Software architecture and code coupling in multi-physics and multi-scale framework

Christophe Kassiotis

Saint-Venant Laboratory for Hydraulics
christophe.kassiotis@enpc.fr

1st April 2012
Short Course at IASS-IACM 2012
GF Sarajevo

Fluid Structure Interaction

- Nearly every structure is surrounded by fluids
- Countless applications
- Among important issues: extreme winds or tsunami impacts on coasts

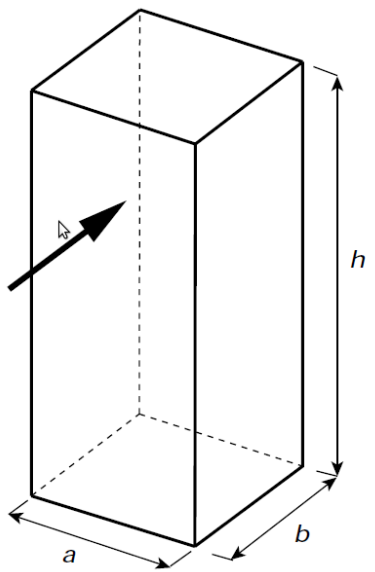
Fluid Structure Interaction

- Nearly every structure is surrounded by fluids
- Countless applications
- Among important issues: extreme winds or tsunami impacts on coasts

Wind action (Eurocode I, P 2.4)

- Elementary geometry: A_{ref}

$$F = p_{\text{ref}} C_e C_z C_d A_{\text{ref}}$$



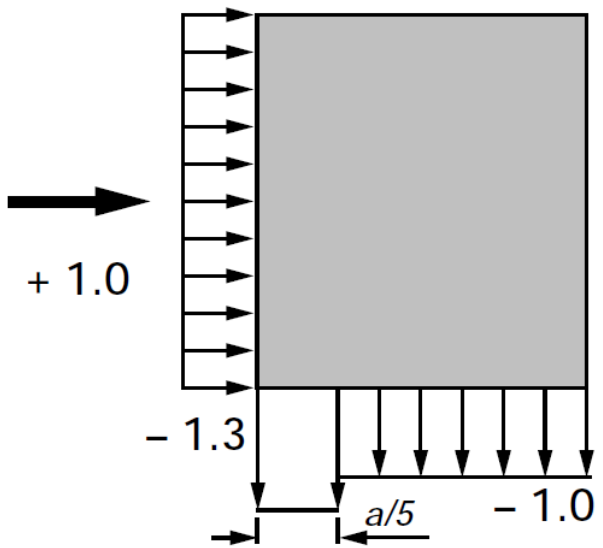
Fluid Structure Interaction

- Nearly every structure is surrounded by fluids
- Countless applications
- Among important issues: extreme winds or tsunami impacts on coasts

Wind action (Eurocode I, P 2.4)

- Elementary geometry: A_{ref}
- Simplified Force actions: $p_{\text{ref}} C_e C_z$

$$F = p_{\text{ref}} C_e C_z C_d A_{\text{ref}}$$



Fluid Structure Interaction

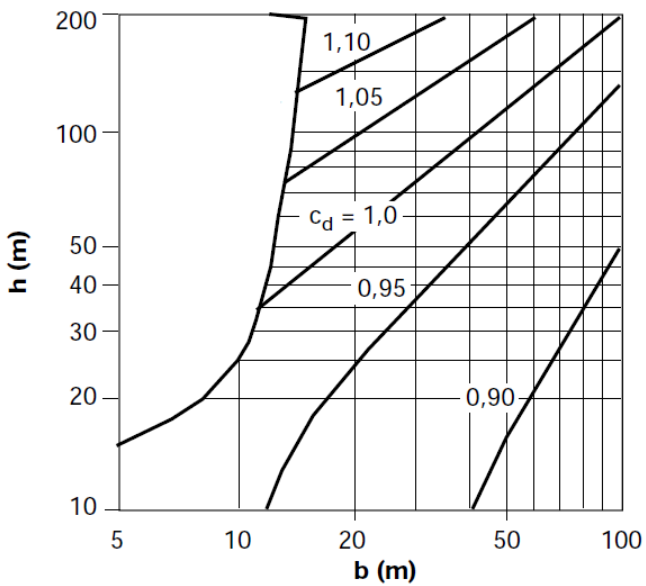
- Nearly every structure is surrounded by fluids
- Countless applications
- Among important issues: extreme winds or tsunami impacts on coasts

Wind action (Eurocode I, P 2.4)

- Elementary geometry: A_{ref}
- Simplified Force actions: $p_{\text{ref}} C_e C_z$
- Simplified Interaction wind / structure : C_d

Only the structure point of view

$$F = p_{\text{ref}} C_e C_z C_d A_{\text{ref}}$$

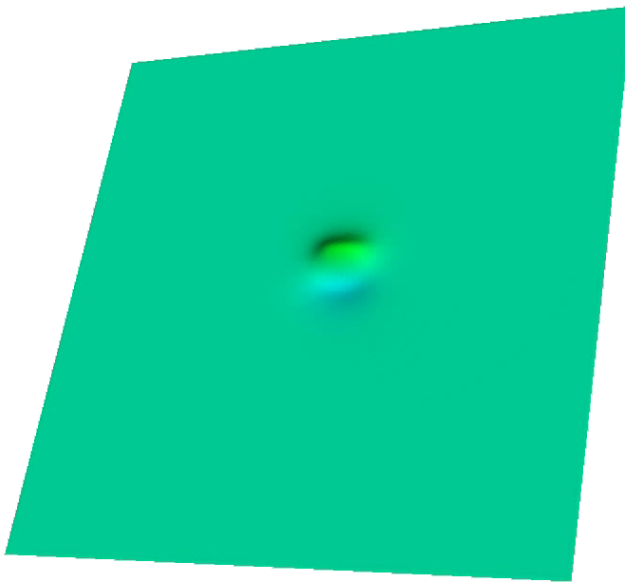


Fluid Structure Interaction

- Nearly every structure is surrounded by fluids
- Countless applications
- Among important issues: extreme winds or tsunami impacts on coasts

