Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXVII, págs. 2959-2965 (artículo completo)
Alberto Cardona, Mario Storti, Carlos Zuppa. (Eds.)
San Luis, Argentina, 10-13 Noviembre 2008

UN FRAMEWORK ORIENTADO A OBJETOS PARA LA IMPLEMENTACIÓN DE MÉTODOS DISCRETOS

Diego Santiago[b], Marco Dondero[a] y Santiago Urquiza[b]

[a]División Soldadura y Fractomecánica, INTEMA-CONICET, Universidad Nacional de Mar del Plata, J. B. Justo 4302, 7600, Argentina, mdondero@fi.mdp.edu.ar
[b]Grupo de Ingeniería Asistida por Computadora, Facultad de Ingeniería, Universidad Nacional de Mar del Plata, CONICET, J. B. Justo 4302, 7600, Argentina, dsantiago@fi.mdp.edu.ar

Palabras clave: Programación Orientada a Objetos, métodos numéricos, C++, FORTRAN.

Resumen. En este trabajo se describe una implementación orientada a objetos de un *Framework* para la resolución de problemas derivados de métodos discretos (Método de Elementos Finitos, Método de Diferencias Finitas, Método de Volúmenes Finitos, Método de Elementos de Contorno, etc). Se eligió el lenguaje de programación C++ como plataforma de desarrollo con el objetivo de obtener un código eficiente, fácilmente extensible y mantenible. Este desarrollo se basa en la arquitectura de un Framework preexistente implementado en Fortran del cual se reutilizan procedimientos vinculados al manejo de matrices, a la resolución de sistemas de ecuaciones, así como ciertas abstracciones de datos y lineamientos arquitectónicos. Se describen las principales abstracciones y clases que soportan la arquitectura, las que dan lugar a un diseño altamente modular, que cumple con el principio de inversión de control y que permite la reutilización total del programa principal, sin sufrir alteraciones, cuando se implementan diferentes formulaciones y métodos. Por último, se realizan estudios de desempeño y eficiencia computacional contrastando con la versión original en FORTRAN, sobre la base de casos adecuadamente elegidos. Los resultados obtenidos muestran que la versión en C++ no produce pérdidas apreciables de eficiencia mientras que representa un mejoramiento significativo en modularidad y extensibilidad, constituyéndose de esta manera, en una alternativa de gran potencial y versatilidad para la rápida implementación numérica de todo tipo de formulaciones, conservando los niveles de desempeño de los códigos de propósito específico.

1 INTRODUCCIÓN

Los grandes avances en el desarrollo de hardware ocurridos durante las últimas décadas permitieron a los ingenieros abordar problemas computacionales más complejos y costosos. En consecuencia, la tecnología del software ha evolucionado, aumentando su complejidad de tal manera que el viejo paradigma de programación procedural está siendo reemplazado por tecnologías de software más avanzadas, tales como Programación Orientada a Objetos (POO).

Un concepto surgido del campo de la computación científica, particularmente del paradigma de POO, es el denominado "Application Framework" (AFs). Un AF es un esfuerzo de la arquitectura de diseño orientado a incrementar la reutilización de software. Esta estructura se puede entender como una aplicación incompleta que puede ser especializada para producir aplicaciones particulares.

La reutilización de los AFs esta basado en la arquitectura de diseño de algunos principios generales como la Abstracción de Datos (AD) y en la Estandarización de Interfaces (EI). La AD es un grupo de procedimientos que buscan desarrollar e implementar componentes genéricos compatibles con contextos cambiantes. En el contexto de la POO con C++, la AD está relacionada con las Clases. Por otro lado, el uso de la EI admite mezclar nuevos componentes con los ya existentes. Este concepto permite agregar funcionalidades sin modificar la estructura del software.

Existen una variedad de códigos del Método de Elementos Finitos (MEF) basados en POO, que fundamentalmente, construyen las clases a partir de las estructuras procedurales clásicas (Touzani, 2002 y Martha, 2002). Si bien estos códigos son más fácilmente mantenibles, no constituyen un AF que pueda ser extensible a otros métodos de análisis ya que están asociados a un método numérico específico (MEF). Existen otros desarrollos con enfoques más generales y una mayor flexibilidad para la aplicación de métodos numéricos (Devloo, 1997 y Beal et al., 1999).

