Vol. 15, 3, 343-355 (1999)

Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería

Elementos finitos orientado por objetos

Marco L. Bittencourt

Departamento de Projeto Mecânico
Faculdade de Engenharia Mecânica
Universidade Estadual de Campinas
Caixa Postal 6051, Campinas, SP, 13083-970, Brasil
e-mail: mlb@fem.unicamp.br

Antônio S. Guimarães e Raúl A. Feijóo

Laboratório Nacional de Computação Científica (LNCC/CNPq)
Av. Getulio Vargas 333, Quintadinha
25651-070, Petrópolis, RJ - Brasil
Tel.: 55-24-233 60 17/61 57, Fax: 55-24-233 61 65/231 55 95
e-mail: feij@alpha.lncc.br

Sumário

Este artigo considera o desenvolvimento de programas para análise estrutural usando programação orientada por objetos. Apresentam-se uma revisão bibliográfica, conceitos de orientação por objetos e classes para análise linear por elementos finitos. Finalmente, dois ambientes para problemas elásticos bidimensionais são considerados, os quais integram ferramentas para a definição de geometria, geração automática de malhas, solução, estimadores de erro, visualização de resultados e interfaces gráficas.

OBJECT-ORIENTED FINITE ELEMENTS

Summary

This paper presents the development of structural analysis softwares based on object-oriented programming. A bibliographical review, object-oriented concepts and classes for linear finite element analysis are presented. Finally , two-dimensional elastic environments are considered, which integrate tools for geometry definition, automatic mesh generation, solver, error estimation, visualization of results and graphic user interfaces.

INTRODUÇÃO

A implementação computacional de métodos numéricos, tais como o Método de Elementos Finitos (MEF), para a análise de problemas de engenharia é de fundamental importância. Sob o ponto de vista do usuário , o programa deve possuir uma interface de utilização simples, ser confiável e eficiente. No que se refere à implementação, desejam-se características como modulação, extensibilidade, fácil manutenção, dentre outras.

Apesar de vários progressos, a produtividade no desenvolvimento de softwares ainda é pequena, comparado ao crescimento vertiginoso da indústria de hardware nos últimos tempos. Desta maneira, torna-se essencial disciplinar a implementação de programas aplicando por exemplo conceitos de engenharia de software[24]. No caso do meio científico, estes requisitos são importantes para uma maior qualificação dos programas, evitando a repetição de procedimentos em cada trabalho realizado. Desta forma, através de uma maior organização, pode-se ter uma base sólida de programas, permitindo a sua rápida extensão para o estudo de novos problemas.

 ISSN: 0213-1315 Recibido: Marzo 1998

O modelo de programação orientada por objetos[17] tem possibilitado aumentar a produtividade no desenvolvimento de programas. Basicamente, identificam-se os agentes principais da base de conhecimento a ser reprentada no programa. Estes agentes ou objetos possuem atributos e métodos descrevendo as características e os comportamentos de interesse. Assim, num programa tem-se um conjunto de objetos implementados através de classes, as quais constituem-se em moldes para a definição dos objetos como instâncias de uma classe. