Runtime Selection Mechanism in OpenFOAM

Bernardas Jankauskas

University of Exeter; Hydro International; STREAM IDC

bj255@exeter.ac.uk

February 15, 2017

Seminar outline

1. RTS principle ideas
   1. Need + convenience
   2. Macros inception
2. RTS in non-templated classes
3. RTS in templated classes

Need + convenience

- Modelling the world is imprecise, so we have millions of different models created;
- OF is not user friendly, so RTS tries to give some convenience;
- Everybody hates recompiling the code when a specific model is hard coded into the solver;
- RTS let's us fix/change the setup when the simulation is already running;

RTS process

Conceptually RTS mechanism performs these following operations:

- In Base Class declares a HashTable conveniently named RunTimeSelectionTable;
- In Base Class has an *autoPtr* outputing function called selector;
- In Base/Derived class registers the name for the RTSelectable class;
- In Derived class add the class to the RTS table;
- In Base/Derived class has a static variable *TypeName* declaring the name for RTS;

Most of that is sorted with the help of various macros.

Macros for shorthand operations

Not CS correct definition: 'Macros are shorthand notations of the code that the preprocessor expands':

```
#define TABLE_SIZE 100
   int table1[TABLE_SIZE];
   int table2[TABLE_SIZE];
```

which preprocessor expands into:

```
int table1[100];
int table2[100];
```

Macro could also behave like a function that takes in arguments:

```
#define getmax(a,b) a>b?a:b
```

Macros for shorthand operations

In RTS there are a few macros used:

```
1) TypeName("TypeName");
2) declareRunTimeSelectionTable
            (autoPtr, baseType, argNames, argList, parList);
3) defineTypeNameAndDebug(AlgorithmBase, 0);
4) defineRunTimeSelectionTable(AlgorithmBase, Arg);
5) addToRunTimeSelectionTable(Base, Derived, Arg)
```

Templated classes replace (3) with:

```
3) defineTemplateTypeNameAndDebugWithName
            (TypeName, TemplateNameString, 0)
```

Macros for shorthand operations

```
1) TypeName("TypeName");
```

Declare class typename that will be used in RTS process. Needed for every RT selectable class and it's parent class.

Macros for shorthand operations

```
2) declareRunTimeSelectionTable
                (autoPtr, baseType, argNames, argList, parList);
```

In base class (or a class that will be used for checking RTS possible types (see combustionModels)) it declares stuffs needed by RTS.

Note - *argNames* can be either *Word* or *dictionary*

Macros for shorthand operations

```
3) defineTypeNameAndDebug(AlgorithmBase, 0);
```

typedef and debug switch macro initialisation macro.

Macros for shorthand operations

```
4) defineRunTimeSelectionTable(AlgorithmBase, Arg);
```

Defines RTS table in a base class.

Macros for shorthand operations

```
5) addToRunTimeSelectionTable(Base, Derived, Arg)
```

Add a RT selectable class to a hash table (usually it would be a derived class, but a base class can be made RT selectable as well).

Macros for shorthand operations

```
3) defineTemplateTypeNameAndDebugWithName
        (TypeName, TemplateNameString, 0)
```

For templated classes. Again does what it says on the tin - sets up all typename and debug stuffs.

Selectors

- TypeNames - check
- declarations and defines - check
- added models to a table - check

One last thing. Computer does not know how to compare the model typename selected in the dictionary with ones added to the table and, if matched, create it. *Selector* method to the rescue!

Selectors

Example where model typename is given as an input.

```
// static Factory Method (selector)
static autoPtr New (const word& algorithmName)
{
    // Find the Factory Method pointer in the RTS Table
    // (HashTable<word, autoPtr(*)(word))
        WordConstructorTable::iterator cstrIter =
        WordConstructorTablePtr_->find(algorithmName);
```

Selectors

```
    // If the Factory Method was not found.
    if (cstrIter == WordConstructorTablePtr_->end())
    {
        FatalErrorIn
        (
        "AlgorithmBase::New(const word&)"
        ) << "Unknown AlgorithmBase type "
        << algorithmName << nl << nl
        << "Valid AlgorithmBase types are :" << endl
        << WordConstructorTablePtr_->sortedToc()
        << exit(FatalError);
    }
    // Call the "constructor" and return the autoPtr
    return cstrIter()(algorithmName);
}
```

