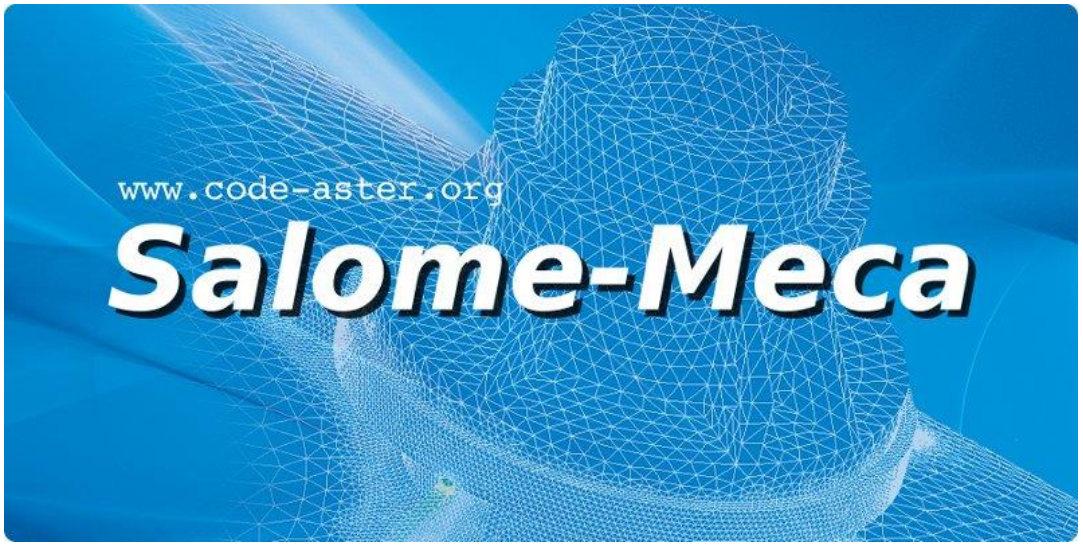
Development in code_aster
ASTERXX – The new architecture of code_aster



Code_Aster, Salome-Meca course material

EDF

Table of contents

Data structures (1/4)

In code_aster “legacy”, data structures are:

A set of jeveux objects (vector, collection, bidirectional map), example: a mesh (sd_maillage) is composed of:

A jeveux vector containing coordinates of nodes

A jeveux vector containing names of nodes

And so on

Link together by names, example: for a mesh named `MAIL` in commands file:

The jeveux vector containing coordinates of nodes is named: ‘`MAIL      .COORDO`’

The jeveux vector containing coordinates of nodes is named: ‘`MAIL      .NOMNOE`’

…

Drawback

No link between objects (in sense of programming language)

No automatic garbage collector

Data structures (2/4)

Wrap C++ of aster data structures

Definition of classes which represents aster data structures
All theses aster data structures inherit of a class named `DataStructure`

Class `DataStructure`

This class contains (among others) the name of sd_aster
Also contains: memory allocation and object '`.TCO`' (aster type of a data structure, ex : sd_maillage)
Warning: changes in sd_aster naming !!!
The name is no longer based on the name give by user in the commands file
Before in Aster legacy: '`MAIL      .COORDO`' (for instance : `MAIL=LIRE_MAILLAGE()`)
Now, first 8 characters are automatically generated: '`00000001.COORDO`'

Data structures (3/4)

Wrap C++ of Jeveux objects

`JeveuxVector`, `JeveuxCollection` and `JeveuxBidirectionalMap`

Used in C++ data structures

Example, `class ModelClass`

```
class ModelClass: public DataStructure
{
protected:
    // (…)
    /** @brief Vecteur Jeveux '.MAILLE' */
    JeveuxVectorLong     _typeOfCells;
    // (…)
public:
    ModelClass( const std::string name = ResultNaming::getNewResultName() ):
        DataStructure( name, 8, "MODELE" ),
        _typeOfCells( JeveuxVectorLong( getName() + ".MAILLE    " ) ),
    // (…)
// shared pointer on this object
typedef boost::shared_ptr< ModelClass > ModelPtr;
    {};
};
```

Data structures (4/4)

Note

For more flexibility, we choose to use pointers…

… But pointers in C++ doesn't used garbage collector

So, in asterxx, we have widely used smart-pointers (i.e. : pointers with garbage collector)

All DataStructure are therefore wrap in a `boost::shared_ptr`

Transparent and easy-to-use

Protect use of "`new`" in C++, no need to use "`delete`"

Never use raw pointer

Data structures in python (1/2)

To make data structures accessible in python, we use the library Boost-python

Boost-python is:

A way to define classes and functions exposed

Explain to code that you want that a C++ class name `TotoClass` (for class) must be exposed to user in a python shell with the name `Toto` (for instance too !)