Tsunami modeling

- Generation



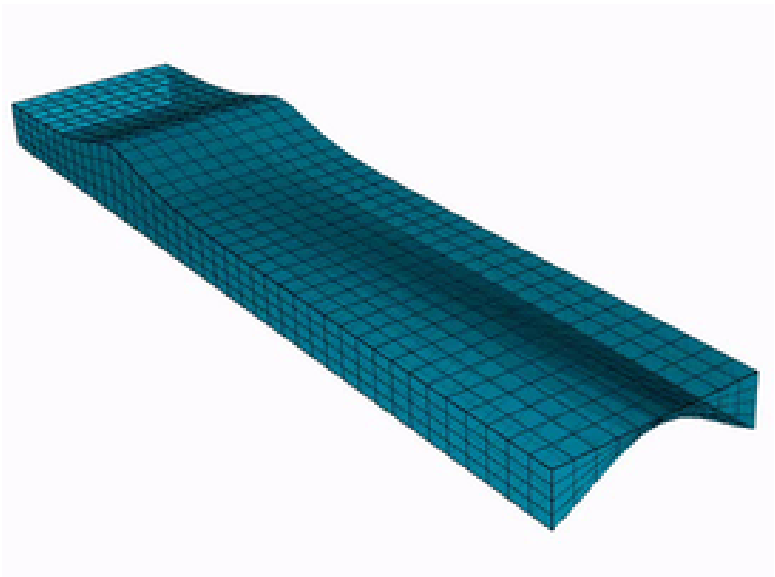
Source: CMLA-Cachan [Dutykh, 09]

Fluid Structure Interaction

- Nearly every structure is surrounded by fluids
- Countless applications
- Among important issues: extreme winds or tsunami impacts on coasts

Tsunami modeling

- Generation
- Propagation



[Kassiotis, 07]

Fluid Structure Interaction

- Nearly every structure is surrounded by fluids
- Countless applications
- Among important issues: extreme winds or tsunami impacts on coasts

Tsunami modeling

- Generation
- Propagation
- Run-up

Source: FBI (American Samoa office), Samoa, September 2009

Fluid Structure Interaction

- Nearly every structure is surrounded by fluids
- Countless applications
- Among important issues: extreme winds or tsunami impacts on coasts

Tsunami modeling

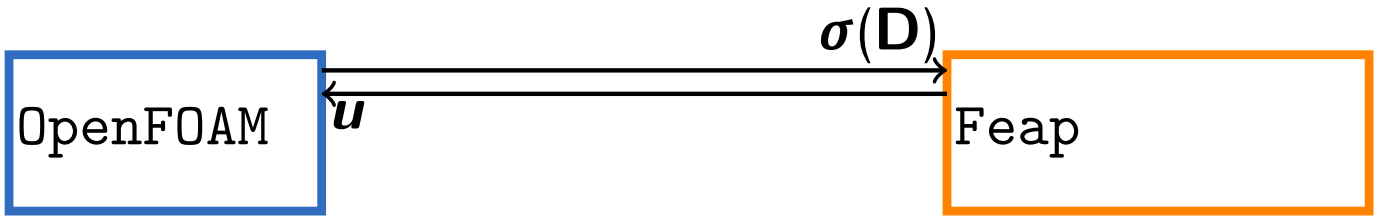
- Generation
- Propagation
- Run-up

Run-up key issues

- Amplitude of the flood
- Resistance of buildings

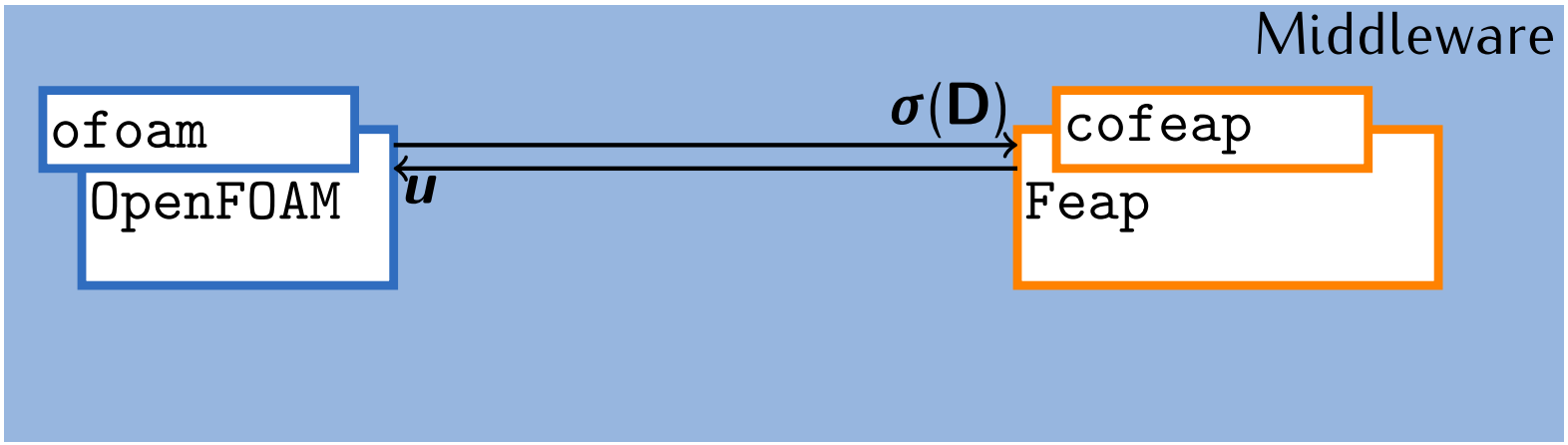
Source: FBI (American Samoa office), Samoa, September 2009

