COBEM 99
XV Congresso Brasileiro de Engenharia Mecânica
15th Brazilian Congress of Mechanical Engineering
22 - 26 de Novembro de 1999 / November 22 - 26, 1999 Águas de Lindóia, São Paulo,

UMA REVISÃO PARCIAL DE ARQUITETURAS ORIENTADAS A OBJETO PARA PROGRAMAS DE ELEMENTOS FINITOS

Rogério José Marczak
Universidade Federal do Rio Grande do Sul, Departamento de Engenharia Mecânica
Rua Sarmento Leite, 425, Porto Alegre - RS, 90050-150, Brasil.
e-mail: rato@mecanica.ufrgs.br

Resumo. *O objetivo deste artigo é discutir alguns trabalhos baseados na aplicação de técnicas de programação orientada a objetos (POO) em programas baseado no método dos elementos finitos (MEF), a fim de destacar algumas vantagens que o uso da POO tem trazido ao desenvolvimento e manutenção de programas de engenharia. Uma revisão bibliográfica parcial é realizada, identificando-se algumas características comuns encontradas em muitos dos programas orientados a objeto (OO) existentes para o MEF. Estas características são utilizadas como base na análise de alguns trabalhos relevantes extraídos da literatura, no âmbito da Mecânica dos Sólidos.*

Palavras-chave: *Programação orientada a objetos, Método dos elementos finitos, Métodos numéricos.*

1. INTRODUÇÃO

As primeiras linguagens de programação orientadas a objeto disponíveis comercialmente surgiram em meados dos anos 80 e seu uso na Engenharia ganhou inpulso no início da década de 90. Embora o uso de encapsulamento de dados já estivesse bastante difundido por linguagens como o C e o Pascal, apenas a partir da normalização de linguagens orientadas a objeto em 1990 e 1991 os fabricantes de compiladores passaram a oferecer pacotes com a confiabilidade necessária para uso difundido de programação com encapsulamento de dados e métodos. A partir daí, é notória a popularização mundial destas linguagens nas mais diversas áreas de aplicação.

A difusão do uso de linguagens OO se deve a uma série de fatores reconhecidos. Os mais claros estão diretamente relacionados ao ciclo de desenvolvimento de programas e seu custo. A constante atualização de códigos utilizados em pesquisa ou comercialmente - a fim de torná-los mais abrangentes e eficazes - atinge atualmente limites práticos muitas vezes intransponíveis. Em alguns casos, trata-se de programas iniciados a mais de trinta anos, desenvolvidos em linguagens já consideradas obsoletas e contendo tipicamente dezenas ou mesmo centenas de milhares de linhas de programação estruturada. Isto criou uma demanda de *extensibilidade* e *reusabilidade* dos programas (ou parte deles) sem incorrer

nos custos de desenvolver programas totalmente novos ou alterar trechos de programas já suficientemente testados e homologados. Esta é uma necessidade relativamente antiga, mas apenas o surgimento da POO propiciou uma solução adequada. Com efeito, esta demanda está intimamente relacionada com as próprias origens de muitas linguagens OO.

Uma descrição detalhada do paradigma de OO foge ao objetivo deste texto. Pressupõe-se um conhecimento básico de programação orientada a objeto, bem como de suas características mais comuns (classes e objetos, encapsulamento de dados e métodos, herança e polimorfismo). Neste trabalho, a hierarquia e o relacionamento entre classes seguirá a notação de Rumbaugh (1991). Uma versão extendida deste trabalho pode ser encontrada em www-gmap.mecanica.ufrgs.br.

2. CARACTERÍSTICAS COMUNS DE ARQUITETURAS OO PARA ELEMENTOS FINITOS

A breve revisão bibliográfica a seguir ilustra muitas das abordagens possíveis para se projetar uma arquitetura OO para programas de elementos finitos. A escolha de uma ou outra abordagem passa necessariamente pela aplicação que se deseja para o programa, seja pesquisa, uso prático em engenharia ou educação. Qualquer que seja o caso, a parte essencial de um programa de elementos finitos são os algorítmos de solução do problema discretizado. É este aspecto que determina a eficiência, a robustez e a estabilidade da(s) solução(ões) obtida(s).