En este trabajo, partiendo de un AF anteriormente desarrollado en FORTRAN (Urquiza et al., 2002), se creó un nuevo AF utilizando POO con C++. Esta nueva versión permite resolver una mayor variedad de problemas como se describirá en el punto 2.

2 ARQUITECTURA DEL DISEÑO

Los modelos computacionales de los problemas físicos se pueden ver como una serie de idealizaciones (como muestra la Figura 1), donde cada una de las cuales introduce una aproximación del problema físico inicial. Estas abstracciones se introducen para hacer resoluble el problema planteado (debido a las restricciones del tamaño del problema y/o del tiempo de ejecución).

Se pueden identificar varios niveles de aproximación que surgen de un análisis del problema físico, ver Figura 1. La idealización de nivel más alto es la de aislar el problema físico real interactuando con su ambiente. El siguiente paso es obtener una descripción matemática de dicho problema que introduce cierto nivel de idealización (ej. Ecuaciones Diferenciales (ED)). La descripción del modelo matemático consiste de una definición del dominio (geometría), y una descripción de la interacción del problema físico con su entorno y las propiedades del mismo (condiciones de contorno, restricciones, materiales, etc.). Si la descripción matemática del problema no puede ser resuelta de forma analítica, entonces se deben usar técnicas numéricas.

La construcción de un problema numérico a partir de un problema matemático involucra otro grupo de idealizaciones. Un mismo problema matemático puede ser resuelto por distintos métodos numéricos, cada uno asociado a diferentes idealizaciones.

Finalmente se puede decir que todos los métodos numéricos generan un -o una sucesión de- Sistema de Ecuaciones Lineales ("Linear System of Equations", LSE). De esta manera se obtiene un conjunto de valores que describen la aproximación del problema físico luego de la serie de idealizaciones aplicadas.

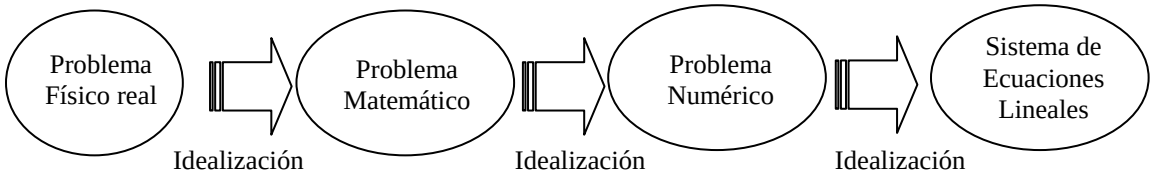


Figura 1: Abstracciones de un problema físico.

Las abstracciones utilizadas para modelar las clases parten del análisis de la construcción del LSE. Es decir, las clases se piensan en función de los pasos requeridos para construir un LSE, independizándose de las abstracciones aplicadas al problema físico. Siguiendo esta filosofía, la implementación se reduce a tres clases básicas. Una asociada a la aproximación de la geometría y propiedades físicas, otra asociada al método numérico, y finalmente una asociada al LSE.

Este programa utiliza abstracciones de una versión anterior realizada en FORTRAN (Urquiza et al., 2002) la cual funciona muy bien para problemas donde el LSE mantiene su estructura simbólica invariante, es decir, el acoplamiento entre las variables no cambia durante la ejecución del programa. Cuando los problemas a resolver involucran un cambio de la estructura simbólica del LSE (e.g., remallado en MEF), se requiere de un gran conocimiento del programa y mucho esfuerzo de desarrollo para realizar esta tarea. Por este motivo se decidió optar por la aplicación de POO para mejorar la versión existente del programa en FORTRAN, haciéndolo más modular y extensible, con vistas a facilitar la implementación de una variedad de problemas tales como los ya mencionados.

Desde el punto de vista del usuario, el tiempo de aprendizaje se reduce notablemente al aplicar el paradigma de la POO y la EI ya que éste sólo tiene que programar el código asociado al acoplamiento entre las variables del elemento (ver punto 3). En otras aplicaciones de programas para MEF (Touzani, 2002 y Martha, 2002) se consideran clases a partir de idealizaciones geométricas, como nodos y elementos, pero dichas clases quedan indefectiblemente vinculadas a un método numérico y/o una formulación particulares, quitándole así generalidad al programa. La estandarización de interfaces y las abstracciones de datos aquí presentadas hacen posible independizar el ensamblaje del LSE de la formulación y geometría particular de los elementos. Esto a su vez, es una de las claves para que el Framework presentado cumpla con el principio de Inversión de Control, alcanzándose niveles altísimos de reutilización del programa principal, que permanece inalterado mientras se agregan elementos de naturaleza radicalmente diferentes en cuanto a geometría y formulaciones, inclusive si se cambia de método numérico (FEM, FVM, FDM, BEM, etc).