Introduction



Minimum requirement: a communication protocole

Introduction

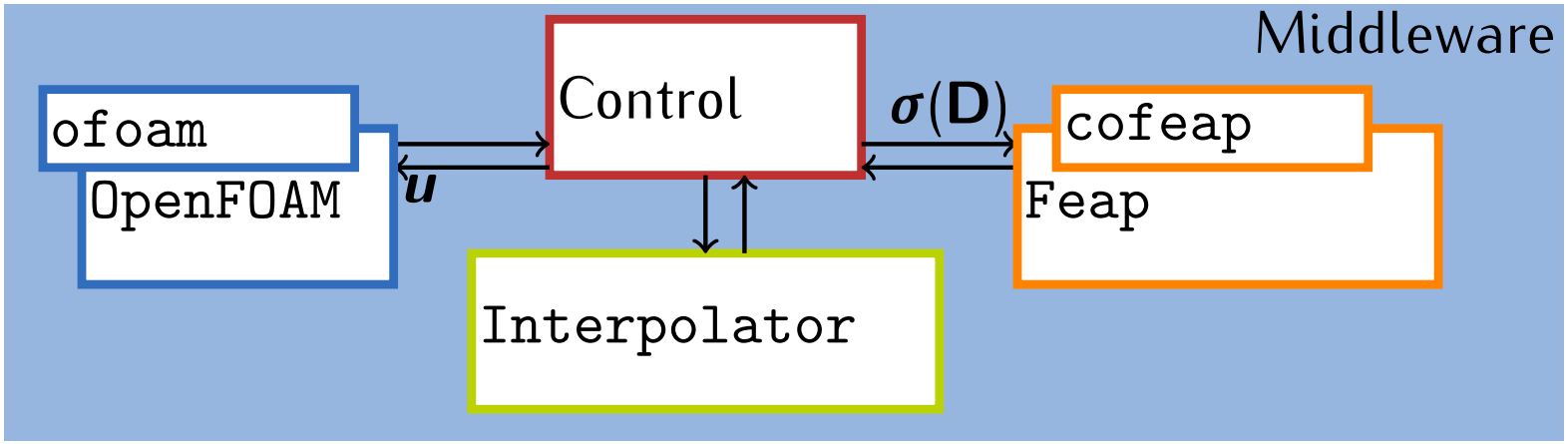


Minimum requirement: a communication protocole

Component technology [Mac Ilroy 68, Szyperski & Meeserschmitt 98]

- Each software communicates via the *middleware*
- Generalized OOP: encapsulation / interface + communication

Introduction



Minimum requirement: a communication protocole

Component technology [Mac Ilroy 68, Szyperski & Meeserschmitt 98]

- Each software communicates via the *middleware*
- Generalized OOP: encapsulation / interface + communication
- `cops` to organize the communication between fluid and structure
- `Interpolator` for non matching meshes

Implementation using Communication Template Library (CTL) [Niekamp, 02]

Plan

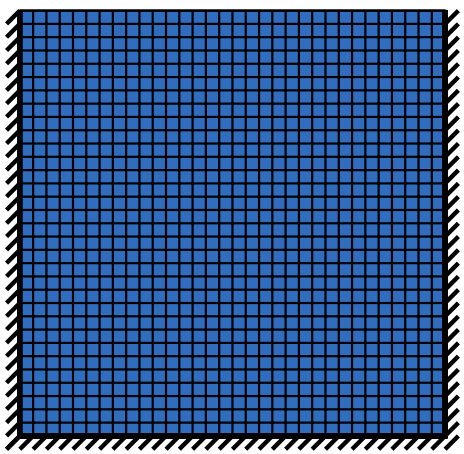
1. Running a coupled simulation
2. Coupling components
3. Building components

Running a coupled simulation

Test case presentation: lid-driven cavity with a flexible bottom

Convenient benchmark

- Mesh simplicity
- Computational time:
  $T_s^{\text{CPU}} = 2.95 \times 10^{-3} s$
  $T_f^{\text{CPU}} = 1.08 \times 10^{-1} s$
- Harmonic solution



Problem parameters

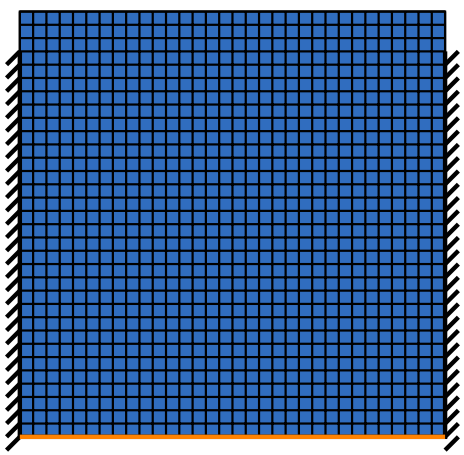
- Fluid problem $\rho_f = 1 kg.m^{-3}$, $\nu_f = 0.01 m \cdot s^{-2}$.
- Fluid boundary conditions:
  - only ∇p required
  - $\boldsymbol{v} \cdot \boldsymbol{e}_x = 1 - \cos(2\pi t / T_{\text{char}})$
- Structure problem:
  $\rho_s = 500 kg \cdot m^{-3}$, $E_s = 250 Pa$ *and* $\nu_s = 0$
- Discretization
  - Fluid: 32x32 cells for $\mathcal{R}e \leq 300$.
  - Structure: 16 quadratic elements.
  - Time step: $\Delta t = 0.1 s$.

Running a coupled simulation

Test case presentation: lid-driven cavity with a flexible bottom

Convenient benchmark

- Mesh simplicity
- Computational time:
  $T_s^{\text{CPU}} = 2.95 \times 10^{-3} s$
  $T_f^{\text{CPU}} = 1.08 \times 10^{-1} s$
- Harmonic solution



Problem parameters

- Fluid problem $\rho_f = 1 kg.m^{-3}$, $\nu_f = 0.01 m \cdot s^{-2}$.
- Fluid boundary conditions:
  - only ∇p required
  - $\boldsymbol{v} \cdot \boldsymbol{e}_x = 1 - \cos(2\pi t / T_{\text{char}})$
- Structure problem:
  $\rho_s = 500 kg \cdot m^{-3}$, $E_s = 250 Pa$ *and* $\nu_s = 0$
- Discretization
  - Fluid: 32x32 cells for $\mathcal{R}e \leq 300$.
  - Structure: 16 quadratic elements.
  - Time step: $\Delta t = 0.1 s$.

