoundary_nodes(33, "NS0",DIMENSION);
```

3.2.6 Field

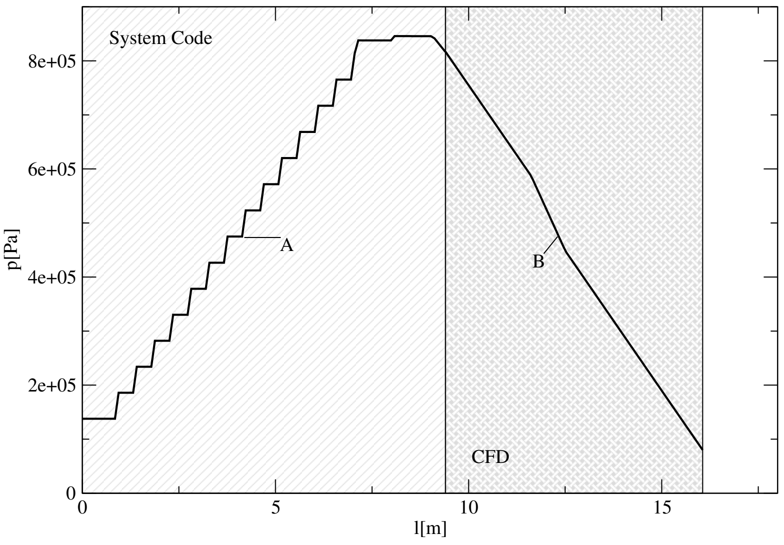


Figure 3.21: Test 1. Pressure field at working condition.

In order to compute the solution at working condition we start the computation from uniform initial condition $v = 0.0$m/s and temperature $T = 400C$. We apply an inlet velocity of 0.1m/s and the value of the pump pressure $DP= .75 \times 10^5$Pa. The pressure from the pump increase the velocity until this matches the pressure losses of the circuit. Figures 3.21–3.23 show the computed solution in the three and mono-dimensional domain when the steady state is reached (working condition). In Figure 3.21 one can see the computed solution for the pressure field. On the left we show the system code results (A) and on the left the CFD results (B) along the center line of the reactor. The system code results are in the stair shape since we consider piecewise approximations (finite volume) in the system code while in the FEM approximation the solution are compute as piecewise linear. The pressure is reported along the mono-dimensional line. At $x = 0$ we are at the beginning of the DOWNWARD axial, at the outlet of the

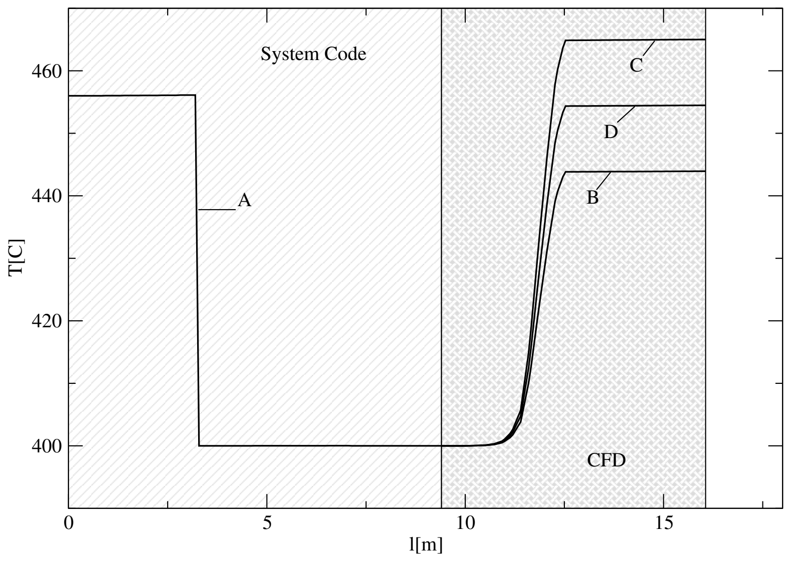


Figure 3.22: Test 1. Temperature field at working condition.

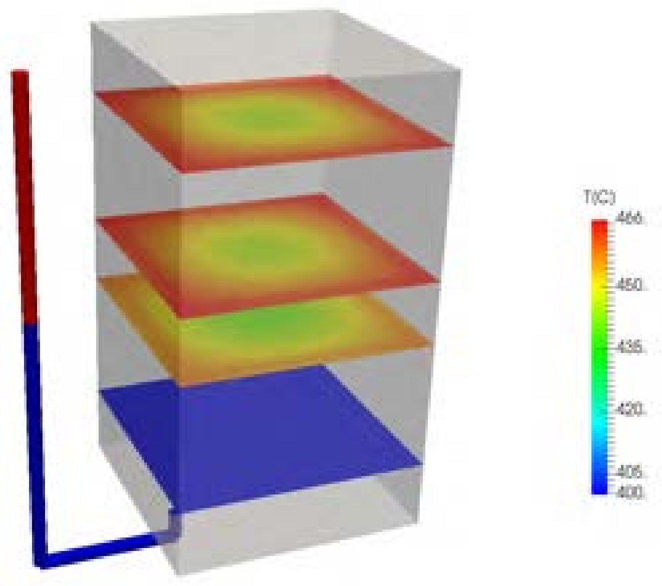


Figure 3.23: Test 1. Temperature field at working condition over different sections at $z = 1, 3, 4, 6$.

reactor. This components consists of three pieces: downward, horizontal and upward. In the downward part the pressure increases linearly as the pressure is mainly defined by gravity. At $x = 7$ the horizontal part starts and the pressure becomes constant. In the middle of the horizontal pipe the pump is set. A small jump due to the pump can be seen at $x = 8$. In the final upward part of the mono-dimensional circuit the pressure decrease almost linearly as a result of the gravity force. At the interface defined by the reactor inlet the pressure between the FemLCore and Cathare is matched perfectly. Inside the three-dimensional reactor the pressure decreases linearly due to the gravity force. The distributed pressure losses are negligible but the accidental pressure drop is visible in the interval $x \in [13, 14]$ as expected.

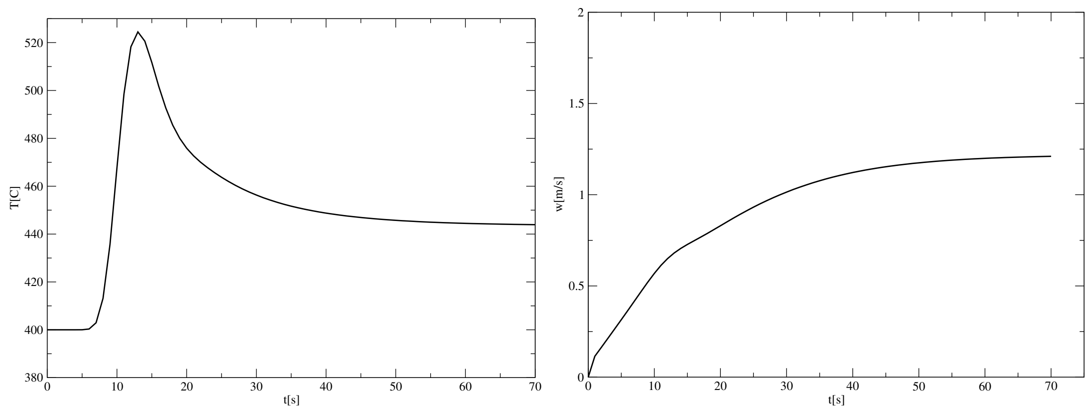


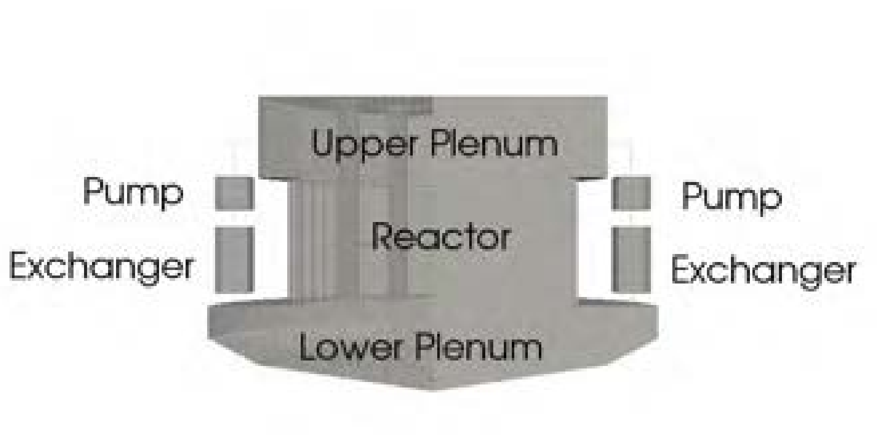
Figure 3.24: Test 1. Temperature (left) and velocity (right) field at the outlet of the DOWNWARD component as a function of time.

In Figure 3.22 the Temperature field at working condition for the test 1 is reported. On the left we show the system code results (A) and on the left the CFD results along three different lines (A), (B) and (C). At the working condition at $x = 0$ we are at the outlet of the reactor. The temperature is constant until the heat exchanger is reached ($x = 3$). The fluid exits the heat exchanger at $T = 400C$. At the interface defined by the reactor inlet the temperature between the FemLCore and Cathare is matched perfectly. Inside the three-dimensional reactor the temperature increases due to the heat source. The mono-dimensional circuit gets the average temperature at the inlet of the reactor. In Figure 3.23 we show the temperature field at working condition over different sections at $z = 1, 3, 4$ and 6. The temperature pattern over these sections is determined by fuel distribution.

In Figure 3.24 the temperature and the velocity field at the inlet of the DOWNWARD component is reported as a function of time. This point is the point that defines the boundary condition at the outlet of the reactor. At $t = 0$ the temperature is $T = 400C$. This point starts to feel the increasing in temperature $t \approx 9$ when the reactor warm up. There is a temperature peak that reaches $T \approx 520C$. The average temperature decreases till reach its steady value. On the right there is the velocity behavior. Since we start with small velocity $v = 0.1$m/s the velocity increases. The pump pressure defines the circuit flow rate which increases till it reaches the steady state at the mean velocity of approximately 1.4m/s.

3.3 Test 2 for FemLCore-Cathare coupling

3.3.1 General description



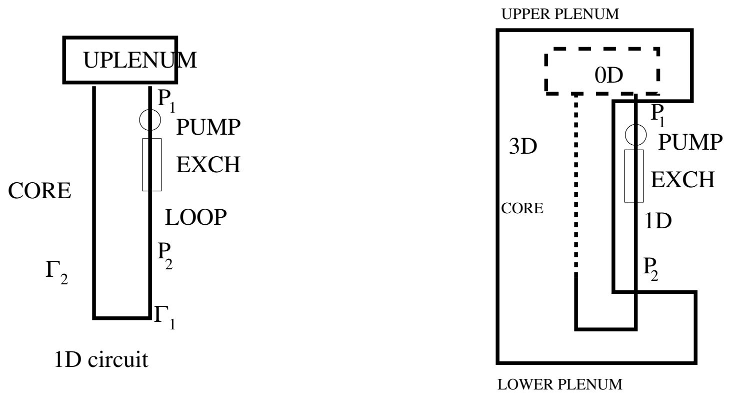


Figure 3.25: Test 2. On the top sketch of the coupling. On the bottom the mono-dimensional circuit (on the right) and the coupling 3D-1D domain geometry (on the left).

In this test we couple a three-dimensional Lead cooled reactor which consists of a core, an upper and a lower plenum with a mono-dimensional primary loop defined through an input deck of the Cathare code.

On the top of Figure 3.25 a general sketch of the reactor and primary loop is shown. Details about the coupling between the mono and three-dimensional domain geometry are illustrated on the bottom. In particular on the left bottom the mono-dimensional mesh is shown. This circuit consists of three pieces: a point volume (UPLENUM), and two *AXIAL* modules (CORE and LOOP). In the mono-dimensional circuit the fluid flows from the *VOLUME* module and moves downward then horizontally and finally vertically to enter again into the *VOLUME* module. On the right bottom the three-dimensional domain, which consists of the reactor plena and core, is drawn together with the mono-dimensional one. In this coupled geometry, the fluid flows from the Upper Plenum FemLCore module and moves downward from point P_1 to P_2 through the LOOP Cathare *AXIAL* module then enters to the Lower Plenum FemLCore module. The three-dimensional region is overlapped to the mono-dimensional one along the horizontal LOOP and the vertical CORE module.

3.3.2 FemLCore 3D mesh and geometry

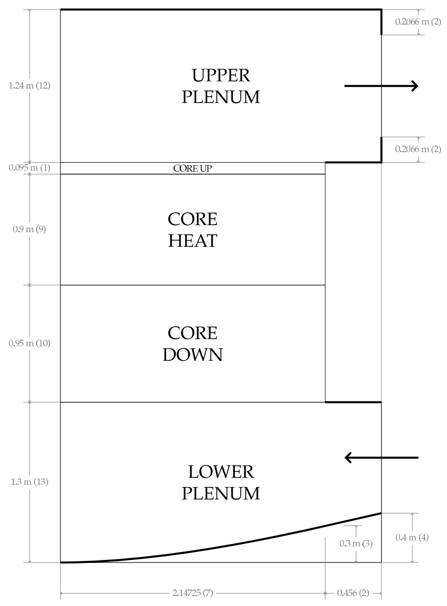


Figure 3.26: Schematic of the reactor core and plenum section.

The LFR model for these computations is the type-pool LFR reactor ELSY (European Lead-cooled System) with schematic section shown in Figure 3.26. This reactor is characterized by a very compact design with main characteristics reported in Table 3.2. As shown in Figure 3.26 we consider the LFR system basically divided into three regions: heat exchange loop (1D-porous), plenum (3D-CFD) and core (3D-porous) region. Let us consider the active (upper) and non-active (lower) core sections and the upper and lower plenum as defined in Figure 3.26. If we set the zero vertical coordinate at the lower point, the core region goes from $1.3m$ to $3.24m$. The active core (upper core) where heat is generated ranges between $H_{in} = 2.25m$ and $H_{out} = 3.15m$. Below the core we have the lower plenum with the inlet between 0 and $H_{lp} = 1.3m$. The lower plenum has an approximate hemispherical form with the lowest region at $H_{bot} = 0$ (reference point). Above the core for a total height of $1.24 = H_{up}m$ there is the upper plenum with the coolant outlet. A detailed description of the ELSY reactor is presented in [9, 10, 14].

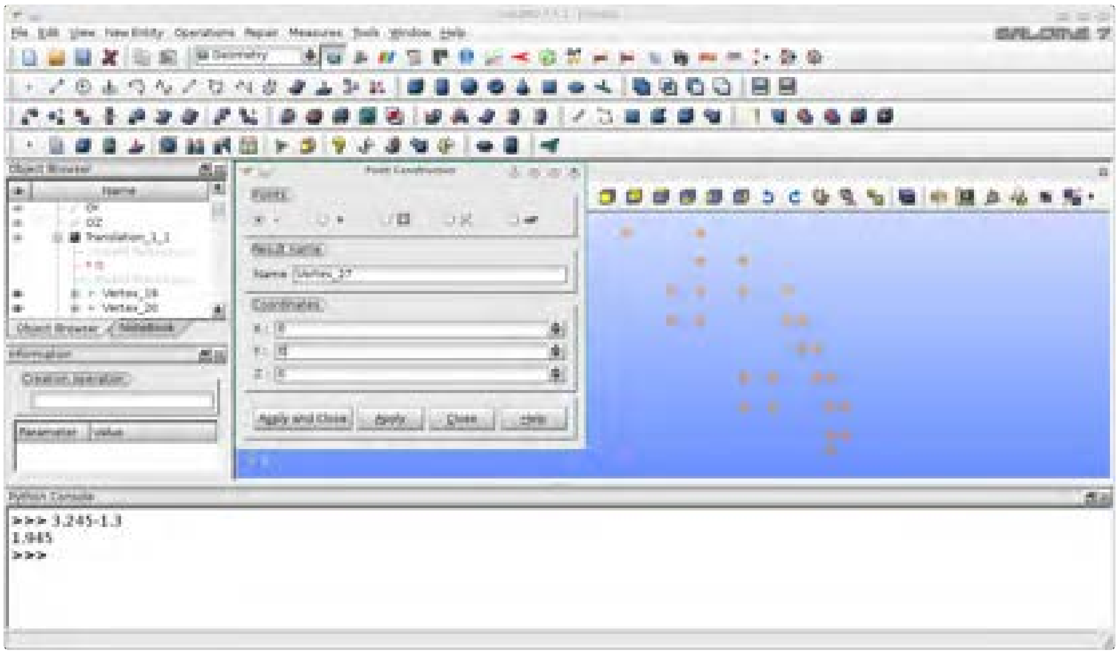
The geometry and the mesh of the FemLCore three-dimensional domain is built in

Parameters	ELSY
Power [MWe]	600
Conversion Ratio	~ 1
Thermal efficiency [%]	42
Primary coolant	Lead
Primary coolant circulation	Forced
Primary coolant circulation for DHR	Natural
Core inlet temperature	673.15K
Core outlet temperature	753.15K
Fuel	MOX (Nitrides)
Fuel cladding material	T91 (aluminized)
Peak cladding temperature	823.15K
Fuel pin diameter [mm]	10.5
Active core dimensions	Height/diameter [m] 0.9/4.32
Primary pumps	8 integrated in the SG
Working fluid	Water-superheated 18MPa,450ÂřC
Primary/secondary heat transfer sys	8 Pb-to-H2O SGs
Direct heat removal (DHR)	Reactor Vessel Air Cooling Sys
	+ 4 Direct Reactor Cooling Sys
	+ 4 Secondary Loops Cooling Sys

Table 3.2: Main characteristics of the ELSY reactor.

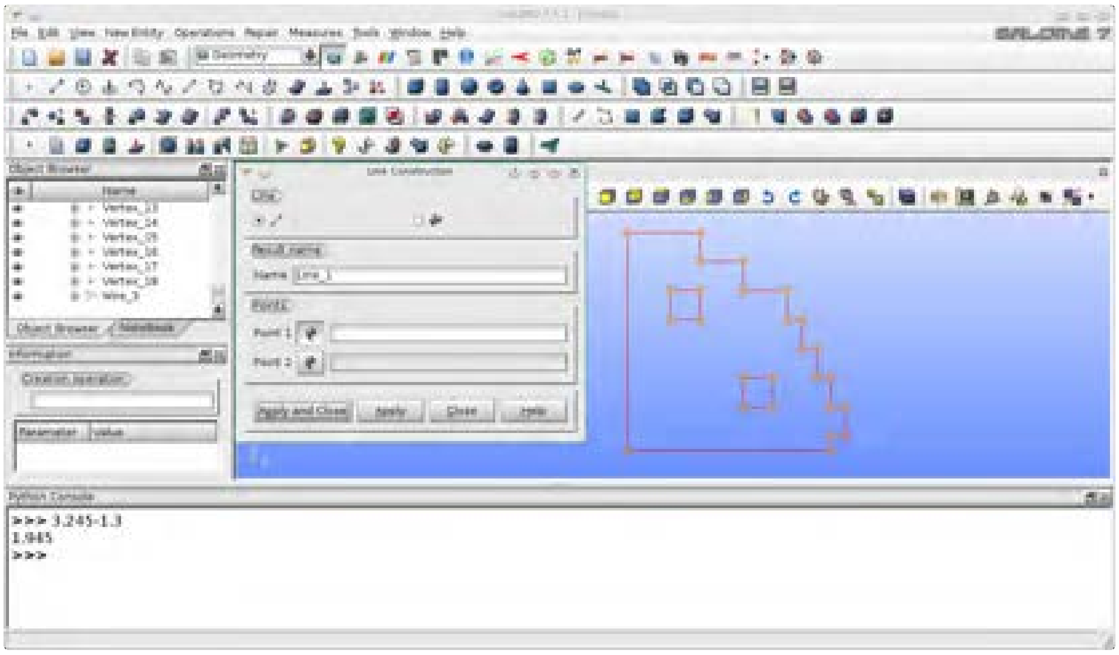
this section. This geometry is rather complex since the core fuel assembly, which is basically squared, should not intersect with the meshing of the reactor in all its parts. The mesh is constructed with quadratic HEX27 elements since the FEM bases should be of Taylor-Hood type: the velocity field is quadratic while the pressure fields is supposed to be linear. We recall that the velocity field should be quadratic, in order to satisfy the BBL condition.

The project is developed under the Salome platform and therefore we use the GEOM and the SMESH modules to draw and mesh the reactor domain. We open the platform and select the GEOM module in order to draw the required geometry. The steps performed with the GEOM module are summarized in Figures 3.27–3.34. As shown in Figures 3.27–3.30 we draw points that identify the external domain of the core, then the surface and finally the core volume. We recall that the interior contains two empty assemblies which are used to locate the emergency rod control. In Figures 3.31–3.33 we summarize the draw of the upper and lower plenum. The FemLCore needs to mark the surfaces where special operations should be performed. In Figure 3.34 we create the geometry groups for boundary conditions and coupling interfaces. The *create group* function of the GEOM module is used. Then we open the SMESH module of the Salome Platform and create the mesh. The hypothesis construction, based on simple uniform subdivisions, is selected. In Figure 3.9 the mesh is generated with the function *generate* of the SMESH module. Since the elements generated by default are linear (HEX8) the



```
>>> 3.245-1.3
1.945
>>>
```

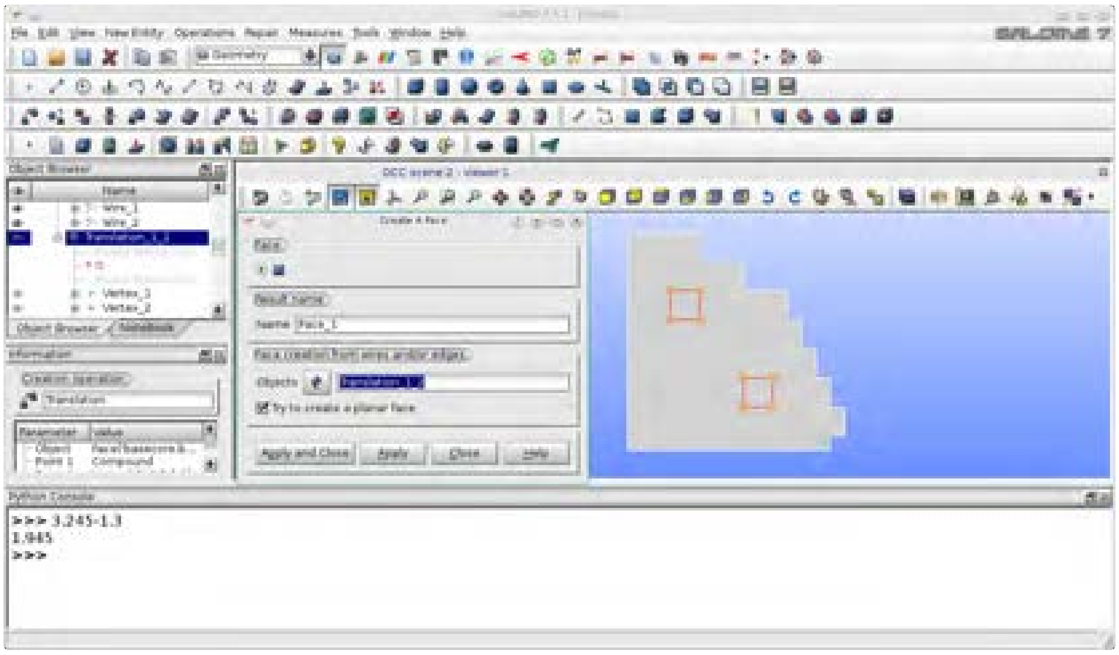
Figure 3.27: Test 2. Core geometry construction: key points.



```
>>> 3.245-1.3
1.945
>>>
```

Figure 3.28: Test 2. Core geometry construction: contour lines.

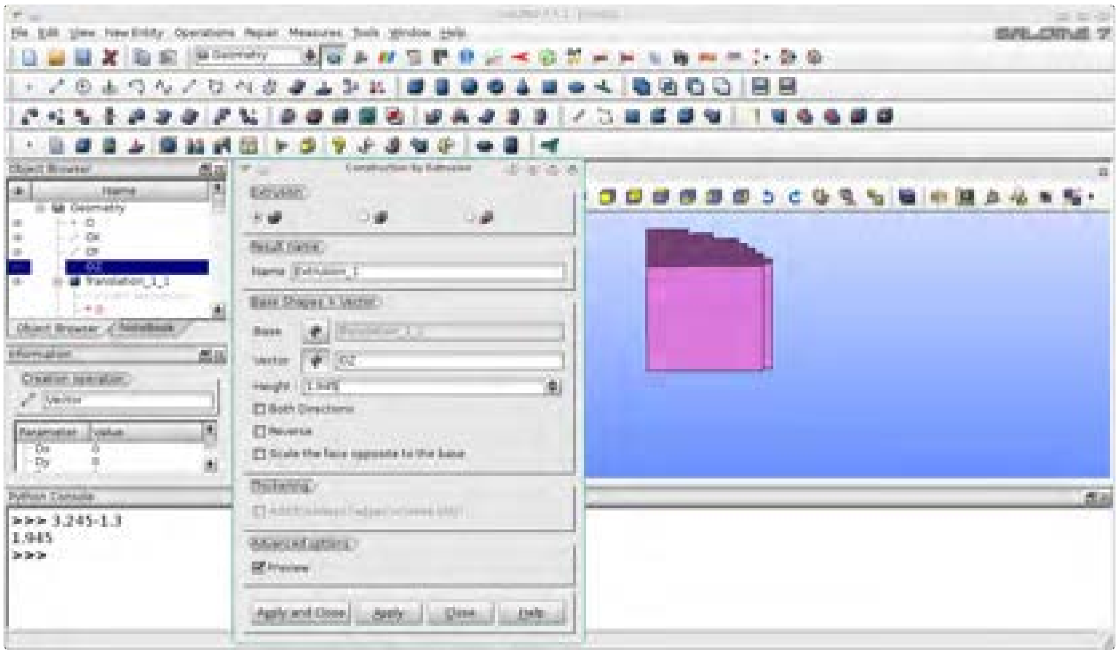
appropriate *from linear to quadratic* function is applied to transform them in quadratic


File Edit View New Entity Operations Repair Measures Tools Window Help
SALOME 7
Object Browser
Information
Create A Face
Face
Result name
Face creation from wires and/or edges
Apply and Close
Apply
Close
Help
Python Console

Name: Face_1
Objects: Translation_1_1
☑ Try to create a planar face

```
>>> 3.245-1.3
1.945
>>>
```

Figure 3.29: Test 2. Core surface construction.



Name: Extrusion_1
Base: Translation_1_1
Vector: OZ
Height: 1.945

```
>>> 3.245-1.3
1.945
>>>
```

Figure 3.30: Test 2. Core volume construction.

ones. The coupling interfaces and boundary condition surfaces must have a local mesh.

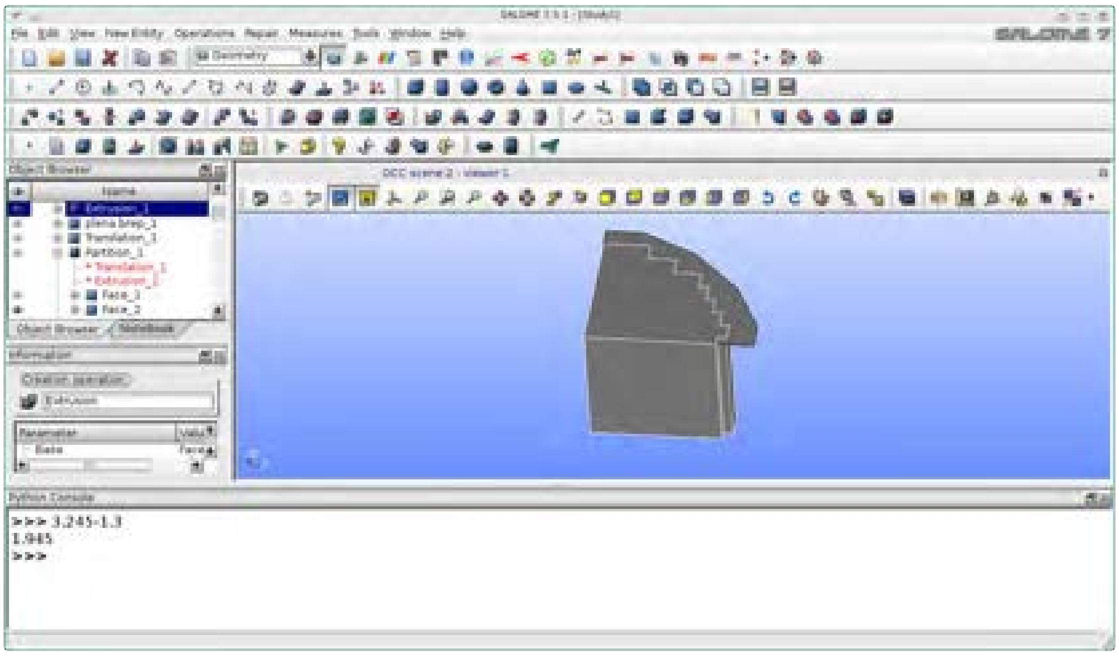
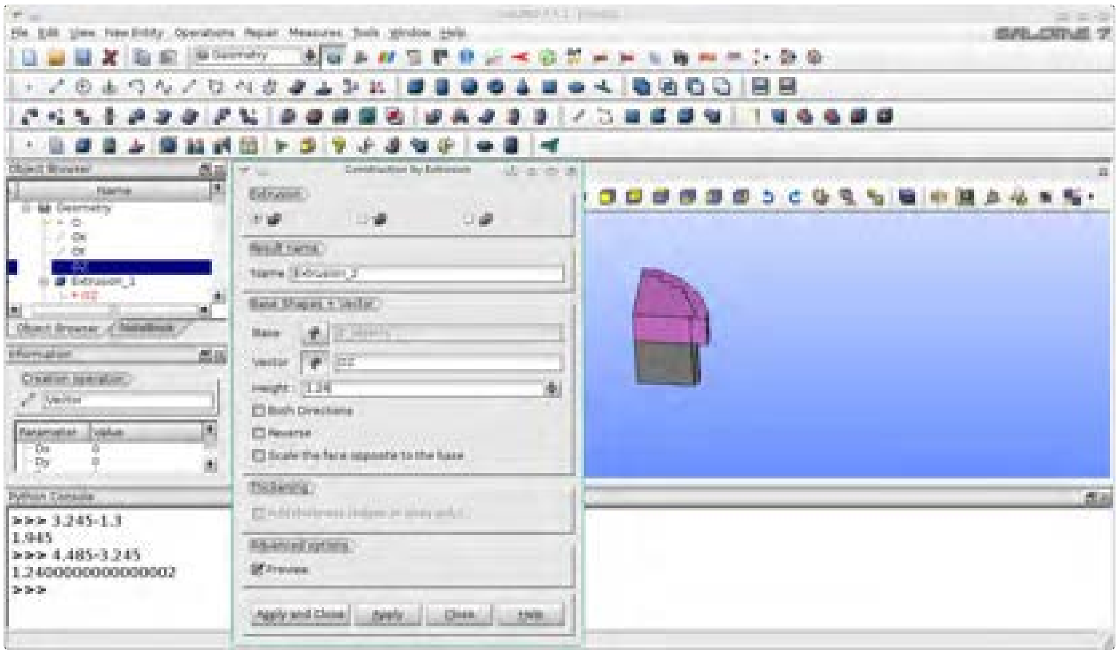

```
>>> 3.245-1.3
1.945
>>>
```

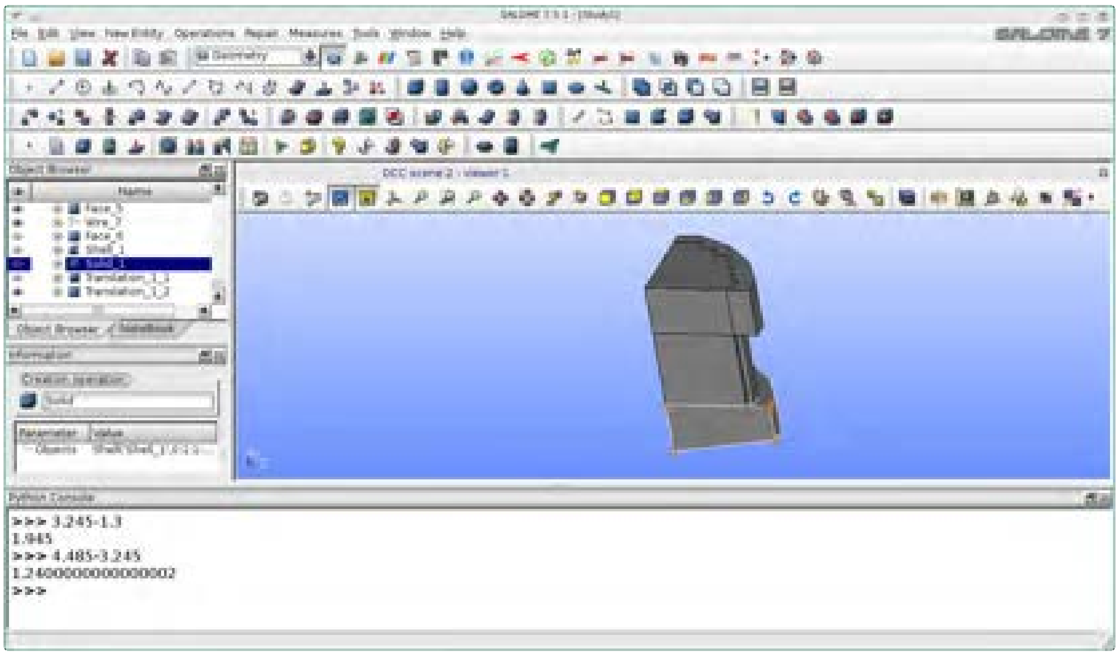
Figure 3.31: Test 2. Upper plenum construction: bottom surface.


Name: Extrusion_2
Vector: DZ
Height: 1.24

```
>>> 3.245-1.3
1.945
>>> 4.485-3.245
1.2400000000000002
>>>
```

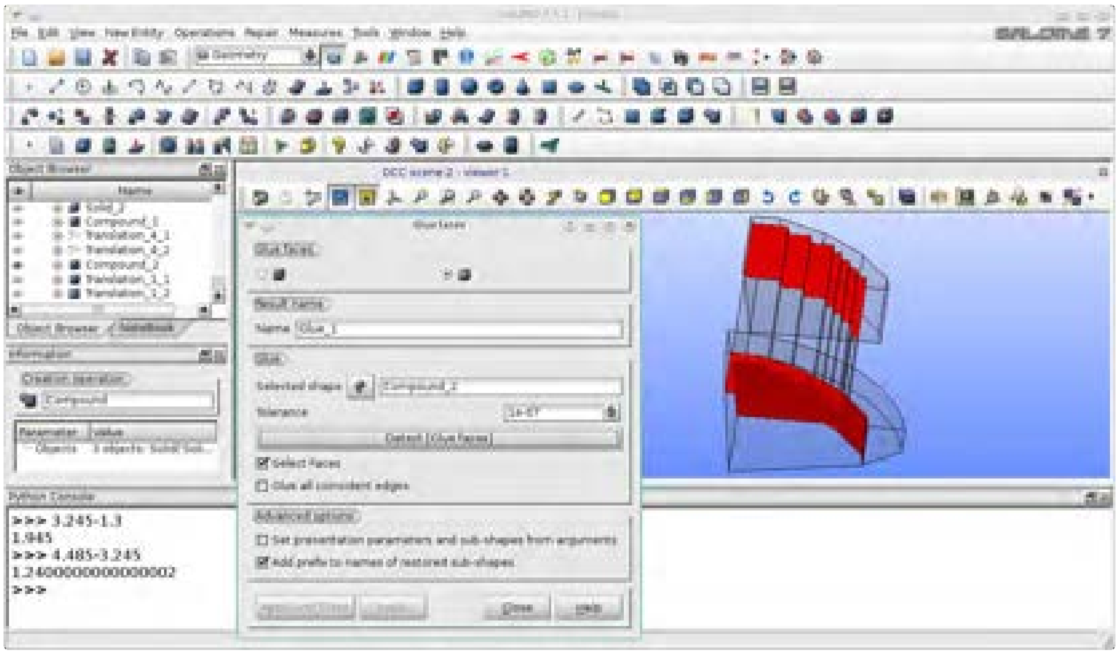
Figure 3.32: Test 2. Upper plenum construction: volume.

The local mesh is necessary to perform any operations such as averaging, integration and



```
>>> 3.245-1.3
1.945
>>> 4.485-3.245
1.2400000000000002
>>>
```

Figure 3.33: Test 2. Lower plenum construction: surface and volume.



Name: Glue_1
Selected shape: Compound_2
Tolerance: 1e-07
☑ Select Faces
☐ Glue all coincident edges
☐ Set presentation parameters and sub-shapes from arguments
☑ Add prefix to names of restored sub-shapes

```
>>> 3.245-1.3
1.945
>>> 4.485-3.245
1.2400000000000002
>>>
```

Figure 3.34: Test 2. Geometry group construction.

differentiation. Instead of creating new groups we can import and adapt the geometric

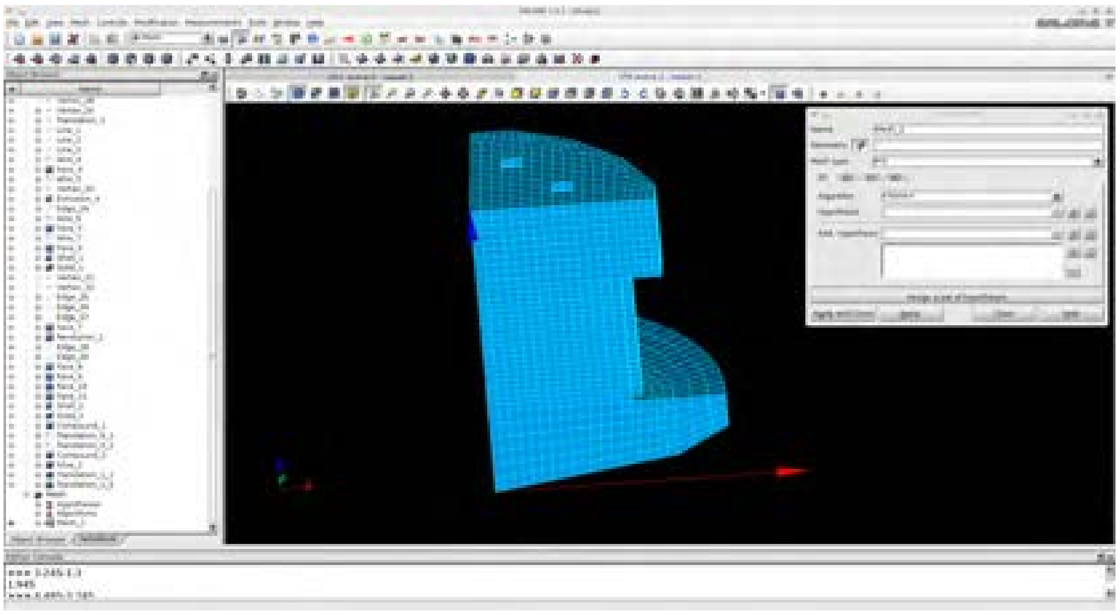


Figure 3.35: Test 2. Mesh construction.

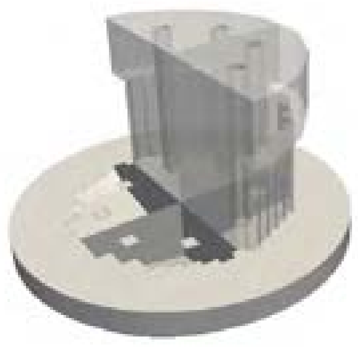


Figure 3.36: Test 2. Total view of the lower plenum computational domain.

groups defined by the *GEOM* module. The labels of the surfaces marked for boundary conditions and coupling interfaces are defined as follows. The reactor inlet is labeled by 4000 while the outlet consists of two parts: 1200 and 1100. A mesh with midpoint refinement can be easily generated by using the FEMuS utility *gencase* in order to improve resolution inside the assembly grid. Finally in Figure 3.36-3.37 the total view of the computational domain. In particular in Figure 3.36 one can see the lower plenum computational domain and in Figure 3.37 the total view of the core and upper plenum computational domain.

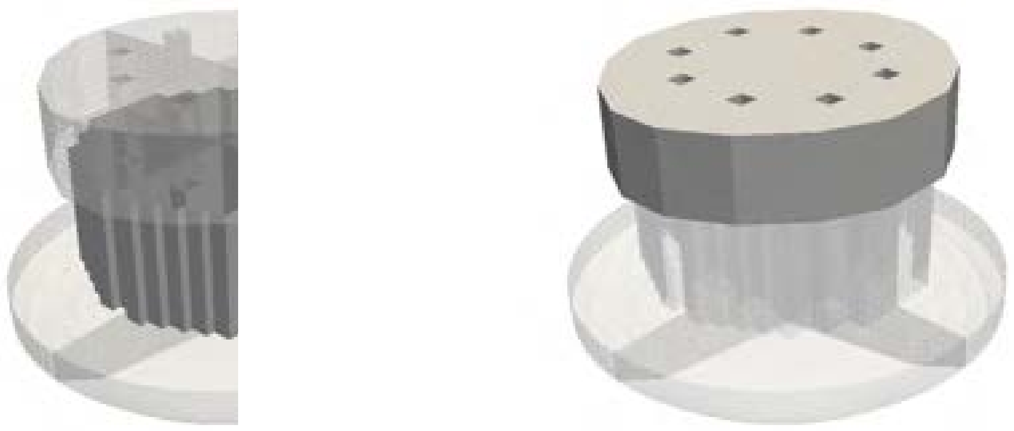


Figure 3.37: Test 2. Total view of the core and upper plenum computational domain.

3.3.3 Mono-dimensional primary loop mesh

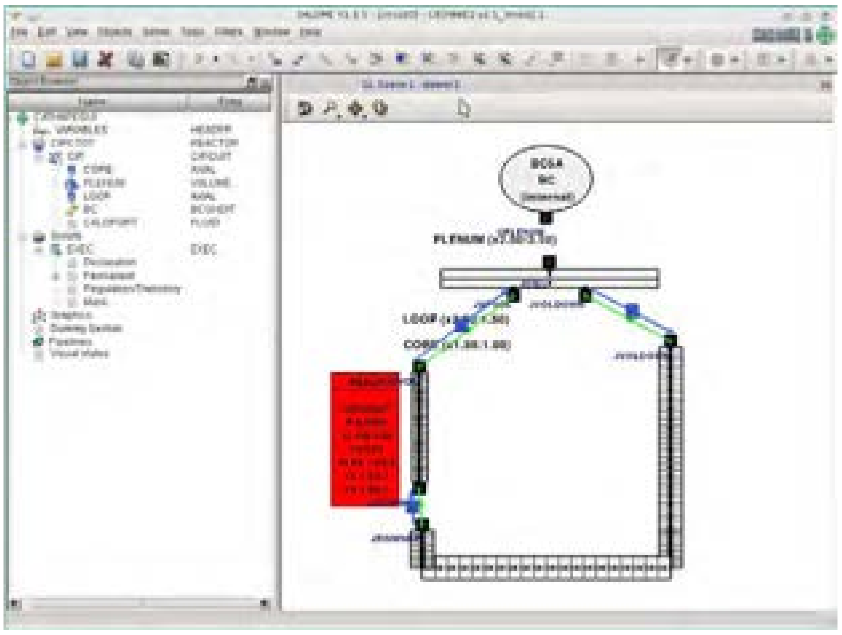


Figure 3.38: Test 2. Cathare circuit: BC, UPLENUM, CORE and LOOP mono-dimensional module.

The mono-dimensional mesh is defined by the file *circuit_test2.dat*. The mesh is generated by using the graphical user interface GUITHARE on Salome platform as shown in Figure 3.38. The circuit (labeled by CIR) consists of 4 modules: CORE, UPLENUM, LOOP, BC. The CORE and LOOP are *AXIAL* mono-dimensional modules. UPLENUM is a 0-dimensional *VOLUME* module and BC is a *BCONDIT* module which is used to define pressure boundary condition in UPLENUM. In the mono-dimensional circuit the fluid flows from the point volume UPLENUM enters the *AXIAL* LOOP module, flows into the *AXIAL* CORE and returns to the volume point UPLENUM as reported in the directive of the *circuit_test2.dat* as

```
CIR     = CIRCUIT CORE UPLENUM CORE BC CALOPORT;
CIRCTOT = REACTOR CIR ;
```

The key word CALOPORT identifies the coolant as Lead if one writes

```
CALOPORT = FLUID LEAD
```

in the file directives. It is clear that the CORE represents the reactor core while the plena are modeled by UPLENUM and the horizontal part of the LOOP module. Only the downward part of the LOOP module is outside the reactor. In Figures 3.39–3.42 details on the circuit components and its mesh are shown.

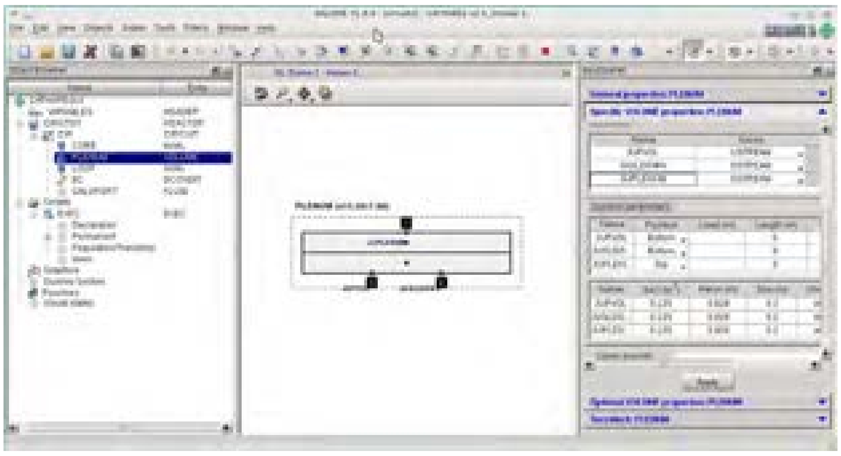


Figure 3.39: Test 2. The *VOLUME* module (UPLENUM).

In Figure 3.39 Cathare *VOLUME* module (UPLENUM) is shown. Here we report the corresponding Cathare command lines for this *VOLUME* component

```
UPLENUM=VOLUME JUPVOL USTREAM JVOLDOWN DSTREAM JUPLENUM DSTREAM;
GEOM VOL3D
0.0  2.14
0.2  2.14
JUPVOL   BOTTOM LENGTH 0. SECT .125 PERI .628 SIZE .2 VERTICAL
JVOLDOWN BOTTOM LENGTH 0. SECT .125 PERI .628 SIZE .2 VERTICAL
```

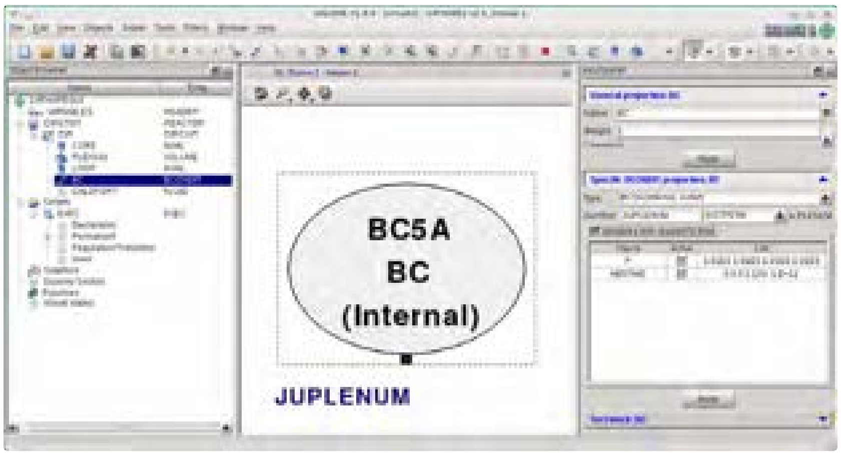


Figure 3.40: Test 2. The *BCONDIT* BC module.

```
JUPLENUM TOP    LENGTH 0. SECT .125 PERI .628 SIZE .2 VERTICAL;
```

UPLENUM is a *VOLUME* module (0-dimensional) with 3 junctions JUPVOL (USTREAM), JVOLDOWN (DSTREAM) and JUPLENUM (DSTREAM). The geometry of UPLENUM, defined by the key GEOM, has two points at level $z = 0.0$ and $z = 2.0$m with constant section of $2.14m^2$. The junction direction is set vertical. The junction JUPLENUM is connect to the *BCONDIT* BC module. The *BCONDIT* BC module is described by

```
BC = BCONDIT JUPLENUM USTREAM ;
MODEL  CIEL BC5A
       P    REALLIST 1.01D5 1.01D5  1.01D5  1.01D5
 ABSTIME    REALLIST  0.0  0.1    100.   1.E+11 ;
```

The BC is a *BCONDIT* module with a junction JUPLENUM (USTREAM). The boundary condition model for BC is BC5A. The BC5A is an external outlet boundary condition that needs to specify the pressure P. In this case, as shown in the directive Cathare code, the pressure P is defined to be constant at 1.01×10^5bar over three intervals of time $[0, 0.1]$, $[0.1, 100]$ and $[100, 1.E + 11]$. In Figure 3.41 Cathare *AXIAL* LOOP module is shown. The corresponding Cathare lines for this component are

```
LOOP = AXIAL JVOLDOWN USTREAM JDOWNUP DSTREAM;
LOOP01 = XAXIS 0. ;
LOOP02 = XAXIS 1.95 ;
LOOP03 = XAXIS 2.35 ;
LOOP04 = XAXIS 5. ;
LOOP05 = XAXIS 5.4 ;
LLOOP = LOOP01
SEGMENT 40 LOOP02   COS -1.
```

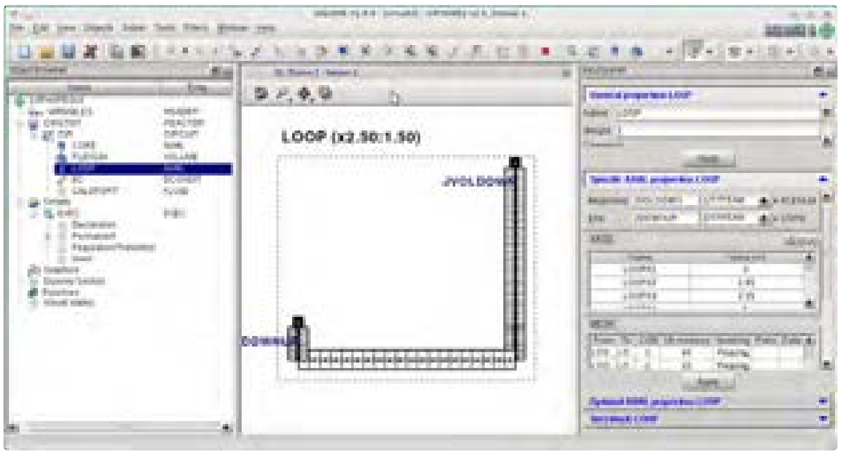


Figure 3.41: Test 1. The *AXIAL* LOOP module.

```
SEGMENT  10 LOOP03  COS -1.
SEGMENT 20 LOOP04   COS  0.
SEGMENT  10 LOOP05  COS  1. ;
MESH LOOP LLOOP;
GEOM LOOP (LOOP01 AND LOOP05)
   SECT 0.125 PERI 0.628 SIZE  0.2 ;
```

LOOP is an *AXIAL* module with two junctions: JVOLDOWN (USTREAM) and JDOWNUP (DSTREAM). This component consists of 5 points: LOOP01 ($x = 0$), LOOP02 ($x = 1.95$), LOOP03 ($x = 2.35$), LOOP04 ($x = 5$) and LOOP05 ($x = 5.4$). It stars at the point L00P1 with 3 segments with 15 20 and 5 elements, respectively. The orientation of these three segment is defined by the cos directive: downward vertical for $COS = -1$, horizontal for $COS = 0$ and upward vertical for $COS = 1$. The downward vertical part is the primary loop outside the reactor while the horizontal and upward part model with a mono-dimensional mesh the lower plenum inside the reactor. In Figure 3.42 Cathare *AXIAL* CORE module is shown. Here we report the corresponding Cathare line for this component

```
CORE = AXIAL JDOWNUP USTREAM JUPVOL DSTREAM;
CORE01 = XAXIS 5.4 ;
CORE02 = XAXIS 7.35 ;
LCORE = CORE01 SEGMENT 30 CORE02 COS 1.   ;
MESH CORE LCORE;
GEOM CORE  (CORE01 AND CORE02)
          SECT 0.125 PERI 0.628 SIZE  0.2 ;
```

As written above the CORE component is an *AXIAL* connected through the reactor core between two junctions: JDOWNUP (USTREAM) and JUPVOL (DSTREAM).

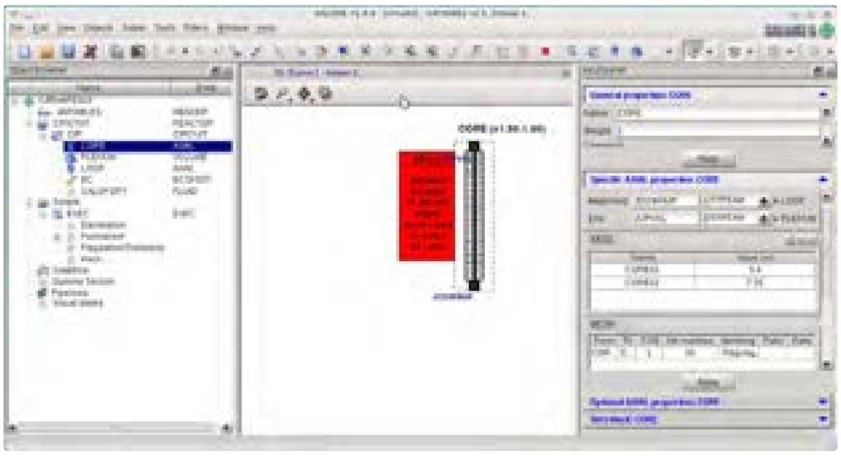


Figure 3.42: Test 1. The *AXIAL* CORE module.

The components has two points: CORE01 ($x = 5.4$) and CORE02 =($x = 7.35$). We remark that we have kept a unique coordinate system along the line for the CORE and LOOP module.

The initial condition in the junction JDOWNUP is given as

```
 REALC JDOWNUP
   P     8.05D5
   TL    400.0D0
   HVSAT
   ALFA  1.0D-5
   VL    1.0D-1
   VV    1.0D-1;
GOPERM;
```

Since Cathare code is a two-phase code we assume gas with the same temperature and velocity with gas fraction irrelevant such that we have $ALFA = 1 \times 10^{-5}$. The computation direction is fixed by

```
 PERMINIT CIR CORE UPLENUM LOOP BC;
```

We start from the CORE to UPLENUM and to LOOP. The BC *BCONDIT* is set at the end as usual.

3.3.4 Fuel distribution and pressure losses

The heat source distributions generated by the fuel assembly and its geometry are computed in the *Reactor* class. The *Reactor* class consists of the *Reactor.cpp* file and its header *Reactor.h* [20, 18]. This distribution is taken from the ELSY reactor. The core reactor is divided in square assemblies whose characteristics are shown in Table 3.3. The number of assemblies in the core is 170 distributed as shown in Figure 3.43. Eight of

	area (m^2)
Pin area	370.606×10^{-4}
Corner box area	5.717×10^{-4}
Central box beam	2.092×10^{-4}
Channel central box beam area	12.340×10^{-4}
Coolant area	473.605×10^{-4}
Assembly area	864.360×10^{-4}
Coolant/Assembly ratio	0.5408

Table 3.3: Test 2. Main characteristics of the fuel assembly.

these locations are dedicated to house special control rods leading to the global number of heating assemblies to 162. Each assembly has a square section with a side length of $L = 0.294m$ and coolant/assembly ratio $r = 0.548$ (see Figure 3.26). For details see [10, 21, 23, 26, 8]. Our model distributes the fuel assemblies in three radial zones:

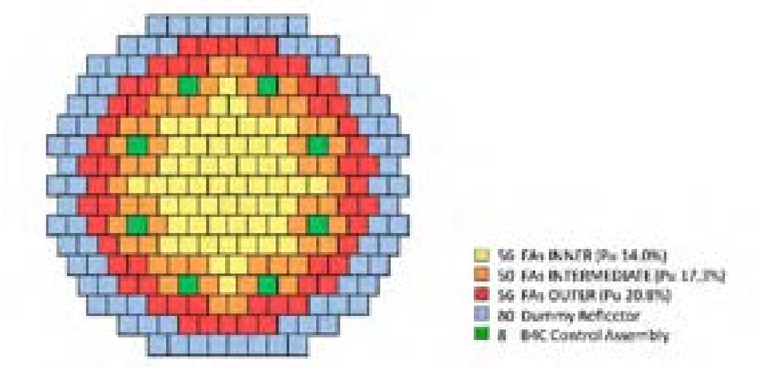


Figure 3.43: Test 2. Reactor fuel overview distributed in three radial zones.

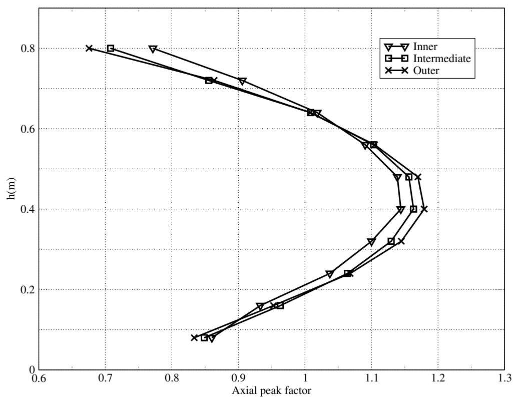


Figure 3.44: Test 2. Axial fuel distribution.

Inner		Intermediate		Outer	
A11/2*	8.606	A16/2*	10.775	A17/2*	10.036
A12/2*	8.801	A25	10.188	A27	10.908
A13/2*	9.043	A26	10.269	A28	7.842
A14/2*	9.301	A34	9.943	A37	8.465
A15/2*	9.560	A36	9.249	Aa7	8.520
A21/2*	8.678	A45	10.132	A55	10.962
A22	8.759	A46	9.399	A56	8.621
A23	8.943	A52	9.696	A65	9.766
A24	9.112	A53	9.857	A66	7.776
A3l	8.811	A54	10.400	A72	9.746
A32	8.921	A62	9.410	A73	8.833
A33	9.071	A64	9.057	A74	7.699
A41/2*	8.850	A71	9.318	A81/2*	8.029
A42	8.903			A82	7.806
A43	9.043			A83	6.922
A44	9.218				
A5l	8.790				
A61/2*	8.725				

	Power $MWth$	# FA	P_{avg} $MWth$	ffrad
Inner	501.41	56	8.95	1.07
Intermediate	489.23	50	9.78	1.10
Outer	491.60	56	8.78	1.25
Total	1482.24	162	9.15	

Table 3.4: Test 2. Horizontal fuel distribution.

$N_{a1} = 56$ fuel assemblies in the inner zone, $N_{a2} = 62$ fuel assemblies in the intermediate zone and the remaining $N_{a2} = 44$ fuel assemblies in the outer one. The power distribution is based on the power distribution factors pk_f, which of a fuel assembly over the average fuel assembly power, The power distribution is obtained by multiplying the average value with the power distribution factor. The core is not axial symmetric but it has two symmetry planes passing through the reactor axis so that only a quarter domain has to be taken into account for the simulations. They are mapped in Table 3.4, where the horizontal distribution is shown and in Figure 3.44 where the axial distribution is reported. The power factor is in the range of 0.74–1.17. In the computational model the following matrix of fuel pick factor pk_f is assumed

$$
\begin{array}{c}
\begin{array}{c} A1 \\ A2 \\ A3 \\ A4 \\ A5 \\ A6 \\ A7 \\ A8 \end{array}
\left[\begin{array}{cccccccc}
0.941 & 0.962 & 0.989 & 1.017 & 1.045 & 1.178 & 1.097 & 0. \\
0.949 & 0.958 & 0.978 & 0.996 & 1.114 & 1.123 & 1.193 & 0.857 \\
0.963 & 0.975 & 0.992 & 1.087 & R & 1.011 & 0.925 & 0. \\
0.967 & 0.973 & 0.989 & 1.008 & 1.108 & 1.028 & 0.931 & 0. \\
0.961 & 1.060 & 1.078 & 1.137 & 1.198 & 0.943 & 0. & 0. \\
0.954 & 1.029 & R & 0.990 & 1.068 & 0.850 & 0. & 0. \\
1.019 & 1.066 & 0.966 & 0.842 & 0. & 0. & 0. & 0. \\
0.878 & 0.853 & 0.757 & 0. & 0. & 0. & 0. & 0.
\end{array}\right] \\
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array}
\end{array}
$$

The assembly is denoted by two numbers. The first number represents the row while the second number the column. $A13$ means assembly located in the row 1 and column 3. R is the control assembly and the number 0.0 denotes the assembly where no fuel is present. The In this test we are not interested in the the natural convection flow however we set all properties as a function of temperature when necessary. For liquid lead coolant one can refer to Table 3.5 [28, 27, 12]. The reference temperature T_{ref} is the inlet temperature in the lower plenum. In many cases the reference temperature is

property	expression	range (K)
density	$\rho = (11367 - 1.1944 \times T)\ \frac{Kg}{m^3}$	600-1700
viscosity	$\mu = 4.55 \times 10^{-04}\, e^{(1069/T)}\ Pa \cdot s$	600-1500
th. expansion	$\alpha_v = 14.77 \times 10^{-6} + 12.20 \times 10^{-9}$	
	$(T - 273.16) + 12.18^{-12}(T - 273.16)^2\ \frac{m^3}{K}$	600-1500
th. conductivity	$\kappa = 15.8 + 108 \times 10^{-4}\,(T - 600.4)\ \frac{W}{m \cdot K}$	600-1500
th. spec. heat	$C_p = 147.3\ \frac{J}{Kg \cdot K}$	600-1500

Table 3.5: Lead properties (600-1500K).

properties	value
Density $\rho_0(400C)$	10562 from Table 3.5
Viscosity $\mu_0(400C)$	0.0022 from Table 3.5
Thermal conductivity $\kappa_0(400C)$	16.58 from Table 3.5
Heat capacity C_p	147.3 from Table 3.5

Table 3.6: Lead properties at T=400C.

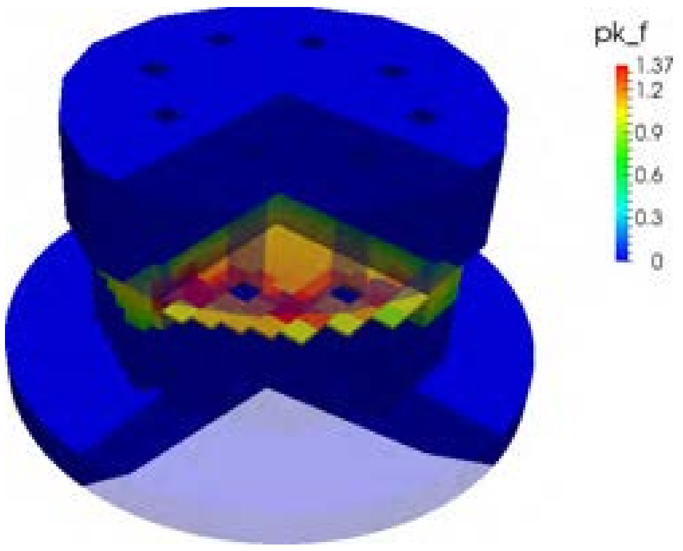


Figure 3.45: Test 2. Distribution of the peak factor pk_f for fuel assembly.

$T_{ref} = 400C$, the inlet temperature in standard working conditions. For the reference temperature T=400C the properties are computed in Table 3.6. In Figure 3.45 we show the distribution of peak factors for fuel assembly in Test 2 as they appear inside the FemLCore code.

In a similar way in Figure 3.46 the distribution of the peak factor for pressure losses are reported. We assume the existence of an inlet and outlet core grid which cause an important amount of pressure losses. We assume also four grids in the middle of the core to keep in place the fuel rods [24, 25, 26]. The accidental pressure losses are assumed as shown in Figure 3.46.

3.3.5 Coupling FemLCore-Cathare on Salome platform for Test 2

The coupling of FemLCore with the Cathare code is obtained in a very similar way as that described in Test 1. The project must be written in the *USER_APPL* directory of the FEMuS code [29]. The basic c++ files of the FemLCore code such as the Navier-Stokes and Energy equation solver together with Cathare links and FORTRAN files can be found in the *SRC* directory. The executable program is generated in this main directory while all the results both from FemLCore and Cathare are written in the *RESU* directory. Also for Test 2 the coupling supervisor, which is main.cpp, starts three problems: P (FemLCore), C (Cathare) and N problem (neutronics or heat sources).

The P problem is solved by a series of modules of the FemLCore code. For the core region a 3D-porous module is considered while for the two plenum regions a CFD module is used. Since the porous module is attached to the three-dimensional CFD module the equation are all rescaled to the reduced, or filtration velocity. For details see []. The *P problem* is defined by two parameter files: *parameters.in* and *param_files.in*. The appropriate *MGUtils*, *MGFE* and *MGGeomEl* classes must be initialized before the construction of the P problem. Once the utility classes (mgfemap(*MGFE*),mgutils[0](*MGUtils*), mggeomel(*MGGeomEl*)) are initialized the P problem can be set by the following standard code lines

```
FEMUS P;                         // constructor
P.init_param(*mgutils[0]);       // init parameter
P.init_fem(*mggeomel,*mgfemap);// init fem
```

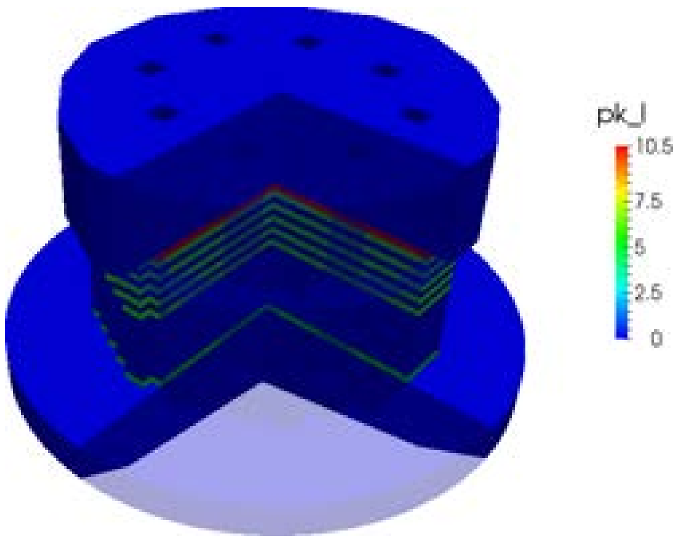


Figure 3.46: Test 2. Distribution of the peak factor pk_l for pressure losses.

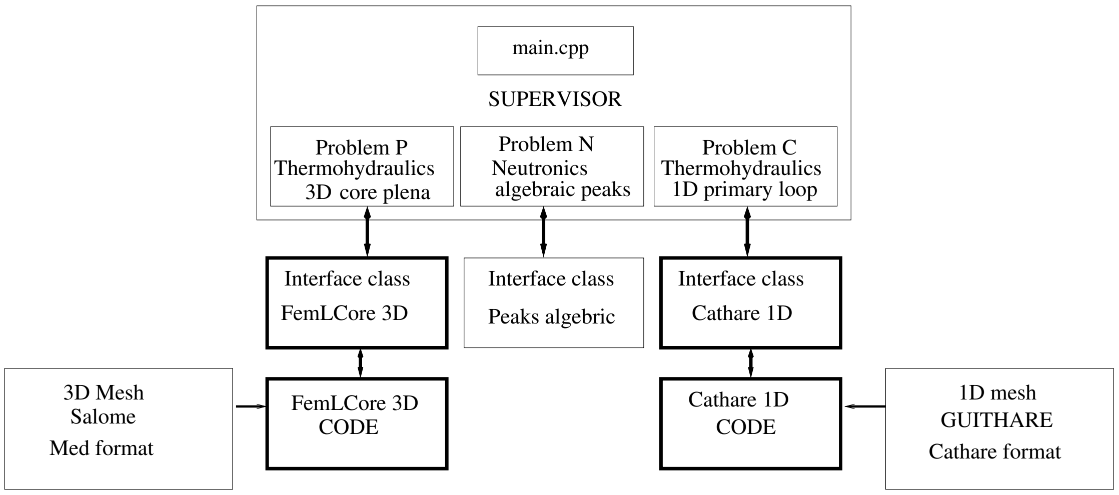


Figure 3.47: Test 2. Schematic of the coupling.

```
  // setting mesh ------------------------------------------
  P.setMesh();                   // set MGmesh
  // setting system ----------------------------------------
  P.setSystem(myproblemP,0,2);   // set system 2 cell vector
  // setting inital condition at t=0 -----------------------
  P.solve_setup(itime_0,time);   // initial time loop (t=0)
```

As one can note all the *FEMuS* problems can run with the same command lines. The difference between Test 2 and 1 is only in the handling of the boundary conditions and coupling interfaces. When the mesh in *MED* format is loaded in the FemLCore mesh structures two copies of the same mesh in different formats are allocated in the interface FEMuS class. This allows the transfer of data from the code to the Salome platform and viceversa. In Table 3.7 a brief list of marked surfaces. In Figure 3.48 The corresponding between the number and the location can be seen. In order to exchange data from the three-dimensional simulation to the mono-dimensional one we need to define interface in the P Problem. The interfaces are created by writing

```
P.init_interface(120,1200, 2,"mesh_file");// top temperature
P.init_interface(110,1100, 2,"mesh_file");// top temperature
P.init_interface(121,1200, 2,"mesh_file");// top pressure
P.init_interface(111,1100,1,"mesh_file"); // top pressure
P.init_interface(400,4000, 2,"mesh_file");// bottom temperature
P.init_interface(401,4000, 2,"mesh_file");// bottom velocity
P.init_interface(402,4000,1,"mesh_file"); // bottom pressure
```

From Figure 3.48 we can see the location of the groups defined from mesh generator. With the first and second line the groups 1200 and 1100 are associated with outlet of the reactor. We recall that in this reactor we have multiple exits. However on the top

location	name	component
inlet	4000	Lower Plenum
outlet 1	1200	Upper plenum
outlet 2	110	Upper plenum
top reactor	2100	Upper plenum
bottom	4100	Lower plenum
wall plenum 0	4200	Lower plenum
wall plenum 1	1300	Upper plenum
wall plenum 2	1400	Upper plenum
wall plenum 3	1500	Upper plenum
ctrl rods 1	200	core
ctrl rods 2	300	core
ctrl rods 3	400	core

Table 3.7: Test 2. Table of geometry and mesh groups.

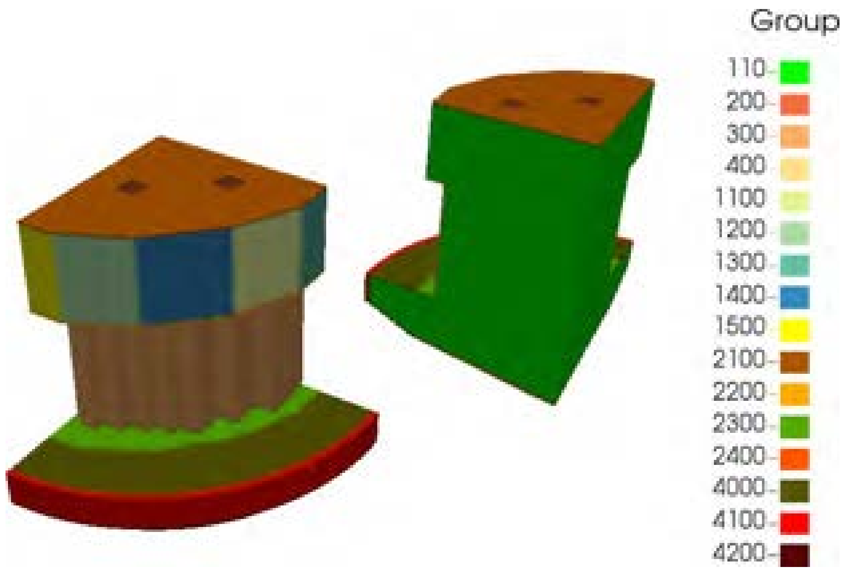


Figure 3.48: Test 2. Geometry and main mesh groups.

surface we need to transfer data for the temperature and pressure fields. Therefore the interfaces with name 120 and 110 are created on the mesh groups 1200 and 1100 for quadratic temperature field while the interface with name 111 and 121 are created on the mesh groups 1200 and 1100 for linear pressure field. The bottom surface is the inlet of the reactor where we need to exchange temperature, velocity and pressure. For this reason three interfaces labeled 400, 401, 402 are defined over the same mesh geometric group 4000.

The function *setSystem(myproblemP,0,2)* initializes the system NS_F and T_F and allocates two piecewise-constant external fields. These external fields can be used by a neutronic code with the appropriate Salome interface. The generation of the heat source is called *N Problem*. Since we are interested in the reactor behavior with fixed fuel composition the interface with a neutronic code is substituted by a simple interface that reads data from a database. This simple interface is defined by the *ReactData* class. In order to define the *N Problem* we write

```
  ReactData *N;
  N = new ReactData();
  N->datagen(P.get_MGExtSystem());
```

N is therefore the interface that controls the data flow for the heat and pressure loss sources inside and out of the reactor.

The *C problem* is a Cathare problem. The interface class is called *Problem_Cathare*. The initialization of this class can be done simply by

```
  Problem_Cathare* c = new Problem_Cathare();
  c->initialize();
```

The interface points for the Cathare problem can be defined as in Table 3.1. The

location	name	component	position (ix)	rad (irad)
inlet	point 1	LOOP	40	0
outlet	point 2	LOOP	1	0
pump	pump pt	LOOP	10	0
exchanger	heat_ex pt	LOOP	15	0

Table 3.8: Test 2. Cathare key interface points.

point 1 is located at the inlet of the Lower plenum, which is the outlet of the Cathare external loop, with coordinates $x = ix\_1$ on the LOOP module. In order to transfer the Cathare values to the three-dimensional code it is important get all the variables with the following Cathare interface command

```
rho_1=c->getValue_Cathare("LIQDENS","LOOP",ix_1,0);
f_1=c->getValue_Cathare("LIQFLOW","LOOP",ix_1,0);
P_1=c->getValue_Cathare("PRESSURE","LOOP",ix_1,0);
h_1=c->getValue_Cathare("LIQH","LOOP",ix_1,0);
T_1=c->getValue_Cathare("LIQTEMP","LOOP",ix_1,0);
v_1=c->getValue_Cathare("LIQV","LOOP",ix_1,0);
```

With *LIQDENS*, *LIQFLOW*, *PRESSURE*, *LIQH*, *LIQTEMP* and *LIQV* as arguments this function returns the value of density, mass flow, pressure, enthalpy, temperature and velocity, respectively.

The pump and the heat exchanger are set in the *LOOP* module. The pump model, which is located at the coordinate $x = ix\_p$, is very simple. In the pump point we impose a constant pressure with

```
 c->setValue_Cathare("DPLEXT","LOOP",ix_p,0,DP);
```

where *ix_p* and *irad* are defined as in Table 3.8. $DP = .75 \times 10^5$Pa is the pump gain in pressure. The heat exchanger is located on the LOOP module at the point with coordinates $x = ix\_h$, see Table 3.8. The heat exchanger can be modeled by the following code lines

```
T_ex=c->getValue_Cathare("LIQTEMP","LOOP",ix_h, 0);
h_ex=c->getValue_Cathare("LIQH","LOOP",ix_h, 0);
h_ex1 = h_ex -w_ex*(cp*(T_ex-T_ex2));
c->setValue_Cathare("ENTLEXT","LOOP",ix_h,0,h_ex1);
```

The cooling temperature T_{ex2} is set to $400C$. The enthalpy h_{ex} is determined iteratively to the corresponding value of the cooling temperature T_{ex2}. The constant w_{ex} is the feedback constant or the underelaxation constant in the iterative algorithm.

At the point 1 and point 2 defined in Table 3.8 we have the 3D/1D interfaces. At the point 2 the three-dimensional state must pass the temperature value to the Cathare code. The mono-coordinate of the Cathare code of the point 2 is labeled ix_2. In order to set the temperature we use a feedback algorithm. Let T_{outr} be the average temperature that flows out from the three-dimensional reactor and t_{21}/h_{21} be the temperature/enthalpy on the other side of the interface on the Cathare mono-dimensional mesh (point 2). The temperature t_{21} is corrected until it reaches T_{outr} by these code lines

```
t_21=c->getValue_Cathare("LIQTEMP","LOOP",ix_2,0;
h_21=c->getValue_Cathare("LIQH","LOOP",ix_2, 0;
h_12 = h_21-(cp*(t_21-T_outr));   // hentalpy correction
c->setValue_Cathare("ENTLEXT","LOOP",ix_2, 0,h_12);
```

The pressure coupling is obtained by imposing the three-dimensional average pressure drop $P[0] - P0[0]$ to the mono-dimensional part of the circuit between the point 1 and point 2. In this way we correct the mono-dimensional drop with the correct one. We write the following code lines

```
 press_p2=c->getValue_Cathare("PRESSURE","LOOP",ix_2,0);
 press_p1=c->getValue_Cathare("PRESSURE","CORE",ix_1,0);
 dp_ioo=c->getValue_Cathare("DPLEXT","CORE",2,0);
 dp_io=dp_ioo-0.5*(fem_rho*(h*g+averageP[0]-averageP0[0])-
                                (press_p1-press_p2));
 dp_io=dp_ioo;
 c->setValue_Cathare("DPLEXT","CORE",2,0,dp_io);
```

On the inlet section of the reactor we need to set temperature and velocity field. Since we do not know the field distribution at the reactor we assume a constant distribution. The temperature T_1 from the point 1 of Cathare should be imposed to the inlet reactor. We write

```
// Temperature coupling  T_1(Cathare) with interface 11(bottom)
std::ostringstream a; a<< T_1;
P.setAnalyticBoundaryValues(400,1, a.str().c_str());
P.write_Boundary_value(11,"T",1);// write inside x_old sol
```

and for the velocity field we write

```
// velocity coupling  v_1(Cathare) with interface 21 (bottom)
std::ostringstream vel; vel<< "IVec*0.+JVec*0+KVec*"<< v_1;
P.setAnalyticBoundaryValues(401,3, vel.str().c_str());
P.write_Boundary_value(21,"NS0",3); // write inside x_old
```

The boundary conditions for temperature and velocity are imposed directly by using the FEMuS interface functions *write_Boundary_value* directly on the old solution vector *x_old*.

The reactor temperature T_{outr} on the outlet is compute as average $0.5(T_{o1}+T_{o2})$ on the surface 110 and 110 by using the FEMuS interface command *getAvOnBoundary_nodes* as

```
T_o1=P.getAvOnBoundary_nodes(110, "T",0);
T_o2=P.getAvOnBoundary_nodes(120, "T",0);
```

In a similar way the average pressure at the outlet over the interface 111 and 121 can be taken as

```
P_o1 = P.getAvOnBoundary_nodes(111, "NS0",DIMENSION);
P_o2 = P.getAvOnBoundary_nodes(121, "NS0",DIMENSION);
P_in = P.getAvOnBoundary_nodes(402, "NS0",DIMENSION);
```

The average value $0.5(P_{o1} + P_{o2})$ can be storage as reference top pressure $P[0]$ for the tree-dimensional reactor. At the bottom we have a single face and therefore the three-dimensional reference pressure at inlet is $P0[0] = P_{in}$ that can be computed as

```
 P_{in} = P.getAvOnBoundary_nodes(402, "NS0",DIMENSION);
```

the pressure at the outlet $P[0]$ is set to P_{in}.

3.3.6 Field solutions

In order to compute the solution at working condition we start the computation from uniform initial condition v = 0.0m/s and temperature T = 400C. We apply an inlet velocity of 0.1m/s and the value of the pump pressure DP= .75 × 10^{5}Pa. The pressure

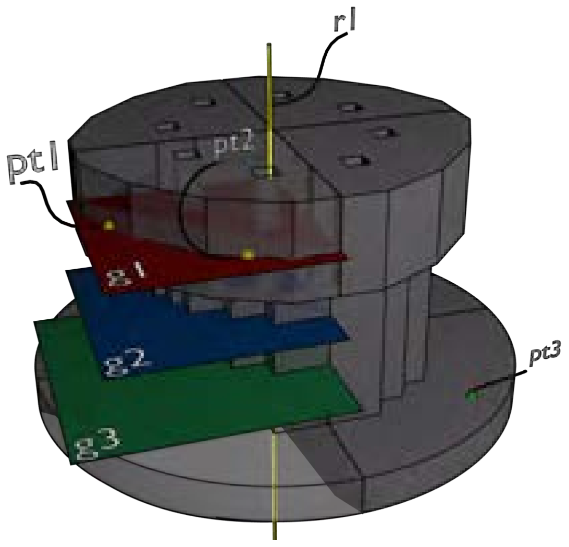


Figure 3.49: Test 2. Global domain overview and reference elements

from the pump increases the velocity until this matches the total pressure losses of the circuit with velocity and temperature transient. The transient is rather long and CPU consuming. Even if Cathare cannot be run in parallel in this configuration the FemLCore problem can be run in a multiprocessor cluster. In this way each processor runs a Cathare copy. This is not very efficient but the Cathare solving time can be considered almost negligible in comparison with the three-dimensional code solver.

However FemLCore and Cathare codes can run independently with a different number of time steps. If one is interested only the steady state solution in working condition the code cathare can be run until the velocity has reached the steady state saving a lot of the computational time.

In the next sections we report the results for the three and mono-dimensional domain when the steady state is reached. Also we report the evolution of the key points, shown in Figure 3.49 defining the interface in order to see the numerical behavior of the feedback control at the single point and of the defective coupling approach described in the previous section.

Three-dimensional FemLCore field solutions. The three-dimensional solutions are obtained by solving the FemLCore modules (P problem). The boundary conditions are enforced by the Cathare mono-dimensional code at the inlet of the reactor. The real distribution of the velocity and temperature at the inlet is a two-dimensional profile that cannot be known. Only the average value of temperature and velocity fields are defined and therefore we impose a constant field.

The pressure solution is reported in Figures 3.50–3.51. The pressure solution p in a

Navier-Stokes system can be always written as

$$p = -\rho g(z - z_0) + p' \tag{3.3}$$

by including the potential of the gravity term inside the pressure gradient. ρ is the density, g the gravity acceleration, z the vertical coordinate and z_0 the reference pressure point. The quantity $p' = dp$ is called partial pressure and it is defined as the difference between the total pressure and the pressure gravity term. Over the 3D/1D coupling interface the definition of the location of z_0, where the reference pressure should be imposed, is not a trivial problem. The reactor takes the pressure imposed from the Cathare code at the outlet of the $AXIAL$ CORE module. Since the outlet of the reactor is over different values of the coordinate zeta it is difficult to impose constant pressure over all the outlet surfaces. The decomposition in (3.3) solves this problem fixing the pressure at a well defined height. On the Figure 3.50 the partial FemLCore

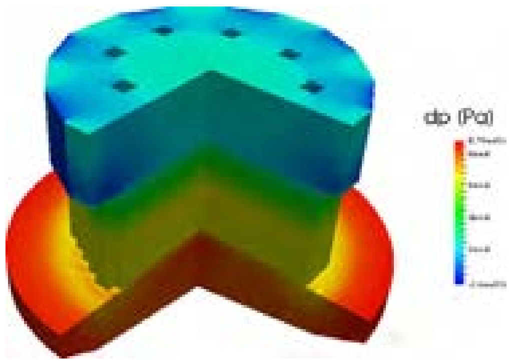


Figure 3.50: Test 2. FemLCore partial pressure field in the three-dimensional domain.

pressure field $dp = p'$ in the three-dimensional domain is shown. In Figure 3.51 details on the same pressure field over different planes for $z = 2$, 3 and 3.4 are reported.

On the Figure 3.52 the FemLCore temperature field T is shown over the whole three-dimensional domain. In Figure 3.53 one can see details on the same temperature field over horizontal sections for $z = 2$, 3 and 3.4 the planes shown in Figure 3.49. Inside the three-dimensional reactor the temperature increases and its pattern over these sections is clearly determined by the fuel distribution ($z = 2$).

The velocity field inside the reactor is not continuous since we have the plenum and the core that satisfy different form of the Navier-Stokes equations. In the core the assembly contains fluid and solid material. The geometric dimensions are real, comprehensive of the solid material, while in the simulation we allow the fluid to flow to occupy the whole assembly section. For this reason in order to maintain the same mass flow in the core and in the plenum, over different flowing area the velocity field should be discontinuous. In order to have a continuous variable to solve we consider the normalized velocity field $\boldsymbol{v}^* = r\boldsymbol{v}$ where r is the assembly occupancy factor. Substantially we rewrite the Navier-Stokes equation with the mass flow variable. On the Figure 3.54 the vertical component w^* of the FemLCore normalized velocity field $\boldsymbol{v}^*$ is shown over the whole

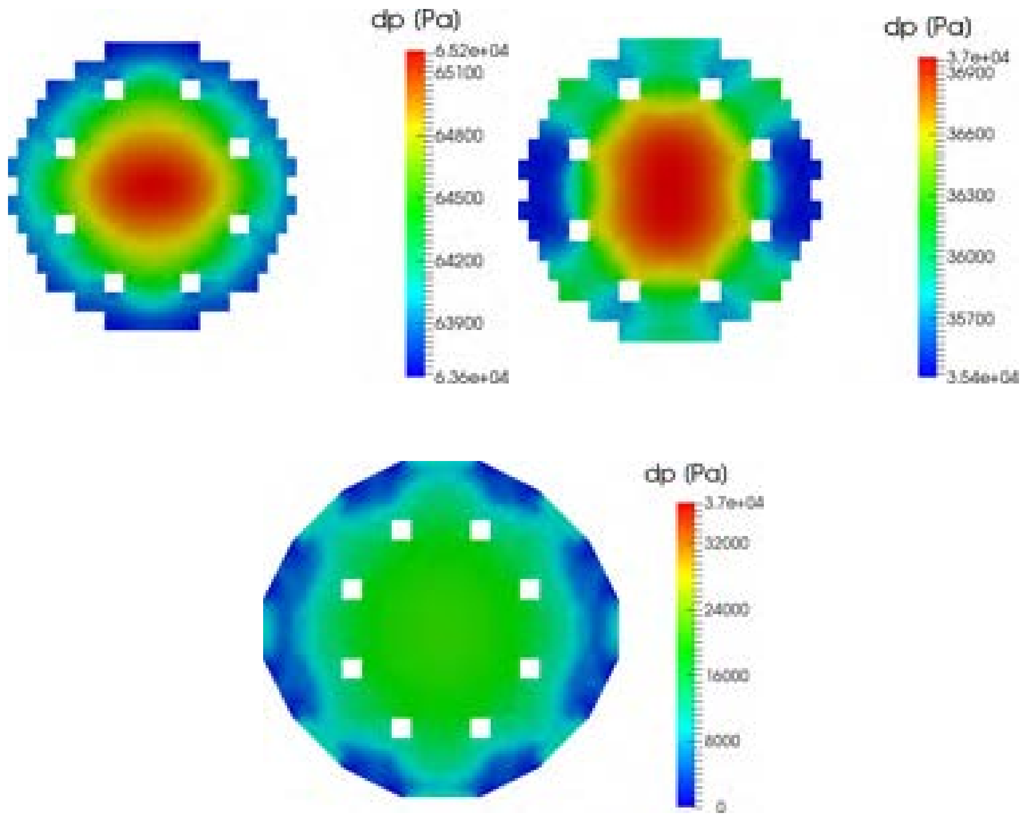


Figure 3.51: Test 2. FemLCore partial pressure field over different planes for $z = 2$, 3 and 3.4.

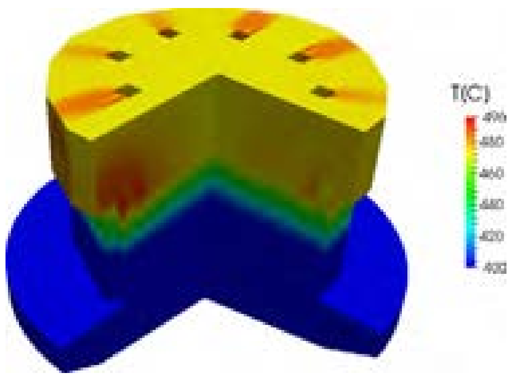


Figure 3.52: Test 2. FemLCore temperature field in the three-dimensional domain.

three-dimensional domain. In Figure 3.55 one can see details on the same w^* field over horizontal sections for $z = 2$, 3 and 3.4.

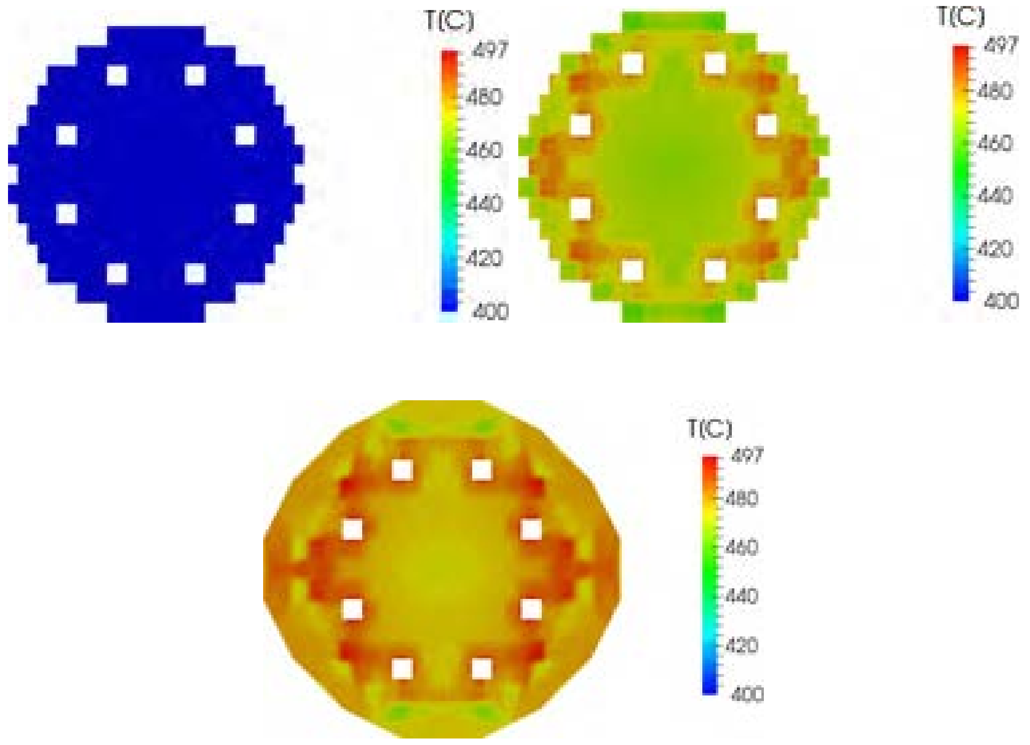


Figure 3.53: Test 2. FemLCore temperature field over different planes for $z = 2$, 3 and 3.4.

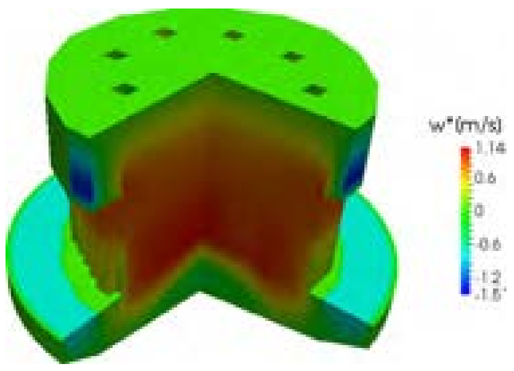


Figure 3.54: Test 2. FemLCore normalized velocity field w^* in the three-dimensional domain.

Mono-dimensional solution over the reactor circuit. We show the results for the mono-dimensional mesh and for the whole circuit. In Figure 3.56 the key points of the mono-dimensional mesh are shown. In red the UPLENUM Cathare *VOLUME* module and in blue the LOOP *AXIAL* module. The CORE *AXIAL* module is in black. In Figure 3.57 the pressure along the line $r1$ of Figure 3.49, is shown on the left. In

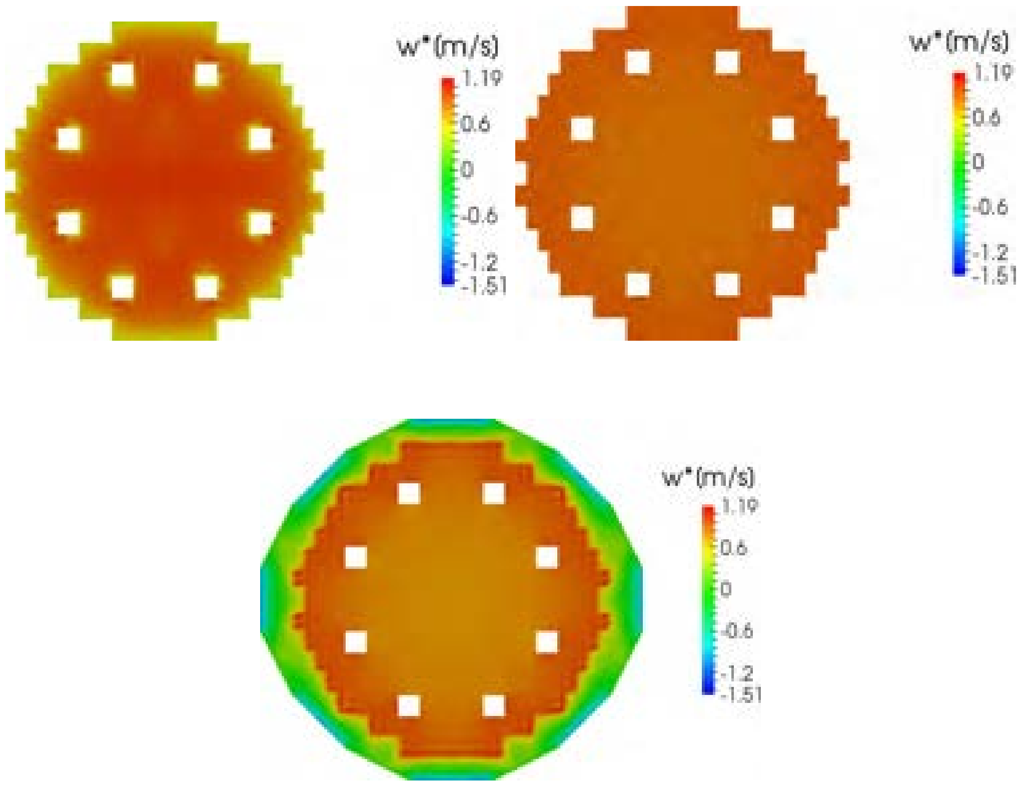


Figure 3.55: Test 2. FemLCore normalized velocity field w^* over different planes for $z = 2$, 3 and 3.4.

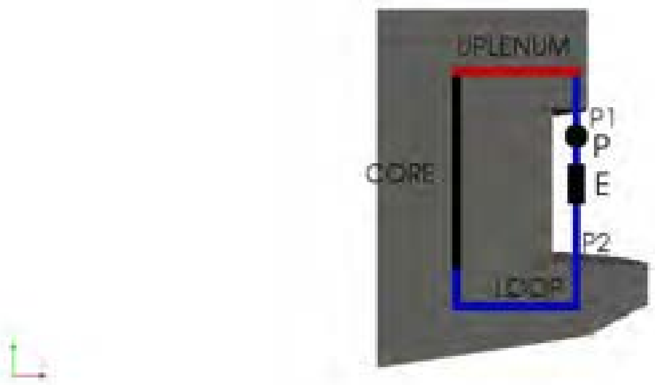


Figure 3.56: Test 2. Key points in the mono-dimensional mesh.

this Figure we report the partial pressure p', namely the pressure without the gravity

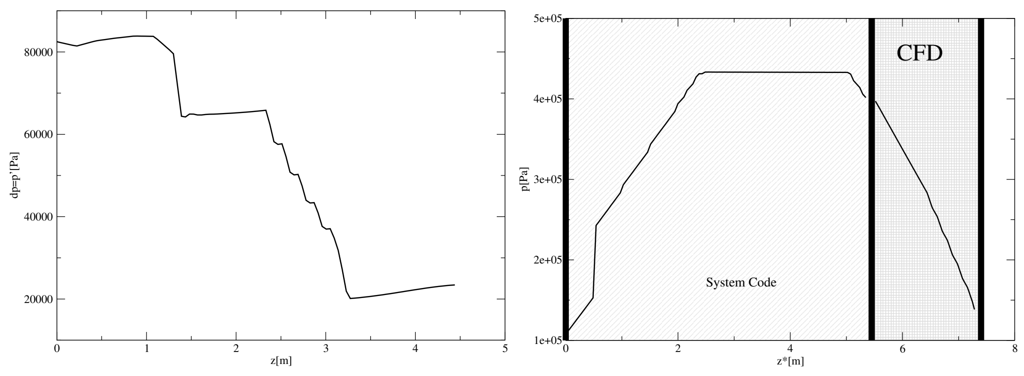


Figure 3.57: Test 2. Total pressure p^* along the central line of the three-dimensional domain (left) and along the plenum-core-plenum-primary loop (right).

contribution. The gravity term is dominant and, if shown, it covers the accidental pressure losses. We can see two large jumps in pressure due to the inlet and outlet grids. In particular the inlet grid has also the duty to steer the flow in the vertical direction. It is possible also to see the pressure losses of the four internal grids. On the right the total pressure in the circuits is shown. The mono-dimensional coordinate starts at the point P_1, defined in Figure 3.56, enters the reactor lower plenum at $x = 1.95m$ and levels horizontally from $2.35m$ and $5m$. At $x = 5.4m$ enters the core and reaches the upper plenum at $7.35m$. Between $x = 1.95m$ and $7.35m$ this mesh overlaps with the three-dimensional mesh. On the right of Figure 3.57 the initial pressure at $x = 0m$ is imposed by the BCONDIT Cathare module and increases linearly due to the gravity term. The pressure has a jump, defined by the pump, and then it starts to increase linearly again. At the bottom the pressure line is almost horizontal since only the distributed pressure losses are taken into account. In the last part the circuit reaches the core where the mono-dimensional pressure behavior is substituted with the three-dimensional one. On

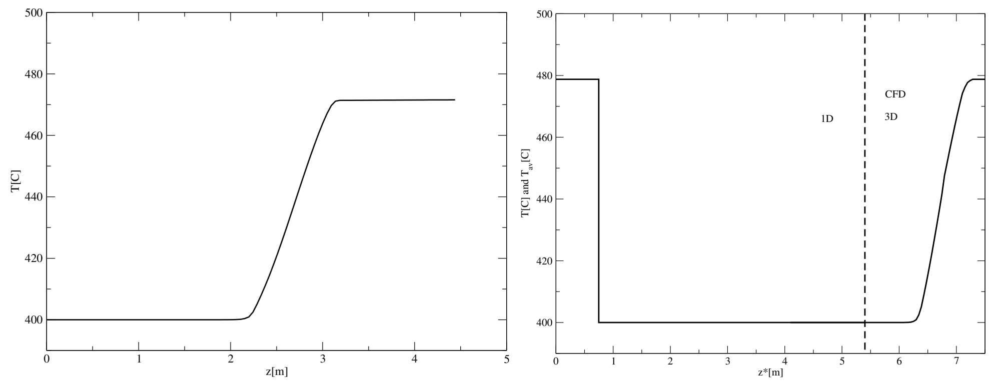


Figure 3.58: Test 2. Temperature along the central line of the three-dimensional domain (left) and average temperature along the plenum-core-plenum-primary loop (right).

the left of Figure 3.58 the temperature along the central line of the three-dimensional domain is shown. On the right the average temperature along the circuit is reported. The average temperature at exit of the the reactor is about $478C$. The fluid flow goes through the pump and the heat exchanger. In the exchanger the temperature drops to 400C. The temperature remains the same until it reaches the core. In the core we plot the average temperature of the three-dimensional simulation. Finally the normalized

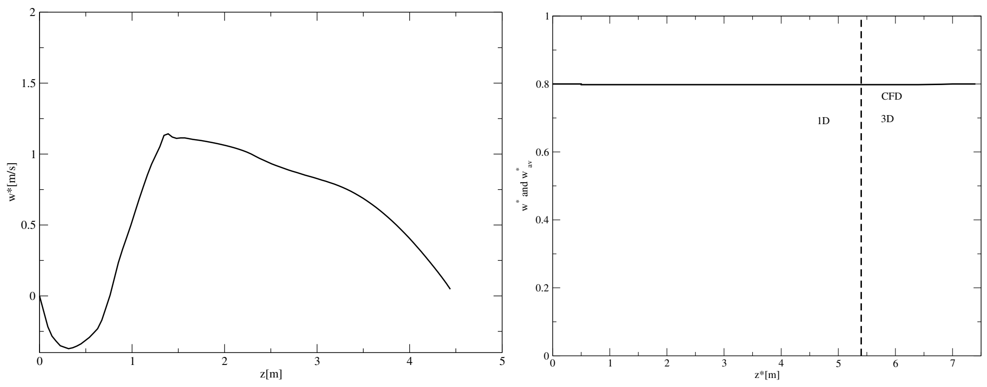


Figure 3.59: Test 2. Normalized velocity along the central line of the three-dimensional domain (left) and average normalized velocity along the plenum-core-plenum-primary loop (right).

average velocity is shown in Figure 3.59. On the left the normalized vertical velocity for a case of open assembly is reported. On the left the normalized velocity on the circuit and its average on the three-dimensional domain are shown.

Solution at the coupling interfaces. In Figures 3.61–3.60 pressure, temperature and the velocity field at the 1D/3D interfaces are shown as a function of time when no internal iterative algorithm is used. Since we use a defective coupling approach every state variable is matched by using sources in the mass, momentum and energy equation. These sources are in form of feedback terms, namely

$$S_i^{k+1} = S_i^k + w_i(\phi_{3D} - phi_{1D})\,, \tag{3.4}$$

where k indicates the k-step of the iterative algorithm. The index i indicates the equation where the source is applied ($i = 0$ mass, $i = 1$ momentum, $i = 2$ energy). The under-relaxation parameter w_i can be tuned to get fast convergence. The state variable from the mono-dimensional solution is indicated as ϕ_{1D} and for the three-dimensional domain as ϕ_{3D}. For the k that tends to infinity S_i^k tends to S_i that implies $\phi_{3D} = phi_{1D}$. However if a steady solution is considered the number of iterations for this feed back control algorithm can be set to one. In this case we say that no internal iterative algorithm is used. In Figure 3.60 the evolution of the state variables at the interface 1D/3D P_2 are shown. On the top we have the pressure and on the bottom the average temperature and velocity. The interface P_2 is the inlet of the reactor. Over this interface there is no need