Selectors

Example where dictionary is given as an input - we need to tell it where to look for the name.

```
static autoPtr<Foam::liquidReaction> New (cons dictionary& dict)
{
    const word modelName = dict.lookup("reationType");

    dictionaryConstructorTable::iterator cstrIter =
        dictionaryConstructorTablePtr_->find(modelName);
```

Selectors

```
    // If the Factory Method was not found.
    if (cstrIter == dictionaryConstructorTablePtr_->end())
    {
    FatalErrorIn
    (
        "liquidReaction::New(const word&)"
    )   << "Unknown liquidReaction type "
        << modelName << nl << nl
        << "Valid liquidReaction types are :" << endl
        << dictionaryConstructorTablePtr_->sortedToc()
        << exit(FatalError);
    }
    // Call the "constructor" and return the autoPtr
    return autoPtr<liquidReaction>(cstrIter()(dict));
}
```

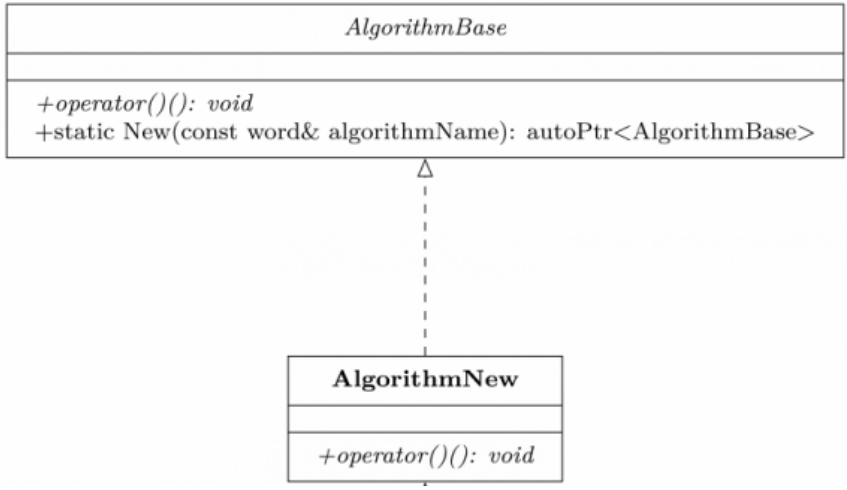
Seminar outline

1. RTS principle ideas

2. RTS in non-templated classes
   1. base/parent + derived/children class structure
   2. Example

3. RTS in templated classes

base/parent + derived/children class structure



Example taken from Sourceflux

Includes:

```
#include "word.H"
#include "messageStream.H"
#include "argList.H"
#include "autoPtr.H"                        // in base, though not sure
                                            // if necessary

using namespace Foam;

#include "typeInfo.H"                       // in both base/derived
#include "runTimeSelectionTables.H"         // in base
#include "addToRunTimeSelectionTable.H" // in derived
```

Base class

```
class AlgorithmBase
{
    public:
    // Declare the static variable typeName of the class AlgorithmBase.
    TypeName ("base");
    // Empty constructor.
    AlgorithmBase () {};

    // Word constructor.
    AlgorithmBase (const word& algorithmName) {};
    // Destructor: needs to be declared
    // virtual because of the OpenFOAM type name.
    virtual ~AlgorithmBase() {};
```

Base class RTS declaration

```
// Macro for declaring stuff needed by RTS. Usually in .H
declareRunTimeSelectionTable
(
    autoPtr,
    AlgorithmBase,
    Word,
    (
        const word& algorithmName
    ),
    (algorithmName)
)
```

Base class selector

```
// Static Factory Method (the New selector).
static autoPtr New (const word& algorithmName)
{
    // Find the Factory Method pointer in the RTS Table
    // (HashTable<word, autoPtr(*)(word))
        WordConstructorTable::iterator cstrIter =
            WordConstructorTablePtr_->find(algorithmName);
    // If the Factory Method was not found.
```