Com a finalidade de discernir as diversas abordagens OO encontradas na literatura, bem como limitar significativamente esta revisão, identifica-se três grandes níveis de abstração encontrados em programas de elementos finitos. São eles:

Baixo nível: Este nível de abstração lida principalmente com ferramentas. Geralmente são identificadas como algorítmos que manipulam entes básicos da álgebra linear, como tensores, matrizes ou vetores. Exemplos típicos são a solução de um sistema linear ou de um problema de autovalores/autovetores. Embora matrizes sejam o caso mais comum, outras entidades também são tratadas neste nível como: palavras, números complexos, funções de interpolação, integração numérica, gerenciamento de memória etc.

Médio nível: Aqui se manipula aqueles entes intermediários, ou seja, que geram e/ou armazenam informações imprescindíveis para solução do problema. São estas informações que relacionam diretamente o problema matemático com o problema físico real. No contexto do MEF, por exemplo, é neste nível que se encontram os nós (pontos no espaço), elementos (geometria), carregamentos, materiais, condições de contorno, superelementos etc.

Alto nível: Referem-se basicamente à estratégia de solução do problema. É neste nível que o problema é solucionado do ponto de vista da equação governante. Aqui se utiliza e se controla as informações geradas nos dois níveis de abstração anteriores para se obter a solução global de um problema descrito por uma malha e um dado material, por exemplo. Esta mesma malha e material pode ser utilizada para se resolver um problema estático não-linear ou um problema dinâmico transiente linear. Em essência, portanto, os dois problemas diferem da forma como as rotinas de alto nível manipularão as informações geradas pelas rotinas de médio nível, utilizando as ferramentas providas pelas rotinas de baixo nível. Em outras palavras, a abstração de alto nível é responsável pelo fluxo das informações durante a solução do problema.

A POO é naturalmente atrativa para organizar a solução de um problema em níveis hierárquicos através de derivação de classes. Dado que as diversas arquiteturas encontradas

Nível de abstração	Denominação
Baixo	Classes auxiliares
Médio	Classes do modelo
Alto	Classes de análise

Tabela 1: Associação genérica entre classes e níveis de abstração.

para programas de engenharia se utilizam de alguma forma dos três níveis discutidos acima, estes serão aqui utilizados para enquadrar qualitativamente em que nível de abstração se encontra uma determinada classe. Com o objetivo de facilitar este enquadramento, a revisão bibliográfica apresentada a seguir faz a associação entre classes e um dos três níveis de abstração de acordo com a denominação dada na tabela 1.

Esta nomenclatura foi escolhida meramente para facilitar a identificação da tarefa básica de uma classe neste trabalho. Classes auxiliares então se referem a classes para matrizes, solução de sistemas, listas de armazenamento etc. As classes do modelo lidam com o modelo computacional de uma forma geral (malha, materiais, condições de contorno, detalhes de geometria etc.), enquanto as classes de análise constituem o algorítmo de solução do problema, gerenciamento dos passos de carregamento em processos incrementais, integração no tempo, dentre outros.

3. ALGUMAS REFERÊNCIAS RELEVANTES

O trabalho de Forde *et al* (1990) é citado como a primeira abordagem do uso de POO em programas de elementos finitos. Basicamente, são implementadas apenas classes do modelo, integradas através de listas. Entretanto, o programa mescla partes em linguagem C e partes em linguagem Object Pascal, o que não permite o uso apropriado de herança e polimorfismo. Este trabalho foi implementado em linguagem C++ por Scholz (1992), mas nenhuma classe de análise é implementada.

Mackie (1992) apresenta uma tentativa de transformar um programa baseado em subrotinas para POO, utilizando linguagem Object Pascal. Miller (1991) apresenta uma discussão similar, mas salientando aspectos como herança, reusabilidade e extensibilidade de programas. Novamente, não são implementadas classes de análise. Feijóo *et al* (1991) implementaram classes básicas para o modelo como: nós, elementos, material, renumeração nodal, numeração dos graus de liberdade etc. Não foram desenvolvidas classes específicas para análise.