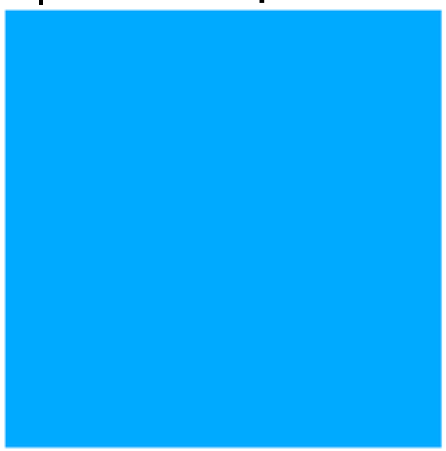
Running a coupled simulation

Test case presentation: lid-driven cavity with a flexible bottom

Convenient benchmark

- Mesh simplicity
- Computational time:
  $T_s^{\text{CPU}} = 2.95 \times 10^{-3} s$
  $T_f^{\text{CPU}} = 1.08 \times 10^{-1} s$
- Harmonic solution

$|\boldsymbol{v}|$ and isopressure



Problem parameters

- Fluid problem $\rho_f = 1 kg.m^{-3}$, $\nu_f = 0.01 m \cdot s^{-2}$.
- Fluid boundary conditions:
  - only ∇p required
  - $\boldsymbol{v} \cdot \boldsymbol{e}_x = 1 - \cos(2\pi t / T_{\text{char}})$
- Structure problem:
  $\rho_s = 500 kg \cdot m^{-3}$, $E_s = 250 Pa$ *and* $\nu_s = 0$
- Discretization
  - Fluid: 32x32 cells for $\mathcal{R}e \leq 300$.
  - Structure: 16 quadratic elements.
  - Time step: $\Delta t = 0.1 s$.

Running a coupled simulation

Test case: file architecture

Activités Fichiers lun. 17:52 Christophe Kassiotis

cavityFSI

Fichier Édition Affichage Aller à Signets Aide

Périphériques
- USBDISK

Poste de travail
- Dossier personnel
- Documents
- Téléchargements
- Musique
- Images
- Vidéos
- Système de fichiers
- Corbeille

Réseau
- Explorer le réseau

Dossier personnel Software cops test run cavityFSI Recherche

Nom	Taille	Type	Date de modification
control	1 élément	dossier	lun. 02 janv. 2012 11:04:31 CET
cops.ctrl	1,2 ko	script/fonction MATLAB	lun. 02 janv. 2012 11:04:31 CET
fluid	3 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
0	3 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
constant	4 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
system	5 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
output	0 élément	dossier	lun. 02 janv. 2012 11:04:31 CET
solid	2 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
Cstruc	1,1 ko	script/fonction MATLAB	lun. 02 janv. 2012 11:04:31 CET
Istruc	733 octets	document texte brut	lun. 02 janv. 2012 11:04:31 CET
ctl.env	318 octets	document texte brut	lun. 02 janv. 2012 11:04:31 CET

Running a coupled simulation

Test case: file architecture

Activités Fichiers lun. 17:52 Christophe Kassiotis

cavityFSI

Fichier Édition Affichage Aller à Signets Aide

Périphériques
- USBDISK

Poste de travail
- Dossier personnel
- Documents
- Téléchargements
- Musique
- Images
- Vidéos
- Système de fichiers
- Corbeille

Réseau
- Explorer le réseau

Dossier personnel Software cops test run cavityFSI Recherche

Nom	Taille	Type	Date de modification
control	1 élément	dossier	lun. 02 janv. 2012 11:04:31 CET
cops.ctrl	1,2 ko	script/fonction MATLAB	lun. 02 janv. 2012 11:04:31 CET
fluid	3 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
0	3 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
constant	4 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
system	5 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
output	0 élément	dossier	lun. 02 janv. 2012 11:04:31 CET
solid	2 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
Cstruc	1,1 ko	script/fonction MATLAB	lun. 02 janv. 2012 11:04:31 CET
Istruc	733 octets	document texte brut	lun. 02 janv. 2012 11:04:31 CET
ctl.env	318 octets	document texte brut	lun. 02 janv. 2012 11:04:31 CET

Running a coupled simulation

Test case: `ctl.env`

- `ctl.env` controls which component is used
- components can be remote executable or libraries (local calls)
- use of template component is possible

```
copsCI -> /home/kassiotis/Software/cops/bin/cops.ctl.so -l lib
SimuCI<real8> -> /home/kassiotis/Software/cofeap/bin/[...]
                                          cofeap.ctl.so -l thread
CFDSimuCI -> /home/kassiotis/Software/components/bin/[...]
                                      pimpleDyMFoam.ctl.exe -l tcp
PointStringInterpolatorCI -> /home/kassiotis/Software/[...]
                     Interpolator/bin/Interpolator.ctl.so -l lib
```

Running a coupled simulation

Test case: file architecture

Activités Fichiers lun. 17:52 Christophe Kassiotis

cavityFSI

Fichier Édition Affichage Aller à Signets Aide

Périphériques
- USBDISK

Poste de travail
- Dossier personnel
- Documents
- Téléchargements
- Musique
- Images
- Vidéos
- Système de fichiers
- Corbeille

Réseau
- Explorer le réseau

Dossier personnel Software cops test run cavityFSI Recherche

Nom	Taille	Type	Date de modification
control	1 élément	dossier	lun. 02 janv. 2012 11:04:31 CET
cops.ctrl	1,2 ko	script/fonction MATLAB	lun. 02 janv. 2012 11:04:31 CET
fluid	3 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
0	3 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
constant	4 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
system	5 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
output	0 élément	dossier	lun. 02 janv. 2012 11:04:31 CET
solid	2 éléments	dossier	lun. 02 janv. 2012 11:04:31 CET
Cstruc	1,1 ko	script/fonction MATLAB	lun. 02 janv. 2012 11:04:31 CET
Istruc	733 octets	document texte brut	lun. 02 janv. 2012 11:04:31 CET
ctl.env	318 octets	document texte brut	lun. 02 janv. 2012 11:04:31 CET