Base class selector

```
if (cstrIter == WordConstructorTablePtr_->end())
{
    FatalErrorIn
    (
    "AlgorithmBase::New(const word&)"
    ) << "Unknown AlgorithmBase type " << algorithmName
    << nl << nl << "Valid AlgorithmBase types are :" << endl
    << WordConstructorTablePtr_->sortedToc()
    << exit(FatalError);
}
// Call the "constructor" and return the autoPtr
return cstrIter()(algorithmName);
}
```

Base class functions + end

```
// Make the class callable (function object)
    virtual void operator()()
    {
        // Overridable default implementation
        Info << "AlgorithmBase::operator()()" << endl;
    }
}; // End of Class definition

// Usually in .C
defineTypeNameAndDebug(AlgorithmBase, 0);
defineRunTimeSelectionTable(AlgorithmBase, Word);
addToRunTimeSelectionTable(AlgorithmBase, AlgorithmBase, Word);
```

Derived class

```
class AlgorithmNew
:
    public AlgorithmBase
{
    public:
    // Declare the static variable typeName of the class AlgorithmNew.
    TypeName ("new");
    // Empty constructor.
    AlgorithmNew () {};
    // Word constructor.
    AlgorithmNew (const word& algorithmName) {};
```

Derived class

```
// Make the class callable (function object)
virtual void operator()()
{
    Info << "AlgorithmNew::operator()()" << endl;
}
}; // End of Class description

defineTypeNameAndDebug(AlgorithmNew, 0);
addToRunTimeSelectionTable(AlgorithmBase, AlgorithmNew , Word);
```

Seminar outline

1. RTS principle ideas
2. RTS in non-templated classes
3. RTS in templated classes
   1. Names and *typedef* macros
   2. makeType Macro

'Issues' with templated structures

For complicated class structures multiple classes are intertwined and are prerequisite to each other, this is a requirement as usually separate parts of the model are split to different classes. In order to register all possible model combinations to our RTS table we then have to use absurdly long names:

```
constTransport<species::thermo<hConstThermo
                <perfectGas<specie> >, sensibleEnthalpy> >
```

To avoid typing all of that we use *typedef* to give an alternative name for this line and then use macros to treat different combinations of all of the classes.

typedef macros

Use typedef to give an alternative name, e.g. volFields:

```
typedef GeometricField<scalar, fvPatchField, volMesh>
                                                volScalarField;
typedef GeometricField<vector, fvPatchField, volMesh>
                                                volVectorField;
```

Example from macro file:

```
typedef Foam::precipitationSystems::PrecModel
            <Foam::precipitationSystems::PrecType>
            PrecModel##PrecType;
```

makeType macros example

Include + Start of the macro with typedef based on macro input variables

```
#ifndef makePrecipitationTypes_H
#define makePrecipitationTypes_H

#include "addToRunTimeSelectionTable.H"

#define makePrecipitationTypes(PrecModel, PrecType, Table)  \
                                                            \
typedef Foam::precipitationSystems::PrecModel               \
        <Foam::precipitationSystems::PrecType>              \
        PrecModel##PrecType;                                \
                                                            \
```

makeType macros example

Define type name and debug switch:

```
defineTemplateTypeNameAndDebugWithName                  \
(                                                       \
    PrecModel##PrecType,                                \
    #PrecModel"<"#PrecType">",                          \
    0                                                   \
);
```

makeType macros example

Add the model to a RTS table:

```
namespace Foam                                                          \
{                                                                       \
    namespace precipitationSystems                                      \
    {                                                                   \
        typedef PrecModel<PrecType> PrecModel##PrecType;                \
                                                                        \
        addToRunTimeSelectionTable                                      \
        (                                                               \
            Table,                                                      \
            PrecModel##PrecType,                                        \
            dictionary                                                  \
        );                                                              \
    }                                                                   \
}
```

Calling the makeType macros

Usually done in ..s.C file. E.g. laminars.C

```
#include "makePrecipitationTypes.H"
#include "liquidChemistryPrecipitation.H"
#include "liquidBaboon.H"
#include "laminar.H"

makePrecipitationTypes(laminar, liquidChemistryPrecipitation,
                                liquidPrecipitationSystem);

makePrecipitationTypes(laminar, liquidBaboon,
                                liquidPrecipitationSystem);
```