Pidaparti e Hudli (1993) apresentam uma proposta simples para análise dinâmica modal e transiente. O modelo é implementado através das classes de médio nível (`Node`, `Isotropic Material`, `Element` etc.). Para a análise, uma classe-mãe `DynamicAnalysis` é usada como base para derivação de duas classes abstratas: `EigenSolution` e `Direct_Integration`, das quais derivam classes que fornecem diversos métodos de solução.

Zimmermann e seus colaboradores (Zimmermann alli, 1992)(Pèlerin e Zimmermann, 1993) desenvolveram e implementaram uma arquitetura OO bastante completa e eficiente para análise por elemento finitos. Embora não contenha um nível de abstração muito elevado, a estrutura de classes deste trabalho reflete visivelmente a experiência prática dos autores com o método dos elementos finitos. A arquitetura apresenta dois níveis básicos de programação: a classe `Domain` e a classe `FemComponent`. A classe `Domain` é responsável pelo gerenciamento do modelo de elementos finitos e da solução das equações (análise), enquanto da classe `FemComponent` são derivadas diversas sub-classes (modelo):

`Element`, `Node`, `Material`,`Load` etc. Desta estrutura básica derivam ou se comunicam diversas outras classes. A Figura 1 ilustra a hierarquia básica.

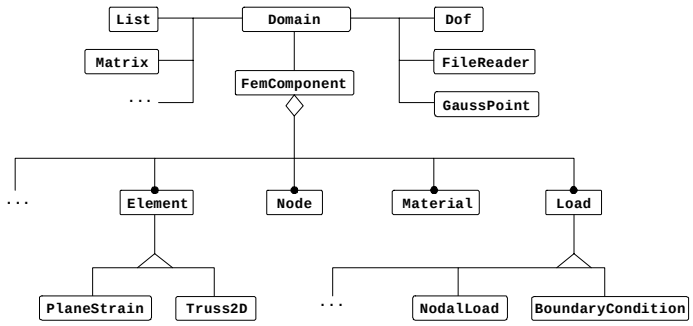

Figura 1: Hierarquia de classes de Zimmermann *at al* (1993).

Uma característica interessante da abordagem de Zimmermann *et al* (1993) é o uso do *princípio da não antecipação*, segundo o qual não se assume *a priori* a existência de qualquer entidade, sejam nós, elementos, carregamentos ou condições de contorno. As próprias classes se encarregam de procurar os dados necessários para construir seus objetos e, se existirem, o que fazer com estes objetos após sua criação. Muito embora avançada, a idéia cria situações um tanto quanto absurdas como o seguinte programa de elementos finitos

```
main {
      Domain Meu_Problema;
      Meu_Problema.solveYourself();
     }
```

Desta forma não resta muito para o usuário fazer caso se deseje utilizar esta arquitetura na solução de um problema não contemplado pelas classes já implementadas. Embora já esteja mostrando sinais de obsolescência, esta arquitetura ainda é considerada uma das mais avançadas abordagens de POO para elementos finitos.

Hedegal (1994) propôs uma estruturação excepcionalmente simples e versátil para implementação de elementos finitos para análise linear e não-linear de problemas de potencial e elasticidade em duas ou três dimensões, chamada `ObjectFEM`. Este trabalho é atrativo pela sua simplicidade e faz uso intensivo de métodos virtuais a fim de permitir a extensão das classes apresentadas. Basicamente, as classes do modelo são implementadas através de nós, elementos, material e propriedades (Figura 2). Os nós e elementos se comunicam, mas apenas os elementos têm acesso aos objetos material e propriedade. As classes auxiliares contam com classes de armazenamento baseadas em listas ligadas e também com classes para manipulação de entidades algébricas. O uso de herança e polimorfismo permite que o usuário regule a profundidade da hierarquia das classes principais (Figura 2). Esta arquitetura conta ainda com classes especiais como a classe `GaussPoint`, que além das funções usuais relacionadas à integração numérica pode armazenar tensores constitutivos, deformações plásticas, tensões e outras variáveis de interesse nos pontos de integração, auxiliando na solução de problemas não-lineares. A classe `Element` realiza as tarefas de montagem das matrizes relevantes, incluindo as necessárias para implementação de formulações Lagrangeanas totais ou atualizadas. Problemas de não-linearidade geométrica e material são mostrados para elementos de barra e de elasticidade bidimensional

(Hedegal, 1994). Não existem classes de análise. São sugeridas funções implementatas pelo usuário (retângulo tracejado na Figura 2) que podem ser facilmente transformadas em classes de análise.

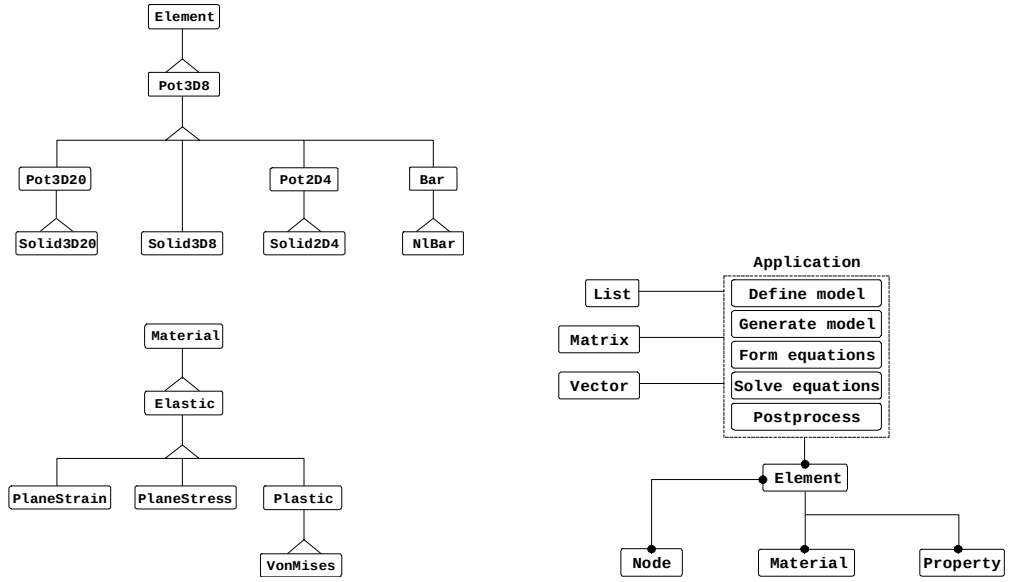

Figura 2: (a) Exemplos da hierarquia de classes proposta por Hedegal (1994). (b) Relação entre as classes de maior nível (o bloco `Application` se refere à parte implementada pelo usuário).

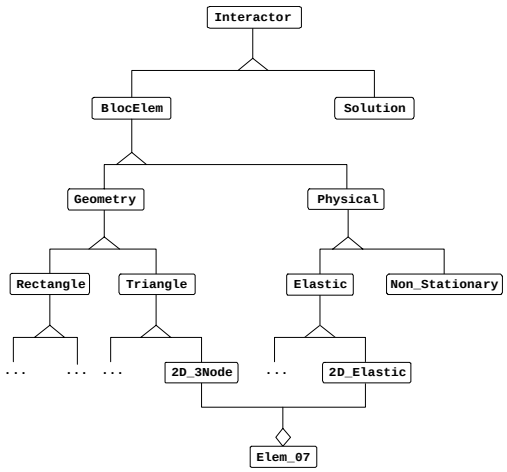

Figura 3: Hierarquia de classes proposta por Kong (1995).

Zeglinski *et al* (1994) implementaram classes auxiliares para manipulação de matrizes que ocorrem tipicamente em elementos finitos. Este trabalho sugere que a inclusão de novos elementos finitos em um programa OO deva fazer uso de derivação simples de classes básicas, a fim de permitir maior flexibilidade do programa na expansão da biblioteca de elementos. A hierarquia das classes é bem rasa, e não são implementadas classes de análise. Um exemplo de implementação em C++ é apresentado e este é injustamente comparado com o equivalente em Fortran. Os trabalhos de Kong *et al* (1995) mostram uma implementação mais geral, voltada para a manipulação dos diferentes tipos de entidades

encontradas em programas de elementos finitos. É oportuno notar que este trabalho implementa uma hierarquia interessante para os elementos finitos, através de uma classe-mãe denominada `BlocElem`. Duas classes derivam de `BlockElem`: uma relativa à topologia do elemento (classe `Geometry`) e outra relativa ao tipo de equação governante do problema (classe `Physical`). Assim, as diversas topologias podem ser combinadas com o problema desejado a fim de prover um elemento específico (Figura 3).