3 IMPLEMENTACIÓN

Partiendo del modelo matemático, las abstracciones se pueden separar en dos grandes grupos. La primera contiene la geometría y sus propiedades, y la segunda está asociada a la descripción matemática de los fenómenos involucrados (ver Figura 2). Al pasar al modelo

numérico, el primer grupo se convierte en un conjunto de aproximaciones contenidos en la clase `GeomModel`. De la misma manera la clase `MathModel` contiene las aproximaciones de los modelos matemáticos.

En la clase `GeomModel` el almacenamiento de la información se basa en una serie de abstracciones de datos que surge del MEF. Estas entidades son: *Nodos, Elementos, Grados de Libertad* (GL), *Tipos y Materiales*. La descripción de estas entidades se encuentra bien documentada en (Urquiza et al., 2002). Las instancias de la clase `GeomModel` se generan a partir de archivos de entrada de datos. Sus métodos devuelven punteros a la información de sus miembros (entidades) cuando son requeridos por las instancias de las clases `ALSE` y `MathModel`.

En la clase `ALSE` ("Assemble LSE") se agrupan los miembros y métodos vinculados con el manejo y la resolución del LSE. Esta clase se comunica con `GeomModel` para crear sus miembros. Tanto la parte simbólica como la numérica de los LSE contenidos en el objeto `ALSE` se construye a partir de la información aportada por `MathModel`, que a su vez solicita la información geométrica a `GeomModel` sólo cuando es requerido.

La construcción del LSE, está basada en los aportes de matrices elementales y sus segundos miembros correspondientes. La estrategia utilizada radica en declarar matrices simbólicas elementales asociadas a los *Tipos*. Como cada *Elemento* tiene asociado un *Tipo* por *SubPaso*, al recorrer todos los *Elementos* se obtiene la parte simbólica del LSE del *SubPaso* (ver Urquiza et al., 2002). Las matrices simbólicas asociadas a los *Tipos* no son modificables dentro de `ALSE`, es decir que la estructura de acoplamientos se asume invariable durante la ejecución de los pasos temporales o incrementales asociados al problema. Cualquier cambio en el LSE debe ser planteado en `GeomModel` a partir de una interrupción en el ALSE cuando se requiera la reestructuración de los acoplamientos, e.g., remallado. La parte numérica posee la misma estrategia de ensamblaje que la simbólica, pero aporta una matriz elemental numérica generada en `MathModel`.

Una vez resueltos los LSE, los resultados son escritos por archivo y se actualizan en `GeomModel`. De esta manera, el objeto `GeomModel` contiene los valores calculados asociados a los GL, permitiendo realizar algún proceso relacionado con la reestructuración del problema (ej: remallado en FEM, Métodos Sin Malla, etc).

Los aportes de las matrices elementales numéricas mencionados anteriormente son realizados por los métodos de `MathModel`. La Clase `MathModel` posee una familia de métodos vinculados a un modelo físico y un método numérico asociado al mismo, ver Figura 2. Esta familia de métodos tiene aplicada una EI para facilitar la incorporación de nuevas funcionalidades. Básicamente la EI se reduce a que las comunicaciones entre el `ALSE`, `MathModel` y `Geomodel` se realizan a través de Matrices Elementales y estructuras estándar que representan las propiedades físicas y geométricas, lo que permite independizar la formulación matemática del problema respecto al proceso de ensamblaje de ecuaciones. El objeto `MathModel` es una colección de operaciones que es recorrida por un iterador del objeto `ALSE` que pasa por todos los *Elementos* incluidos en `GeomModel`. Todas las operaciones dentro de `MathModel` comparten su semántica y los nombres de sus operaciones, lo cual es otro aspecto fundamental para lograr la EI, haciendo posible que el usuario sólo tenga la necesidad de aportar el código asociado al método numérico que incluirá, naturalmente, la descripción matemática del modelo físico del problema a resolver.

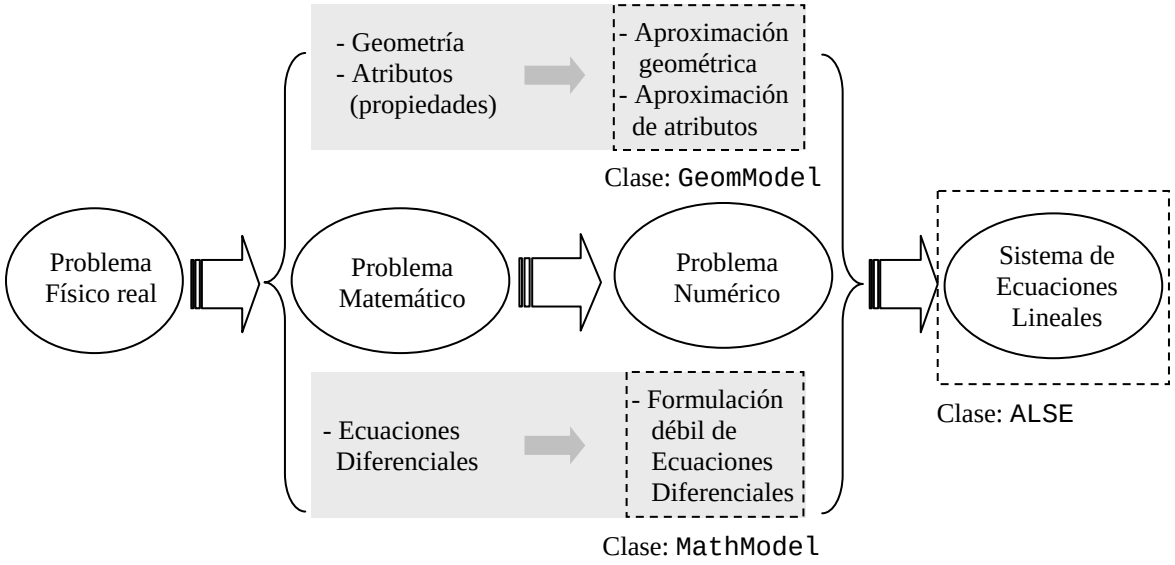
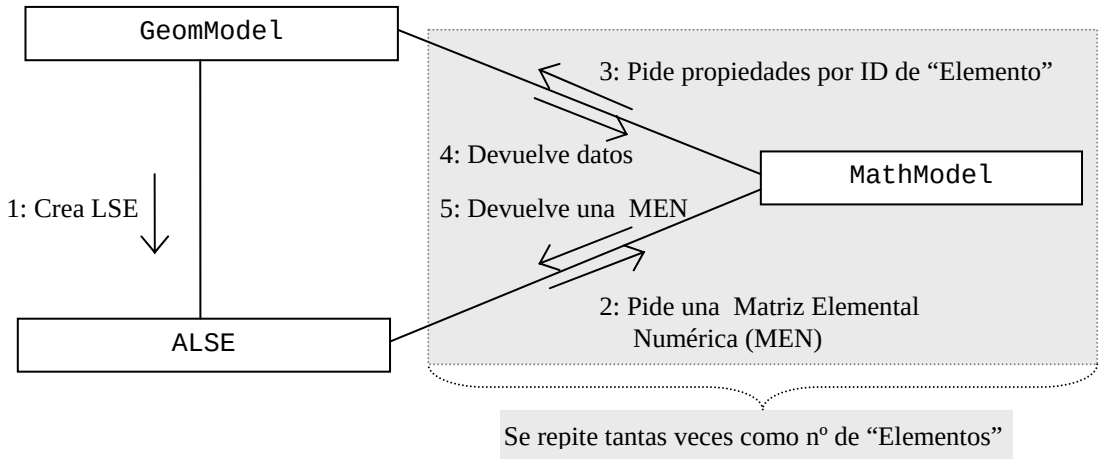
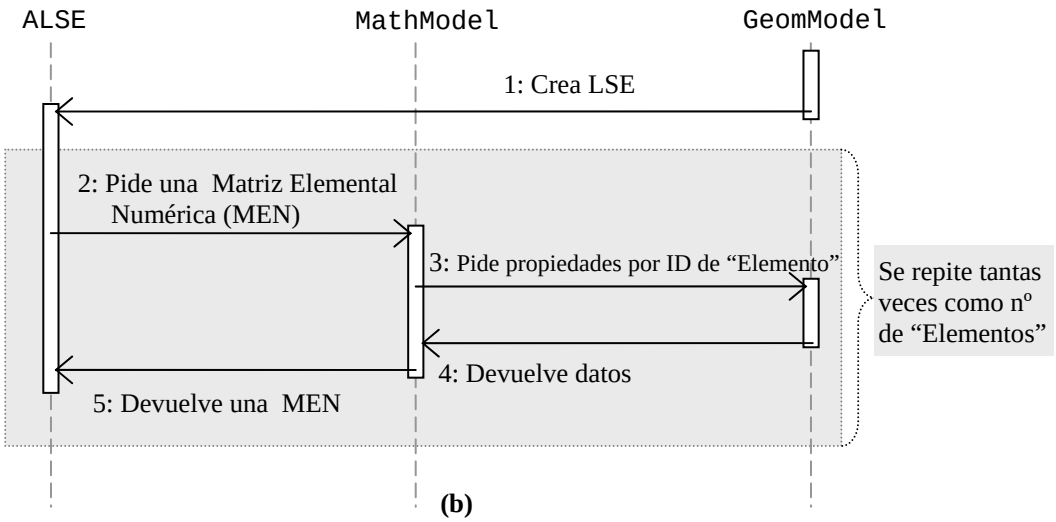