Running a coupled simulation

Test case: control/cops.ctrl controls cops

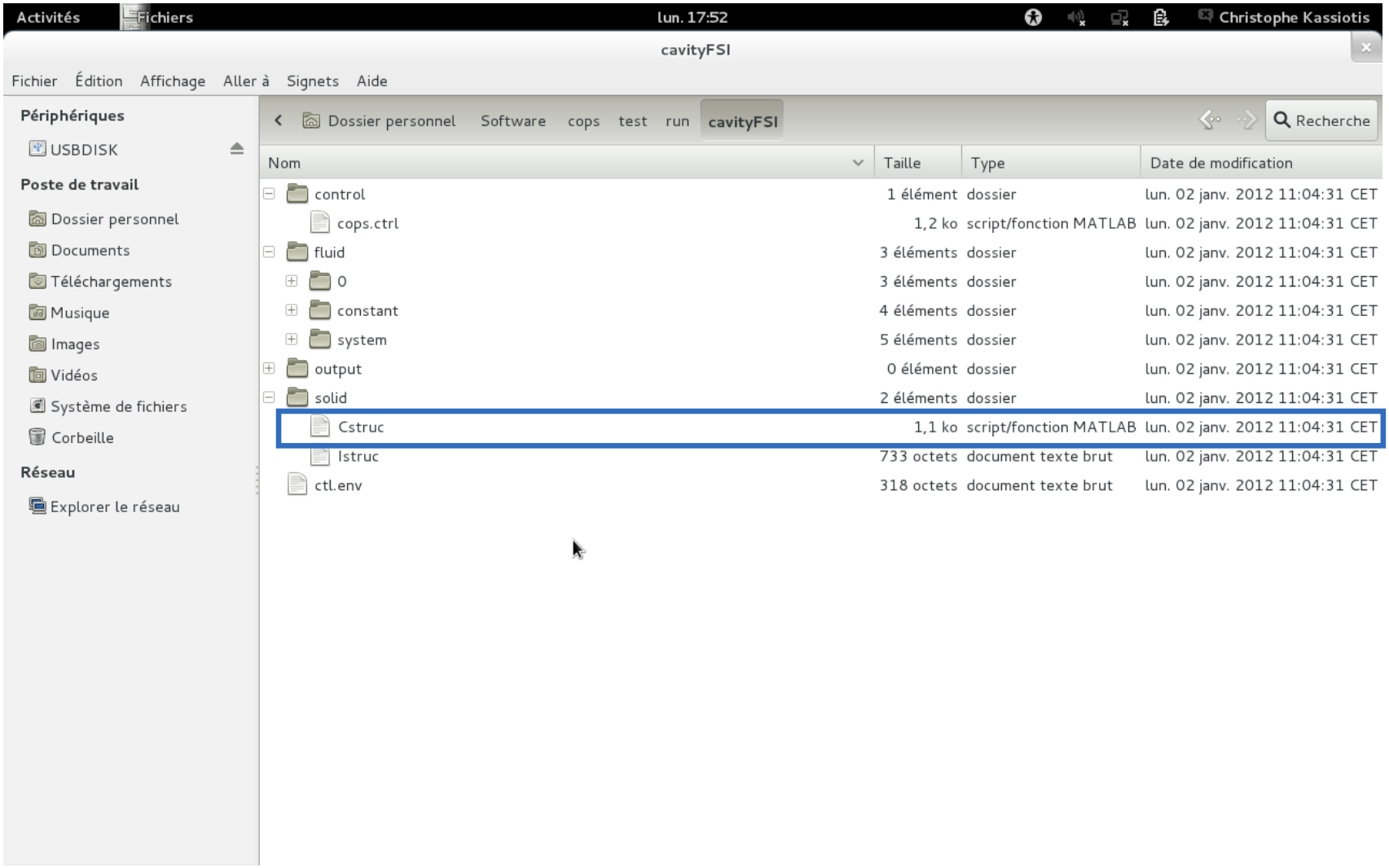
- **co**ps itself is a component
- cops.ctrl controls which coupling algorithm is used
- syntax based on boost_programm_options

```
verbose = 2

[time]
   begin = 0.0
   step = 0.1
   end  = 20.0
   writeInterval = 1

[solver]
   name = BGS
   collocated = true
   corrector = false
   iterationMax = 100
   tolerance = 1e-5
   predictor.order = 2
```

```
[solver.convergence]
   name = absolute
   relativeToInitial = false

[relaxation]
   name = Aitken
   value = 0.35

[interpolator]
   radius1 = 1e-7
   radius2 = 1e-7
   verbose = true
```