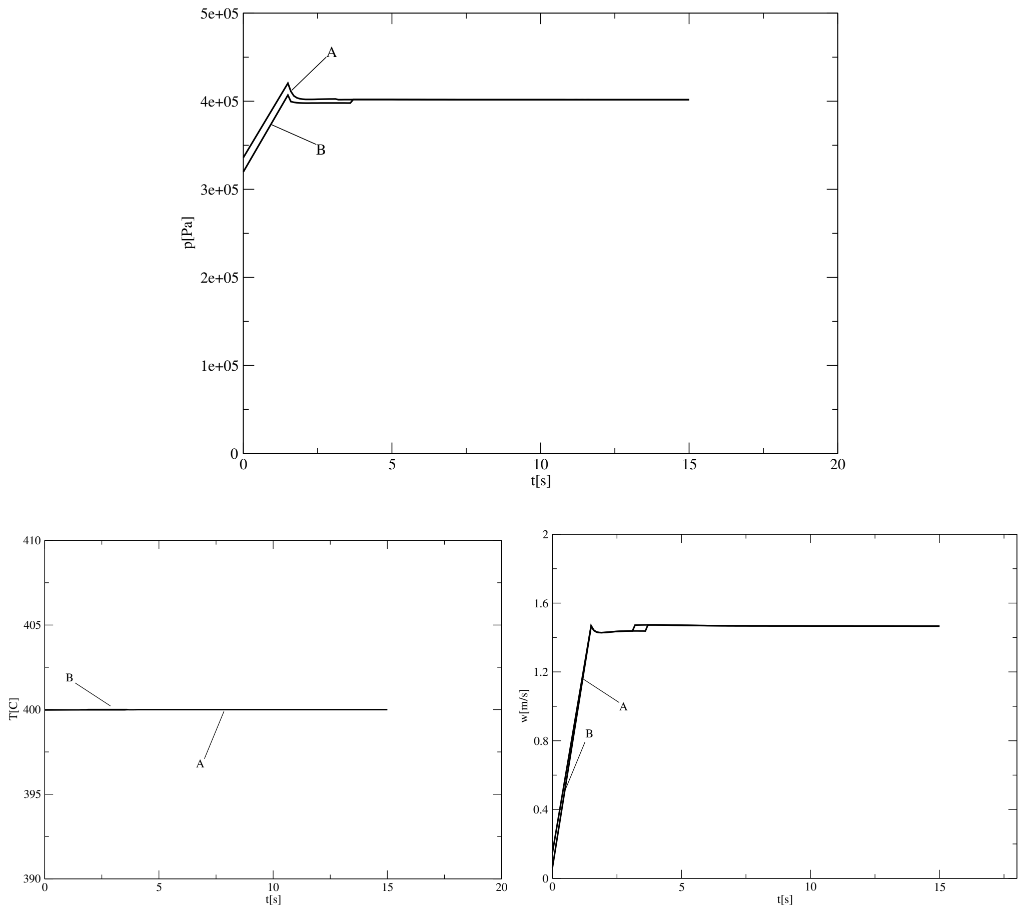
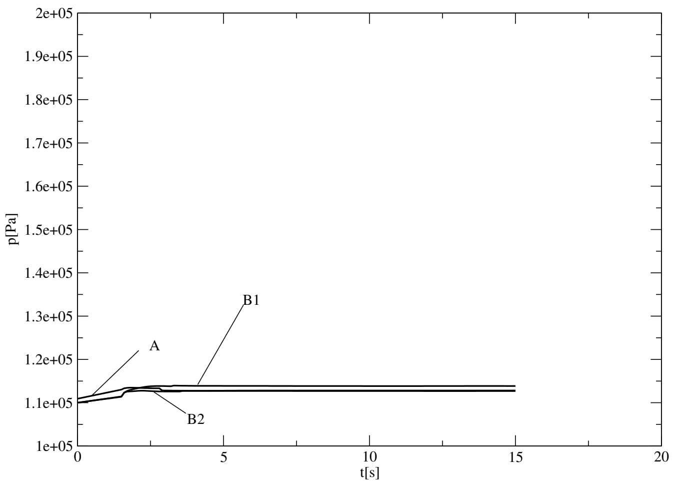


Figure 3.60: Test 2. Evolution of the state variables at the interface 1D/3D P_2: circuit (A) and three-dimensional (B) solution values.

of the feedback algorithm on temperature and mass flow since they are simply boundary conditions imposed from the circuit to the three-dimensional simulation. During the evolution, the three-dimensional values follow the transient determined by the pump. However the pressure is determined with the feedback control algorithm. As one can see the matching is almost perfect. The difference is due to the fact that the transfer of data in these two codes is passed with one step delay. In Figure 3.61 the evolution of the state variables at the interface 1D/3D P_2 is shown. The state variables are pressure on the top and average temperature and velocity on the bottom. The interface P_2 is the outlet of the reactor. The reactor has multiple exits which are the points $pt1 - pt2$ of Figure 3.49, where the state variables takes no constant values. The value imposed to the mono-dimensional simulation is imposed by using sources located in the CORE *AXIAL* module that overlaps the three-dimensional domain. The pressure is set quickly at the beginning of the evolution. The two exits B_1 and B_2 and the mono-dimensional inlet keep the fixed pressure dictated by the BCONDIT module. At $t = 0$



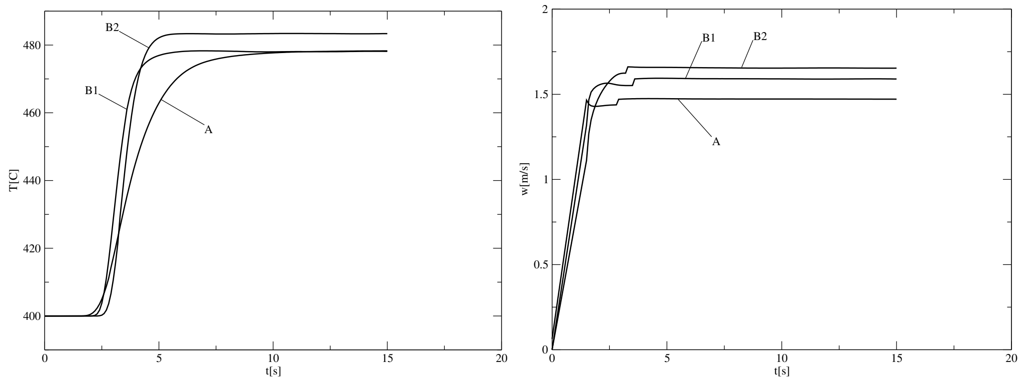


Figure 3.61: Test 2. Evolution of the state variables at the interface 1D/3D P_1: circuit (A) and three-dimensional (B) values.

the temperature is $T = 400C$. This point starts to feel the increasing in temperature $t \approx 3$s when the reactor warm up. The average temperature increases till reach its steady value. Two points in two different exits B_1 and B_2 are shown together with the average temperature imposed to the circuit. The same happens for the velocity. The pump pressure defines the circuit flow rate which increases till it reaches the steady state at the mean velocity of approximately 1.48m/s. It is worth while to note how smooth is the numerical convergence at the interface when the coupling defective approach is used. If compared with the simple coupling boundary approach used in previous reports we can say that this approach is more computational efficient.

4 Conclusion

In this report we have presented the development of the FemLCore-Cathare code coupling. We have developed two interface classes: one for FemLCore modules and another for the Cathare code. The data transfer functions for the Cathare code have been adapted from the *ICoCo* class developed by the Cathare team. This Cathare interface is simple and does not rely on a specific *TrioField* type as in the *ICoCo* class. This class can print the Cathare variable state in XDMF-HDF5 format that allows us to see solutions at each time step during the code execution [35]. Great difficulties arise from the complexity of combining the three and the mono-dimensional mesh together. The input-output formats for meshes and solutions have been adapted to be compatible. Data are exchanged at each time step through a supervisor code. In order to use the supervisor both codes must be written as shared libraries. The Cathare interface is therefore also a wrapper class to the Cathare Fortran routines. Furthermore the FemLCore can run in parallel by using the PETSC libraries while a copy of Cathare code runs for each processor [33].

Two tests have been performed. In the first test a simple three-dimensional core model reactor and a primary mono-dimensional loop, which consists of a pump and a heat exchanger, have been considered. In the second test a three-dimensional CFD-porous reactor with core and lower-upper plenum have been coupled with a primary mono-dimensional loop. The defective coupling approach has been shown to be a very effective numerical tool. The continuity of the velocity and temperature fields have been imposed by using feedback control algorithms in the Cathare source of mass, momentum and energy equations. The convergence is smooth and does not present particular difficulties. This approach is particular interesting in evaluating the importance of the three-dimensional approach since this appears as correction to the mono-dimensional solutions. If the corrections are substantial then CFD tree-dimensional computations are necessary for the correct evaluation of the system code results. If the three-dimensional corrections are not substantial then the CFD code in only necessary to evaluate the state field in that particular region and can be neglected in the remaining part of the domain. FemLCore-Cathare coupling is an affective tool to reproduce the multiscale thermohydraulics. Once the interface on the Salome platform is created these codes can be coupled with other modules such as neutronic and structural codes.

Bibliography

[1] *Cathare: Installation manual*, report STMF_LMES_RT_12-041A, CEA

[2] *Cathare: Main New Features and Improvements in the CATHARE 2 V2.5_3 mod2.1 delivery documentation*, report STMF_LMES_RT_12-042A, CEA

[3] *Cathare: Dictionary of operators and directives*, report STMF_LMES_RT_12-040, CEA

[4] *Cathare: Dictionary of Result Post-processing*, report STMF_LMES_RT_12-039A, CEA

[5] *Cathare: User Guidelines*, report NT_SSTH_LDAS_2005-034_UG, CEA

[6] *Cathare: User's Manual*, report NT_SSTH_LDAS_2005-035_UM, CEA

[7] *ICoCo documentation*, CEA

[8] *ELSY Work Program. European Lead-cooled SYstem (ELSY) Project*, Technical report, EURATOM, Management of Radioactive Waste (2006).

[9] L. Cinotti, âĂIJReactor Assembly preliminary ConfigurationâĂİ, ELSY DOC 08 049, Del Fungo Giera Energia, (2008).

[10] A. Alemberti, J. Carlsson, E. Malambu, A. Orden, D. Struwe, P. Agostini, S. Monti, *European lead fast reactorâĂŤELSY*, Nucl. Eng. Design, 241, pp.3470–3480 (2011).

[11] X. Cheng, N.I. Tak, *CFD analysis of thermal-hydraulic behavior of heavy liquid metals in sub-channels*, Nucl. Eng. Design, 236, pp.1874–1885 (2006).

[12] I. Di Piazza, M. Scarpa, *Rassegna di Letteratura sulla Termoidraulica dei Bundle Refrigerati a Metallo Liquido Pesante*, ENEA Report LM-FR-001 (2012).

[13] A. Cervone and S. Manservisi, *A three-dimensional CFD program for the simulation of the thermo-hydraulic behavior of an open core liquid metal reactor*, Technical report lin-thrg 108-2008.

[14] S. Bnà, S. Manservisi and O. Le Bot, *Simulation of the Thermal-hydraulic behavior of Liquid Metal Reactors using a Three-Dimensional Finite Element Model*, Technical Report DIENCA-UNIBO 2010.

[15] F. G. Bornia, M. Finelli, S. Manservisi, V. Mikhin, M. Polidori and K. Voukelatou, *Development and validation of FEM-LCORE code for the thermal hydraulics of open cores*, Technical Report DIENCA-UNIBO 2011.

[16] F.Bassenghi, G.Bornia, L.Deon and S. Manservisi, *Implementation and validation of the NURISP platform*, Technical report CIRTEN-UNIBO 2011.

[17] G. Bornia, D. Cerroni, S. Manservisi, M. Polidori and F. Donato, *FEM-LCORE code: parallelization, turbulence models and code integration*, Technical Report ENEA-UNIBO 2012.

[18] D. Cerroni, S. Manservisi and E. Vincenzi, *Developing multiscale transient simulations with fem-lcore code*, Technical Report ENEA-UNIBO 2013.

[19] D. Cerroni, R. Da Vià, S. Manservisi, F Menghini and G. Pozzetti *Integration of the FemLCore code in the SALOME platform*, Technical Report ENEA-UNIBO 2014.

[20] F.Bassenghi, G.Bornia, A. Cervone and S. Manservisi, *The ENEA-CRESCO platform for simulating liquid metal reactor*, Technical report LIN-THRG 210-2010.

[21] L. Barucca, M. Gregorini, *DHR Main components Functional Sizing*, DOC/08/047, Ansaldo Nucleare, (2008).

[22] M. Bottcher, *CFX Analyses of the RVACS Decay Heat Removal System*, KIT/INR-09-01, Wissenschaftliche Berichte (2009).

[23] L. Cinotti, *Reactor Assembly preliminary Configuration*, ELSY-DOC08049, Del Fungo Giera Energia (2008).

[24] A. Onea, M. Bottcher, D. Struwe, *Analysis of the pressure loss through the heat exchanger for the primary side of the ELSY nuclear reactor*, KIT/INR-09-03, Wissenschaftliche Berichte (2009).

[25] A. Onea, M. Bottcher, D. Struwe, *Detailed CFD analysis of the pressure loss on the primary side for the heat exchanger of the ELSY fast lead-cooled reactor by applying unit slice models*, ASME-ATI-UIT, Sorrento, Italy (2010).

[26] A.Travleev, *ELSY core design static, dynamic and safety parameters with the open square FA* Appendix C: Preliminary Core AnalysisâĂİ,FPN-P9IX-006, ENEA (2009).

[27] R. N. Lyon, *Liquid-Metals Handbook*, 2nd ed., Atomic Energy Comm., Washington D.C. (1952)

[28] OECD/NEA, *Handbook on Lead-bismuth Eutectic Alloy and Lead Properties, Materials Compatibility, Thermal-hydraulics and Technologies*, OECD/NEA No. 6195, ISBN 978-92-64-99002-9 (2007)

[29] FEMuS (Finite element multiphisics library), *http://slipknots.ing.unibo.it/femus*

[30] MPI (Message Passing Interface) forum, *http://www.mpi-forum.org*

[31] OpenMPI library, *http://www.open-mpi.org*

[32] *PARAVIEW visualization software*, Official documentation at *http://www.paraview.org*

[33] PETSc (Portable Extension Toolkit for Scientific Computation), *http://www.mcs.anl.gov/petsc/petsc-as*

[34] CEA/DEN, *SALOME Documentation*, CEA/DEN, EDF R&D, OPEN CASCADE (2007–2008).

[35] *HDF5 library*, *http://www.hdfgroup.org/HDF5*

[36] *VTK library*, *http://www.vtk.org*

[37] LASPACK (Linear Algebra Sparse Matrix Package): *http://www.mgnet.org/mgnet/Codes/laspack/html/laspack.html*

[38] LIBMESH package: *http://libmesh.sourceforge.net*

List of Figures

List of Tables