Figura 2**:** detalle de las abstracciones y relaciones con las clases.



(a)



(b)

Figura 3**:** Diagrama de colaboración (a) y de secuencia (b) para la creación de la parte numérica del LSE.

La EI permite la implementación de un vector de punteros a sus métodos (“function pointers”), acelerando la selección dentro de la familia. Esta selección y la ejecución del método se realizan una vez por cada entidad *Elemento* contenida en `GeomModel`. De esta manera este proceso se realiza eficientemente, lo que es determinante para la velocidad de cálculo. Las aplicaciones de computación científica como la presente están desarrolladas para hacer uso intensivo y eficiente de los recursos (memoria y velocidad de acceso de datos). El acceso a datos a través de funciones de métodos de las clases (e.g. “getter” “setter”) tiene un elevado costo computacional comparado con el obtenido mediante punteros. Por este motivo, se decidió acceder a los datos de `GeomModel` haciendo uso intensivo de punteros. Si bien esta implementación violaría, en principio, el ocultamiento de información planteado por el paradigma de la POO, esta decisión esta orientada a minimizar la memoria requerida y reducir el tiempo de acceso a la información minimizando las indirecciones. Es de destacar que aun así se mantiene la modularidad, si bien se le transfiere al programador (el desarrollador del Framework) la responsabilidad del adecuado manejo de los datos en unos pocos casos puntuales que son criticos para el rendimiento de la aplicación. Por otra parte, el implementador (quien personaliza el Framework para una aplicación específica) sólo tiene acceso a los datos locales del *Elemento* (instancia de `MathModel`), con lo cual dichos datos permanecen ocultos a él.

En la Figura 3 (a) se muestra un diagrama de colaboración donde se observa el flujo de la comunicación entre las clases. En este diagrama se denota, en la zona con fondo gris, las comunicaciones para hacer el ensamblaje numérico de una matriz elemental. Este proceso se repite tantas veces como *Elementos* haya. En la Figura 3 (b) se puede ver la misma información en un diagrama de secuencia. Este tipo de diagrama muestra las comunicaciones en tiempo de ejecución del programa dando una idea de la secuencia de funcionamiento del mismo.

