Interface Guide

for FEDDLib and AceFem

1 Interface Overview

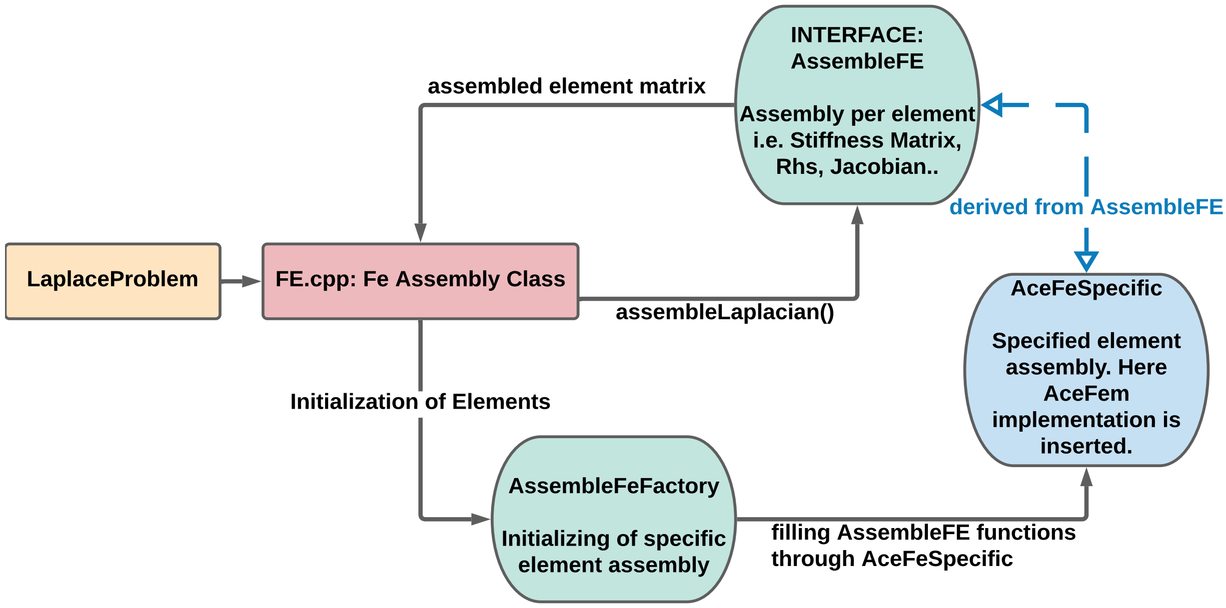

Figure 1.1: Accessing AceFem Implementation. Example for assembling finite element stiffness matrix for Laplace problem.

For implementing the interface we need three new classes:

- `AssembleFE`
- `AssembleFEFactory`
- `AceFeSpecific`

1.1 `AssembleFE` Class

`AssembleFE` is the basis class and interface for the finite element assembly.
We access the `AssembleFE` Implementation in the general finite element assembling class `FE`.
→ For each element we should be able to assemble different finite element entities, for example stiffness matrix, right hand side or Jacobian matrix, with the `AssembleFE` object.

```
// Assembly of element matrix for laplacian operator
AssembleFE.assemblyLaplacian();
```

Code 1.1: Calling `assemblyLaplacian()` of `AssembleFE` in `FE.cpp`

The `AssembleFE` class only owns virtual assembling functions.

```
virtual void assemblyLaplacian() = 0;
virtual void assemblyRHS() = 0;
```

Code 1.2: An example of virtual assembly functions of `AssembleFE`.

Those virtual functions are initialized through `AssembleFEFactory` with the corresponding `AceFeSpecific` assembly functions.
As an `AssembleFE` object represents one element, the implementation within `AssembleFE` is independent of any parallel implementation. The `FE` class distributes the local element matrices to the global matrices accordingly.

1.2 `AssembleFEFactory` Class

The `AssembleFEFactory` class fills the virtual functions of `AssembleFE` with the corresponding functions of the specific problem. It builds the `AceFeSpecific` object of the `AssembleFE` basis object.

```
AssembleFEFactory(string problemType, AssembleFE_Type assembleFE ):
{
if(problemType == "Laplace"){
       assembleFE = new AceFeSpecificLaplace();
}
else if( ..)

}
```

Code 1.3: Constructor of `AssembleFEFactory` that initializes the `assembleFE` object.

1.3 `AceFeSpecific` Class

The `AceFeSpecific` class is derived from `AssembleFE` and extends the virtual functions with specific assembly rules. Each specific problem corresponds to a `AceFeSpecific` class of functions.
For a Laplace problem we would have a `AceFeSpecificLaplace` class with its corresponding assembly of right hand side and stiffness matrix (see Code 1.4).

For adding a new element or assembly routine (or **AssembleFem** code), one must simply add a new or extend an `AceFeSpecific` class that fulfills `AssembleFE`'s assembly requirements and add the build information to `AssembleFEFactory`. This way the element assembly can be accessed equally in the `FE` class for each specific assembly routine that is concealed in the `AceFeSpecific` classes.

```
void assemblyLaplacian(Matrix &ElementMatrix){

    numNodes= nodesRefConfig_.size(); // number of nodes per element
    dPhi = this->getDPhi(dim_, FEType_); // nabla phi
    Matrix B(dim_); // transformation matrix
    Matrix Binv(dim_); // inverse of transformation matrix
    this->buildTransformation(B); // building inverse
    detB = B.computeInverse(Binv); // determinent of inverse
    dPhiTrans = this->applyBTinv( dPhi,Binv ); // applying inverse to dPhi

    // Filling element matrix with correct values
    for (int i=0; i < numNodes; i++) {
        for (int j=0; j < numNodes; j++) {
            ElementMatrix[i][j] = //assembly rules;
        }
    }
}
```

Code 1.4: `AssemblyLaplacian()` in `AceFeSpecificLaplace`.