Uma área que ganhou um significativo impulso com o advendo da POO é possibilidade de implementação robusta e extensão de programas de elementos finitos adaptativos. Devloo (1997) propõe um ambiente bastante completo para análise de problemas através do MEF. Além das classes de análise (descrição da equação diferencial), as classes do modelo foram divididas em modelo geométrico e modelo computacional. O modelo geométrico é governado pela classe `TGeoGrid`, e manipula três entidades geoméricas: nós (classe `TGeoNod`), elementos (classe `TGeoEl`) e nós do contorno da malha (classe `TGeoNodBc`), cujos objetos são armazenados em listas binárias. O modelo computacional é gerenciado pela classe `TCompGrid`, que armazena listas de graus de liberdade (classe `TDofNod`), material (classe `TMaterial`), condições de contorno (classe `TBndCond`) etc. Esta classe deriva da classe do modelo geométrico (`TGeoGrid`), o que permite que a mesma malha geométrica seja utilizada para geração de malhas computacionais diferentes (ideal para adaptatividade h) não-simulatâneas. É neste nível que são calculadas as matrizes do problema, mapeamento de elementos, funções de interpolação e imposição de condições de contorno, dentre outras coisas. A análise é basicamente implementada pelo usuário, através dos vários métodos disponíveis em uma classe-mãe auxiliar para matrizes (`Tmatrix`). Outras classes úteis também são implementadas como integração numérica (`TIntRule`), pré-processamento e pós-processamento. A figura 4 ilustra as classes `TGeoEl`, `TCompEl`, `TMaterial` e `TIntRule` deste trabalho, mostrando um nível relativamente raso de hierarquia.

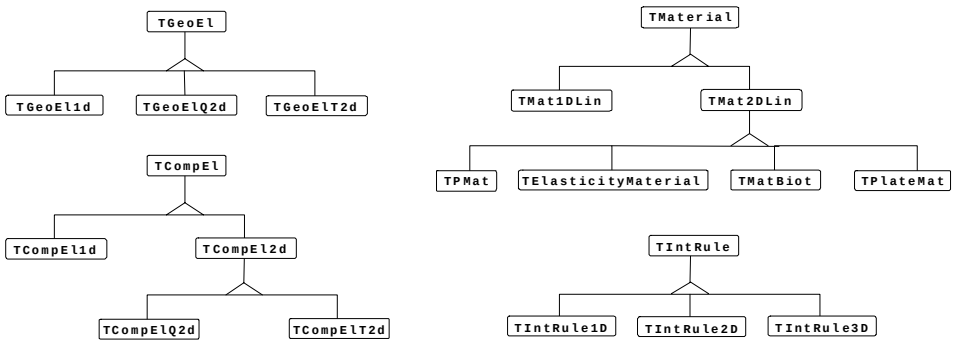

Figura 4: Hierarquia de algumas das classes de Devloo (1997).

A maioria dos trabalhos referentes a aplicação de POO em elementos finitos ainda se limita à análise linear estática e dinâmica de estruturas. Entretanto, alguns trabalhos têm ampliado a aplicaç ão de POO para problemas estruturais não-lineares e mesmo para problemas de outras áreas. Menétrey e Zimmermann (1993) extendem os trabalhos anteriores de Zimmermann *et al* (Zimmermann alli, 1992)(Pèlerin e Zimmermann, 1993) para problemas de plasticidade. No entanto, esta extensão torna obrigatória a redefinição de muitas das classes da arquitetura original. O trabalho de Olsson (1998) utiliza uma implementação em C++ para problemas geometricamente não-lineares de vigas. Além das classes usuais para o modelo, são descritos os métodos de duas classes para análise,

responsáveis pelo gerenciamento do processo incremental ao longo de uma trajetória de equilíbrio não-linear (*path-following*).

