8th. World Congress on Computational Mechanics (WCCM8)
5th. European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2008)
June 30 – July 5, 2008
Venice, Italy

A NON INTRUSIVE IMPLEMENTATION OF THE EXTENDED FINITE ELEMENT METHOD IN THE Z-SET FINITE ELEMENT TOOLKIT

***F. Feyel[1] and R. Foerch[2]**

[1] ONERA MSMD
29, Av. De la Division
Leclerc, F92322 Châtillon
feyel@onera.fr

[2] NW Numerics Inc, 150
Nickerson Street Suite 102,
WA 98109, Seattle
foerch@nwnumerics.com

Key Words: *XFEM, Implementation, Object-Oriented, C++*

ABSTRACT

Z-set is a finite element software devoted to non-linear mechanical analysis for materials and structures. It has been developed for about 25 years, and was rewritten during the 90's and is now entirely developed under an object-oriented paradigm (and C ++ as the support language). Today it consists in about 700 000 lines of code. The goal of this software is (i) to be robust enough to handle complex real industrial computations, but at the same time (ii) to be widely open to serve as a rich finite element toolkit able to welcome various modern research developments. It is thus used both by people from the industry and by researchers.

Some of the XFEM features was introduced two years ago in Z-set to describe crack propagation problems. The goal of this talk is to show some object oriented technics used in Z-set that greatly simplifies the implementation of XFEM in Z-set.

Z-set is based massively on polymorphism, inheritance and on client/server relationships. Templates are more or less only used for containers. A few design patterns are really the foundations of Z-set and allow a great flexibility within the software. These are mainly (i) the object factory pattern and (ii) the wrapper pattern.

- the object factory pattern is a way to provide an automatic association between any derived class and a keyword. Given the base class name and the keyword, the object factory returns, if possible, a pointer to the base class, pointing to an instance of the derived class associated with the keyword. The standard object factory pattern has been extended to make it more automatic: using macros and static objects, it allows the automatic fill-in of the underlying tables which maintain the association between derived classes and keywords. Z-set is entirely built on this strategy that provides a very convenient way, using plugins, to enrich the toolkit and also to modify its standard behavior by providing new associations in place of standard ones.

– the wrapper pattern is a technic to intercept all method calls to an object by using a specific derived class. After the substitution of the object by its wrapper, the calling object still proceeds as usual. All calls are then directed to the wrapper, that usually redirects them to the real object after some pre-treatments. Some post-treatments may also occur before returning in the calling object.

We assume in this work that the finite element problem is solved using a standard incremental approach, where the unknown are the displacements increment during the time increment. In Z-set a specific top-level object is responsible for coordinating the work of all sub-objects required during the computation: mechanical Newton-Raphson algorithm, finite element mesh, boundary conditions, etc. Many of this sub-objects derive from a virtual base class named a « problem component ». This base class does not contain anything useful; it only declares various methods to be called at predefined strategic place of the solution procedures: before initialization, after initialization, before increment, after increment, before iteration, etc.

The top level object owns a dynamic list of problem components, built during the input stage (ie during the input data file parsing). When needed, the predefined method is called for all problem components.

XFEM is theoretically a non intrusive method. However since it is based on a modification of the field kinematic, its implementation may induce deep modification in various solver layers. The main goal of our implementation of the XFEM was to avoid, as much as possible, a too deep intrusion.

In our vision, the various finite element elementary bricks (integration, shape functions, material behavior, etc...) should be kept clearly separated: XFEM should not result in a specific “XFEM-3 nodes triangle-elastic-finite-element”. Rather than this, we tried to develop a mechanism that automatically enhances any finite element implementation. Thanks to the two design patterns described in the previous sections, we designed an XFEM implementation where no modifications have been made to the standard toolkit.

The implementation of the XFEM features has thus consisted first in writing a specific problem component. After the mesh initialization, control is given to this XFEM problem component. XFEM-wrappers are then created to encapsulate the finite element objects which are then substituted in the mesh object.

Knowing the discontinuity description (for instance using level sets, or a surfacic mesh of it), the XFEM component is able to try to cut all elements, and decide which nodes have to be enriched and how to enrich them. The nodes are tagged accordingly to the XFEM element wrappers. At this point, the XFEM element wrappers have all needed informations to create, if necessary, a new integration object to be used by their real finite element. A new interpolation object is also created to compute, when needed, the extended shape functions and their gradient. This interpolation object is also used transparently by the real finite element object. By supplying these two objects (integration & interpolation), the code of the wrapped finite element has absolutely not been modified.