Natively compatible with python garbage collector

No more need to use `DETRUIRE` in commands file

Reasonably easy to use (at least by imitation), example : Model

```
class_< ModelClass, ModelClass::ModelPtr,
        bases< DataStructure > > ( "Model", no_init )
       .def( "__init__", make_constructor(
             &initFactoryPtr< ModelClass >) )
       .def( "addModelingOnMesh", &ModelClass::addModelingOnMesh )
```

Data structures in python (2/2)

After writing Boost-python and compile : class ready to use in python

```
# import module code_aster
import code_aster
# start code_aster (jeveux for real)
code_aster.init()

# have fun with code_aster :
# define a empty model !
myOwnModel = code_aster.Model()
# unbelievable : you have defined a model that is stored in python variable
# with more than 8 characters !

 myOwnModel = 3.0
# even more incredible : you have deleted a sd_aster without using DETRUIRE !
```

Sources organization (1/2)

New folders

src/bibcxx folder contains:

All data structures

New C++ code (example: `MECA_STATIQUE` C++ style)

src/code_aster folder contains essentially:

Subfolder Cata which contains capy files (syntax definition of commands file) and the new syntax checker

Subfolder Commands which contains interfaces which call old Fortran operators from a aster command

Practically: 2 items, name of the commands et type of the results of the commands

```
from ..Objects import Model
from .ExecuteCommand import ExecuteCommand


class ModelAssignment(ExecuteCommand):
    command_name = "AFFE_MODELE"

    def create_result(self, keywords):
        self._result = Model()

    def post_exec(self, keywords):
        self._result.setMesh(keywords["MAILLAGE"])

AFFE_MODELE = ModelAssignment.run
```

Sources organization (2/2)

bibcxx

Contains lot of subfolders (Algorithms, DataFields, …)

Important subfolder: PythonBindings

Define python interface to C++ objects

Usually headers define functions, classes and member functions and .cxx files implement functions and member functions

Link between data structures and Fortran

Jeveux objects are still build in Fortran (in most cases)

2 possibilities:

Use of aster commands => call of Fortran operators from python

Use of build member function => call of Fortan operators from C++

How is build the link between C++ and jeveux objects

With the name!

OP* allocate a set of jeveux objects which are pointed by wrappers in C++ class

Then garbage collector is operational

Automatic call of `JEDETR`

Asterxx in a (big) nutshell (1/4)

From a commands file

```
import code_aster
from code_aster.Commands import *

code_aster.init()

mail1 = LIRE_MAILLAGE( FORMAT = "MED" )

model = AFFE_MODELE( MAILLAGE = mail1,
                     AFFE = _F( MODELISATION = "3D« ,
                     PHENOMENE = "MECANIQUE",
                     TOUT = "OUI", ), )
```

How to put together all what I said?

First, have a look to src/code_aster/Commands/affe_modele.py

Asterxx in a (big) nutshell (2/4)

First, have a look to src/code_aster/Commands/affe_modele.py

```
# (…)
class ModelAssignment(ExecuteCommand):
    # What is the name of the defined command ? => Creation of link between command and “capy”
    command_name = "AFFE_MODELE"

    # What is the aster type of the result produced by the command ?
    def create_result(self, keywords):
        self._result = Model()

    # What must be done at the end of the command (after calling OP*)
    def post_exec(self, keywords):
        self._result.setMesh(keywords["MAILLAGE"])

# What is the name that users will use to call the command ?
AFFE_MODELE = ModelAssignment.run

# When user will type : AFFE_MODELE(), it is ModelAssignment.run which will be call
# And then ModelAssignment.run will call OP0018 after reading capy (for AFFE_MODELE, op=18 in capy)
```

Asterxx in a (big) nutshell (3/4)

Then, what is `Model()` ?

Go to src/bibcxx/PythonBindings/ModelInterface.cxx

```
// (…)
#include "PythonBindings/ModelInterface.h"
// (…)
 class_< ModelClass, ModelClass::ModelPtr,
            bases< DataStructure > > ( "Model", no_init )
// (…)
         .def( "addModelingOnMesh", &ModelInstance::addModelingOnMesh )
         .def( "addModelingOnGroupOfCells", &ModelInstance::addModelingOnGroupOfCells )
         .def( "addModelingOnGroupOfNodes", &ModelInstance::addModelingOnGroupOfNodes )
         .def( "build", &ModelClass::build )
//(…)
```

In summary, `Model` (in python) is link (the same as) to `ModelClass` in C++ describes in src/bibcxx/Modeling/Model.h

Asterxx in a (big) nutshell (4/4)

Then, what is `ModelInstance()`?

Go to src/bibcxx/Modeling/Model.h

```
// (…)
class ModelClass: public DataStructure
{
    protected:
// (…)
        /** @brief Vecteur Jeveux '.MAILLE' */
        JeveuxVectorLong     _typeOfCells;
        /** @brief Vecteur Jeveux '.NOEUD' */
        JeveuxVectorLong     _typeOfNodes;
        /** @brief Vecteur Jeveux '.PARTIT' */
        JeveuxVectorChar8    _partition;
// (…)
```

`ModelClass` is a C++ class containing jeveux objects (such as `'.MAILLE'`, `'.NOEUD'`, …)

So, it's just a sd_modele with all its jeveux objects!

Asterxx and development (1/2)

Develop in Fortran

Old-style

Almost nothing changes (Fortran, capy, catalogues, …)

What changes?

Write a python file in src/code_aster/Commands/*.py to link name of a command to a capy

Write (maybe, only if you need) a new data structure

A C++ class in src/bibcxx/… to describe what is inside the data structure

A Boost-python file in src/bibcxx/PythonBindings/… to make data structure available in python

Develop in python

With the new “object oriented” style

Looks like macro python

For most developers

Ask the core team for new features (data field handling, data structure allocation, …)

Read the documentation

Asterxx and development (2/2)

Develop in C++

Essentially for core team

For specific needs (for instance : performance issue)

End of presentation

Is something missing or unclear in this document?

Or feeling happy to have read such a clear tutorial?

Please, we welcome any feedbacks about code_aster training materials.

Do not hesitate to share with us your comments on the code_aster forum dedicated thread.