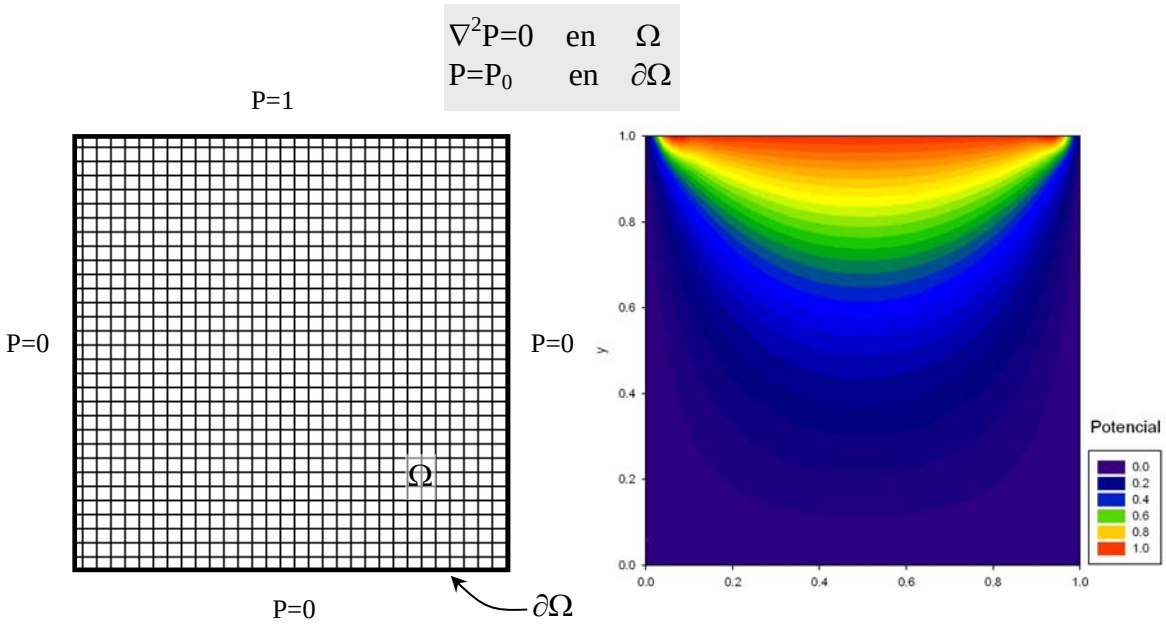


Figura 4**:** Problema implementado: condiciones de contorno (a) y solución (b).

4 RESULTADOS Y DISCUSIONES

Se implementó un problema del tipo de Laplace para una placa rectangular con condiciones de contorno de Dirichlet utilizando el Método de Diferentas Finitas (MDF). La geometría analizada y la solución del problema se observan en la Figura 4.

Se resolvieron casos con diferentes discretizaciones para evaluar el tiempo de ensamblaje de la parte numérica del LSE. Los tiempos obtenidos con el programa aquí descripto se comparan con la versión anterior desarrollada en FORTRAN. En la Tabla 1 se muestran los resultados calculados con un procesador AMD Athlon64 y 1.5 GB RAM. Como se puede observar, los tiempos de ensamblaje son levemente mayores en la versión en C++ (entre un 3% y un 5%). Esta pérdida de eficiencia es despreciable frente a los beneficios obtenidos en estructuración y extensibilidad asociados a la POO.

nº de elementos	Versión procedural (FORTRAN)	Versión OOP (C++)
40.000	0.0625	0.0625
90.000	0.141	0.156
490.000	1.10	1.16
1.000.000	2.52	2.61

Tabla 1**:** Tiempo de ensamblaje (en segundos) de la parte numérica del SEL.

5 CONCLUSIONES

Se logró obtener una versión orientada a objetos de un programa existente íntegramente procedural. Esta versión es más fácilmente reutilizable y extensible que la anterior, sin sacrificar eficiencia.

Las ventajas relativas de esta nueva implementación se resumen en la capacidad de extender la variedad de problemas a resolver. Como se mencionó anteriormente, los métodos numéricos que requieren una reestructuración del SEL (como el MEF con remallado, PFEM, SPH, DEM, etc) se pueden implementar ahora con un menor esfuerzo de programación.

Con la arquitectura aquí expuesta, será posible ahora integrar malladores automaticos y refinadores adaptivos donde las condiciones de borde y otras propiedades fisicas estén basadas en descripciones asociadas a entidades geométricas y no directamente a las mallas discretas. Este punto será objeto de futuros desarrollos conjuntamente con la optimización y paralelización del Framework, en la busqueda de alcanzar altos desempeños conservando la flexibilidad de la herramienta.

REFERENCIAS

M.W. Beall and M.S. Shephard. An object oriented framework for reliable numerical simulations. *Eng. Comp.* 15, pp. 61 – 72, 1999.

P.R.B. Devloo, PZ: an object oriented environment for scientific programming. *Comp. meth. Appl. Mech. Engng.*, 150(1-4):133-153, 1997.

Luiz Fernando Martha. An Object-Oriented Framework for Finite Element Programming. *Fifth World Congress on Computational Mechanics*, Vienna, Austria, July 7–12, 2002.

Rachid Touzani. An Object Oriented Finite Element Toolkit. *Fifth World Congress on Computational Mechanics*, Vienna, Austria, July 7–12, 2002.

Santiago A. Urquiza, Marcelo J. Venere. An application framework architecture for FEM and other related solvers. *Mecánica Computacional XXI*, pp. 3099-3109, 2002.