As publicações acima demonstram algumas das diversas possibilidades de implementação de programas de elementos finitos OO. Entretanto, todas possuem algum nível de dependência entre suas classes, possivelmente devido à cultura de programação em linguagens convencionais como Fortran, Pascal e C. Mais recentemente, alguns trabalhos têm destacado a necessidade de se isolar o máximo possível as diversas classes umas das outras, em especial as classes responsáveis pelo modelo das classes responsáveis pela análise. Isto é consequência direta da necessidade de se realizar alterações e extensões nos programas, a fim de capacitá-los para outros tipos de análise. Caso exista uma interdependência muito grande entre as classes, mesmo aquelas modificações mais simples podem ficar comprometidas. O trabalho de Menétrey e Zimmermann (1993) é um exemplo concreto disto, onde a extensão de um programa para problemas de plasticidade envolveu modificações nas classes originais. Embora isto seja tolerável para firmas que desenvolvem softwares, impõe uma grande limitação para usuários de bibliotecas de classes, caso não se tratem de classes abstratas suficientemente genéricas.

Os trabalhos de Archer *et al* (1996) apresentam seis classes básicas para análise linear e não linear de problemas estáticos ou dinâmicos: `Analysis` (análise) , `Model` (modelo), `ConstraintHandler`, `ReorderHandler`, `Map` e `MatrixHandler` (Figura 5.a). Esta proposta apresenta uma hierarquia de classes rasa, dificultando a reutilização do código. Por outro lado, os autores obtiveram sucesso em isolar o modelo de elementos finitos da análise propriamente dita. Isto é realizado através de uma classe denominada `Map`, que permite a comunicação entre as classes `Model` e `Analysis` auxiliada por classes de manipulação de graus de liberdade, de imposição de restrições e de matrizes. Demais classes derivam destas classes básicas. Também foram desenvolvidas classes para subestruturas.

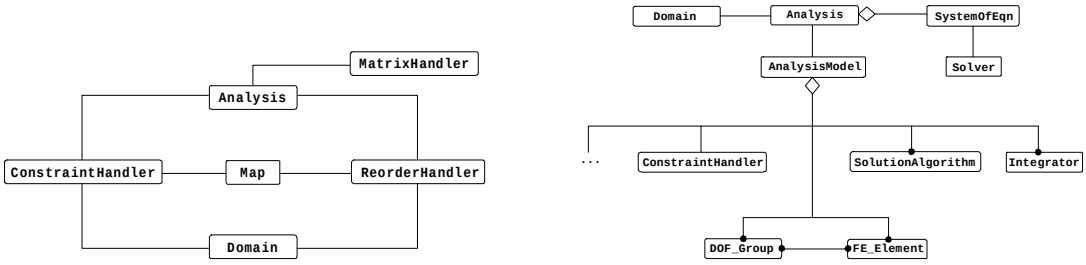

Figura 5: (a) Hierarquia das classes de maior nível de Archer (1996). (b) Hierarquia de classes de maior nível de McKeena (1997).

Besson *et al* (1997) propõem uma arquitetura bastante completa, compreendendo classes de modelo e de análise integradas em uma classe-mãe `Probem`, além de diversas classes auxiliares. Esta abordagem foi criada especificamente para facilitar o desenvolvimento a longo prazo de softwares de elementos finitos. A hierarquia das classes é relativamente ampla e foi projetada para reduzir a interdependência entre as classes, facilitando a expansão do código. A proposta é dotada de classes abstratas para solução de casos bastante específicos como problemas de contato e otimização, além de algorítmos para controle de convergência de processos incrementais. Os detalhes de sua implementação são omitidos.

O trabalho de McKeena (1997) constitui um exemplo representativo do alto grau de evolução que as arquiteturas OO para elementos finitos atingem atualmente. Trata-se de um projeto bastante avançado, incluindo classes de modelo e de análise muito bem desen-