Running a coupled simulation

Test case: file architecture



Running a coupled simulation

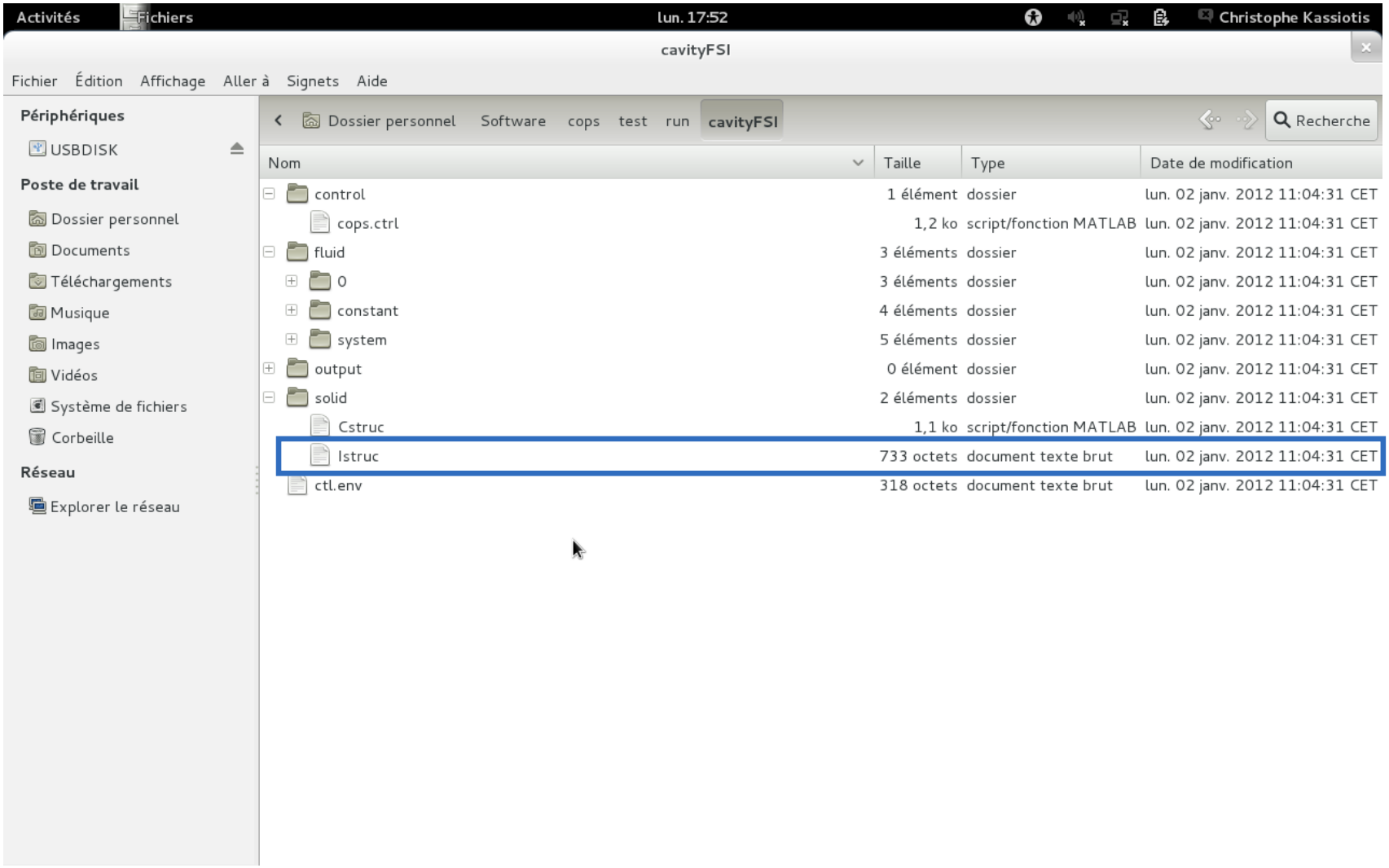
Test case: solid/Cstruc controls coFeap

- coFeap is made for coupled and multi-scale simulations
- Cstruc controls how coFeap is used
- cforc for the way to impose coupling forces
- maxLoop for the number of iterations...

```
inputFile = Istruc
symetricProblem = false
mute = true
maxLoop = 35
setLoad = cfor
```

Running a coupled simulation

Test case: file architecture



Running a coupled simulation

Test case: solid/Istruc

- Istruc is a normal Feap input file

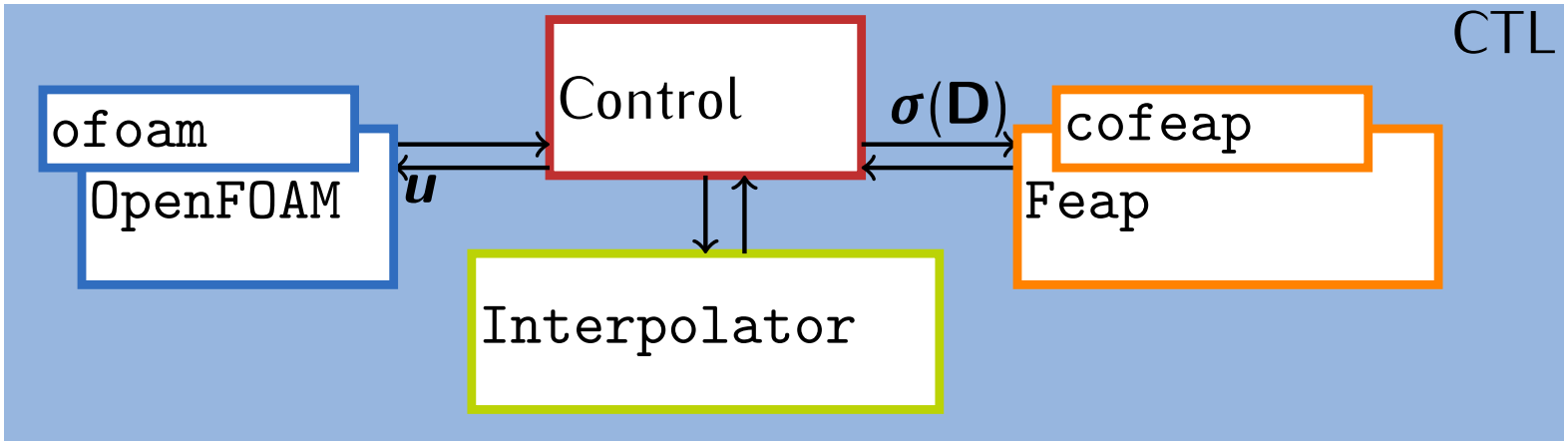
```
feap : structure for FSI
    ,,,2,2,8

BLOCK
  CARTESIAN 32 2 1 1 1 0 8
    1 0 -0.002
    2 1 -0.002
    3 1 0
    4 0 0

! Finite deformation/St-Vt Kirff mat.
MATE,1
  SOLID
    FINIte
    ELAS STVK 250.0 0.0
    DENS MASS 500
    MASS LUMP
```

```
!boundary
EBOU
  1 0.0 1 1
  1 1.0 1 1

end

!time integration scheme
batch
   DT,,0.1
   TRAN,ALPH,1,1.5,1
   TIME
end

stop
```