volvidas e independentes, além de classes para decomposição de domínio (sub-estruturas) e classes voltadas especificamente para processamento paralelo. A profundidade da hierarquia de classes é a mais alta dos trabalhos aqui analisados, garantindo um ótimo nível de extensibilidade e reusabilidade do código (Figura 5.b). As classes do modelo são gerenciadas por uma classe chamada `AnalysisModel`, que se encarrega de separá-las das classes de análise e fazer a comunicação entre elas. Esta arquitetura é dotada de classes de análise especiais divididas em dois grupos: tipo de análise e tipo de algorítmo a ser usado. Isto é realizado por duas superclasses: `Analysis` e `SolutionAlgorithm`. Da classe `Analysis` derivam diversos tipos de análise possíveis: análise estática, modal, transiente etc., enquanto da classe `SolutionAlgorithm` derivam subclasses que executam os diversos algorítmos requeridos: Linear, Newton-Raphson, BFGS etc. Isto permite que um dado tipo de análise possa ser realizada por diferentes algorítmos, se desejado (Figura 6). Adicionalmente, as classes `SystemOfEqn` e `Solver` permitem fazer uso da esparcidade típica das matrizes de elementos finitos e adotar diversos métodos de solução do sistema de equações. A Figura 6 ilustra uma parte desta arquitetura, detalhando um pouco mais algumas das classes ilustradas na Figura 5.b.

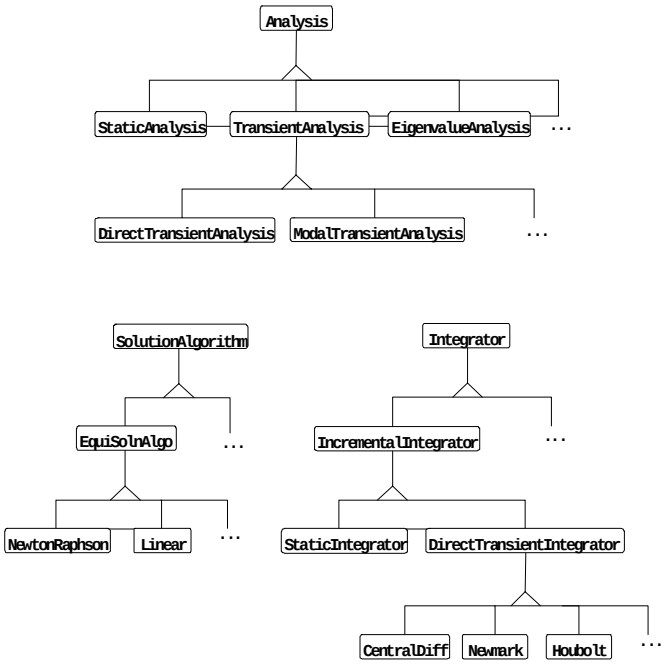

Figura 6: Hierarquia de algumas subclasses da arquitetura de McKeena (1997) (ver Figura 5.b).

Quanto às classes auxiliares, no contexto do presente trabalho, estas são geralmente constituídas por classes para manipulação de matrizes (incluindo solução de sistemas), vetores (incluindo operações algébricas básicas) e listas (incluindo o gerencialmento das mesmas: inserção, remoção e modificação de termos), além de outras ferramentas. Excelentes bibliotecas para manipulação de matrizes podem ser encontradas na literatura. Obviamente, outros tipos também são encontrados, dependendo da aplicação, mas estas três categorias englobam a grande maioria das classes auxiliares. As bibliotecas de classes de Lu *et al* (1995), Scholz (1992) e Zeglinski *et al* (1994) são bons exemplos da farta literatura sobre classes para matrizes. Outros exemplos de arquiteturas gerais podem ser encontradas em pacotes comerciais e de domínio público, que em geral oferecem uma interface bastante ampla para o usuário (Beck alli, 1995)(Langtangen alli, 1996).

4. Conclusões

Este trabalho apresentou uma pequena revisão bibliográfica de alguns trabalhos de POO bastante citados na literatura de elementos finitos. Embora limitada pelo espaço, a revisão ilustra o grande universo de possibilidades de implementação de filosofias OO utilizadas com o MEF. Alguns exemplos interessantes de arquiteturas são explorados um pouco mais, fornecendo idéias para o desenvolvimento de arquiteturas similares. Espera-se que este artigo possa auxiliar profissionais interessados em POO aplicado ao MEF fornecendo uma lista de referências iniciais para estudo. Referências adicionais podem ser encontradas em outras revisões bibliográficas (Hedegal, 1994)(Archer, 1996)(McKeena, 1997)

Referências

J. Rumbaugh, M. Blaha, W. Premerhani, F. Eddy, e W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

B. W. R. Forde, R. O. Foschi, e S. F. Steimer. Object-oriented finite element analysis.*Computers & Structures*, 34(3):355–374, 1990.