Plan

1. Running a coupled simulation
2. Coupling components
3. Building components

Coupling components



Coupling CTL components

- The component is defined by its interface (`ci`)
- The component can be called in Java, C++, Fortran
- You don't really need to know how it is implemented
- Examples: ofoam interface (`cfdsimu.ci`), coFeap interface (`simu.ci`),

Coupling components

coFeap's interface softsimu.ci

```
#ifndef __SIMU_CI_
#define __SIMU_CI_

#include <ctl.h>
#define CTL_ClassTmpl SimuCI, ( scalar1 ), 1
#include CTL_ClassBegin

#define CTL_Constructor1  (const string /*filename*/), 1
#define CTL_Constructor2  (const string /*filename*/, \
                           const array<scalar1> /*param*/), 2
[...]
#define CTL_Method6 void, set_load,
                          (const array<scalar1> /*load*/), 1
[...]
#define CTL_Method17 void, get_stiff,
                           (array<int4> /*row's index*/, \
                            array<int4>/*column's index*/, \
                            array<scalar1> /*matrix*/), 3
[...]
#include CTL_ClassEnd

#endif
```

Coupling components

calling the component: cops Steklov-Poincaré class

```
template<class simu, class vecReal>
class SteklovPoincare
{
    public:
        // Consructor
        SteklovPoincare(simu& s) : _s(s) {};

        // Compute the Steklov-Poincaré operator for time step dt
        void operator()( const real& dt, const vecReal& u, vecReal& l)
        {
            _s.set(u);
            _s.solve(dt);
            _s.get(l);
        };

    private:
        simu& _s;
};
```

Coupling components

calling the component: coFeap test client

```
#include <vector>
#include <iostream>
#include <simu.ci>

int main(int argc, char ** arg)
{
    // instanciation
    SimuCI<double>::use("+ -d ../test/run/therm_2D/");
    SimuCI<double> feap("Cqua4");

    // get stiffness matrix
    std::vector<double> stiff;
    std::vector<int> I, J;
    feap.get_stiff(I,J,stiff);

    // posttreatement
    [...]

    return 0;
}
```

Coupling components
compiling the client

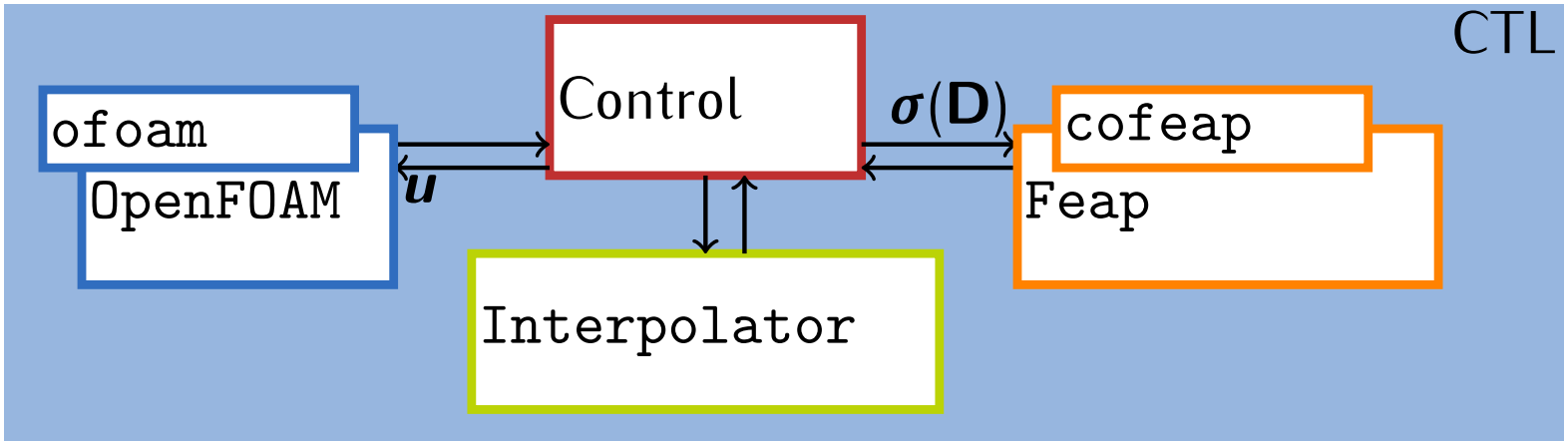
- the client depends only on the `.ci`
- the client is linked with `ctl`
- you don't need to link with any coFeap library

```
g++ -fPIC -O3 -I../cofeap/include/ci getStiffTest.cpp
                                        -o getStiffTEST -lctl
```

Plan

1. Running a coupled simulation
2. Coupling components
3. Building components

Building components



Building CTL components

- The component is defined by its interface (`ci`)
- The interface is implemented in pre-processing C
- The method are implemented in the language of the existing software
- You need to know what the software really does

Building components

Defining the interface: get_stiff in simu.ci

```
#ifndef __SIMU_CI_
#define __SIMU_CI_

#include <ctl.h>
#define CTL_ClassTmpl SimuCI, ( scalar1 ), 1
#include CTL_ClassBegin

#define CTL_Constructor1  (const string /*filename*/), 1
#define CTL_Constructor2  (const string /*filename*/, \
                           const array<scalar1> /*param*/), 2
[...]
#define CTL_Method6 void, set_load,
                          (const array<scalar1> /*load*/), 1
[...]
#define CTL_Method17 void, get_stiff,
                           (array<int4> /*row's index*/, \
                            array<int4>/*column's index*/, \
                            array<scalar1> /*matrix*/), 3
[...]
#include CTL_ClassEnd

#endif
```