S. P. Scholz. Elements of an object-oriented fem++ program in C++. *Computers & Structures*, 43(3):517–529, 1992.

R. I. Mackie. Object oriented programming of the finite element method. *Int. J. Num. Meth. Engng.*, 35:425–436, 1992.

G. R. Miller. An object-oriented approach to structural analysis and design. *Computers & Structures*, 40(1):75–82, 1991.

R. A. Feijóo, A. C. S. Guimarães e E. A. Fancello. Algunas experiencias en la programacion orientada por objetos y su aplicacion en el metodo de los elemento finitos. Technical report, LNCC - Laboratório Nacional de Computação Científica - Relatório 015/91, 1991.

R. M. V. Pidaparti e A. V. Hudli. Dynamic analysis of structures using object-oriented techniques. *Computers & Structures*, 49(1):149–156, 1993.

T. Zimmermann, Y. Dubois-Pèlerin, e P. Bomme. Object-oriented finite element programming: I. governing principles. *Comput. Methods App. Mech. Engrg.*, 98:291–303, 1992.

Y. Dubois-Pèlerin e T. Zimmermann. Object-oriented finite element programming: Iii. an efficient implementation in C++. *Comput. Methods App. Mech. Engrg.*, 108:165–183, 1993.

O. Hedegal. *Object-Oriented Structuring of Finite Elements*. PhD thesis, Aalborg University, 1994.

G. W. Zeglinski, R. P. S. Han, e P. Aitchison. Object-oriented matrix classes for use in a finite element code using C++. *Int. J. Num. Meth. Engng.*, 37:3921–3937, 1994.

X. A. Kong e D. P. Chen. An object-oriented design of fem programs. *Computers & Structures*, 57(1):157–166, 1995.

P. R. B. Devloo. Pz: An object oriented environment for scientific programming. *Comput. Methods Appl. Mech. Engrg.*, 150:133–153, 1997.

P. Menétrey e T. Zimmermann. Object-oriented non-linear finite element analysis: Application to J2 plasticity. *Computers & Structures*, 49(5):767–777, 1993.

A. Olsson. An object-oriented implementation of structural path-following. *Comput. Methods Appl. Mech. Engrg.*, 161:19–47, 1998.

G. C. Archer. *Object-Oriented Finite Element Analysis*. PhD thesis, University of California at Berkeley, 1996.

J. Besson, R. Foerch, G. Cailletaut, K. Aazizou, e F. Hourlier. Large scale object oriented finite element code design. 1997.

F. T. McKenna. *Object-Oriented Finite Element Programming: Frameworks for Analysis, Algorithms and Parallel Computing*. PhD thesis, University of California at Berkeley, 1997.

J. Lu, D. W. White, W. F. Chen, e H. E. Dunsmore. A matrix class library in C++ for structural engineering computing. *Computers & Structures*, 55(1):95–111, 1995.

R. Beck, B. Erdmann, e R. Roitzsch. Kaskade 3.0 - an object-oriented adaptive finite element code. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, TR 95-4, 1995.

H. P. Langtangen. Details of finite element programming in diffpack. Technical report, Department of Mathematics, University of Oslo, 1996.

A PARTIAL BIBLIOGRAPHICAL REVIEW OF OBJECT ORIENTED ARCHITECTURES FOR FINITE ELEMENT SOFTWARE

Rogério José Marczak
Universidade Federal do Rio Grande do Sul, Departamento de Engenharia Mecânica
Rua Sarmento Leite, 425, Porto Alegre - RS, 90050-150, Brasil
e-mail: rato@mecanica.ufrgs.br

Abstract: *The objective of this paper is to discuss some works based on the application of object oriented programming (OOP) in finite element (FE) software development. This review highlights some of the advantages that the use of OOP brings in engineering software development and maintenance. A critical review of some relevant references allow the identification of a few common characteristics found in several object oriented finite element codes, which are then used as a base to further analyze references commonly cited in the Solid Mechanics literature.*

Keywords: *Object oriented programming, Finite element method, Numerical methods.*