Building components

Implementing a method: get_stiff in coFeap

```
      subroutine simu_get_stiff_impl(row_size,row,col_size,
&     col,Stiff_size,Stiff)

      implicit none

      include   'pointer.h'
      [...]
      include   'fdata.h'   !.............fl(9)

      integer       Stiff_size, row_size, col_size,JD
      integer*4     row(row_size), col(col_size)
      integer       ii, jj, nn, kk
      integer       Matrix_size
      real*8        Stiff(Stiff_size), tol
      logical       tr, fa, cfr, exst
      data          tr /.true./ , fa / .false. /

      jd = np(20+npart)
      Matrix_size = neq + 2*mr(jd+neq-1)
```

Building components

Implementing a method: get_stiff in coFeap

```
      if((row_size.lt.Matrix_size).or.(col_size.lt.Matrix_size)   43
     &           .or.(Stiff_size.lt.Matrix_size)) then            44
          row_size = Matrix_size                                  45
          col_size = Matrix_size                                  46
          Stiff_size = Matrix_size                                47
! output tangent matrix                                          48
        else                                                      49
          tol = 1e-10 ;                                           50
! flag for matrix symetric                                       51
          cfr = use_tang.eq.1                                     52
! set the values na, nau, nal                                    53
          call presol(cfr, exst)                                  54
! set the dynamic parameters                                     55
          if(fl(9)) call dsetci                                   56
          do ii = 1,3                                             57
             ctan(ii) = gtan(ii)                                  58
          enddo                                                   59
! build the [U]TANG matrix                                       60
          call formfe(np(40),np(26),na,nal,tr,fa,cfr,fa,3,1,numel,1)  61
! get the [U]TANG matrix with an IJV format                      62
```

Building components

Implementing a method: get_stiff in coFeap

```
! get the [U]TANG matrix with an IJV format                    63
          nn = 1                                               64
          row(nn) = 0                                          65
          col(nn) = 0                                          66
          Stiff(nn) = hr(na)                                   67
          do ii = 1,neq-1                                      68
             jj = ii - mr(JD+ii) + mr(JD+ii-1)                 69
             do kk = mr(JD+ii-1), mr(JD+ii)-1                  70
! get the L part                                               71
                if(abs(hr(nal+kk)).ge.tol) then                72
                   nn = nn+1                                   73
                   row(nn) = jj                                74
                   col(nn) = ii                                75
                   Stiff(nn) = hr(nal+kk)                      76
                endif                                          77
! get the U part                                               78
                if(abs(hr(nau+kk)).ge.tol) then                79
                   nn = nn+1                                   80
                   row(nn) = ii                                81
                   col(nn) = jj                                82
                   Stiff(nn) = hr(nal+kk)                      83
                endif                                          84
```

Building components
Connecting interface and methods

- CTL_ConnectF is used to connect methods in Fortran with the interface
- methods avec to be implemented in a file named simu_XXX_impl.f
- the template component is specified in the connect file

```
#define CTL_ConnectF

#define CTL_ClassPrefix simu
typedef double scalar1;

#include <simu.ci>

void CTL_connect()
{ ctl::connectF<SimuCI<double>, ctl::Extern::SimuCI>(); }
```

Building components

Defining the interface: get in cfdsimu.ci

```
#ifndef __CFDSIMU_CI_
#define __CFDSIMU_CI_

#include <ctl.h>

#define CTL_Class CFDSimuCI
#include CTL_ClassBegin

#define CTL_Constructor1
  ( const string /*case control file*/ ), 1
#define  CTL_Method1 void, set,
  ( const string /*fieldName*/, const array<real8> /*field*/ ), 2
#define  CTL_Method2 void, get,
  ( const string /*fieldName*/, array<real8> /*field*/ ), 2
#define  CTL_Method3 int4, solve,
  ( const real8 /*timeStep*/ ), 1
#define  CTL_Method4 int4, goback, (), 0
#define  CTL_Method5 void, getnodes,
  ( int4 /*dimension*/, array<real8> /*nodes*/ ) const, 2

#include CTL_ClassEnd
#endif
```

Building components

Implementing a method: get in ofoam

```
#include "addToRunTimeSelectionTable.H"
#include "pointForces.H"
void ofoam::pimpleDyMFoam::get( const std::string& fieldName,
    std::vector<double>& field, const std::string& patchName ) {

    // pointers required
    const dynamicFvMesh& mesh = mesh_();
    const Time& runTime = runTime_();

    // redirect output
    std::ofstream Out( outputName_.c_str(), std::fstream::app );
    std::streambuf* OldBuf = std::cout.rdbuf(Out.rdbuf());

    // redirect output
    if( !parallel_ )
        Info << "<ofoam::X::get> get " << fieldName << " on patch " <<
             << " ( tEnd:" << runTime.endTime().value()
             << ", t:" << runTime.value() << ")" << endl;

    label patchID = mesh.boundaryMesh().findPatchID( patchName );
    // throw error if patch does not exist
    if( patchID == -1 ){
```

Building components
Linking interface and methods

- Connect between C++ classes and components

```
#define CTL_Connect

#include <ci/cfdsimu.ci>
#include <pimpleDyMFoam.H>

struct connectDetailsCI {
    CTL_Constructor( 1, ( const std::string& ), 1 );
    CTL_Method( 1, void, ofoam::pimpleDyMFoam::set,
      ( const std::string&, const std::vector<double>& ), 2);
    CTL_Method( 2, void, ofoam::pimpleDyMFoam::get,
      ( const std::string&, std::vector<double>& ), 2);
    CTL_Method( 3, int, ofoam::pimpleDyMFoam::solveLoop,
      ( const double& ), 1);
[...]
};

void CTL_connect() {
  ctl::connect<CFDSimuCI, ofoam::pimpleDyMFoam, connectDetailsCI >();
}
```

Conclusion

- Components are build from existing software
- We use the CTL that allows good performance
- The interface is programmed in C pre-processor
- The methods are implemented in the existing software language (Fortran, C++)
- Once the component build, one does not need to know what is behind the interface
- The component is called with a client (generally programmed in C++, but it is not mandatory)
- The component can be called as a library, a thread or a remote executable
- The component itself can run in parallel (transparent for the client)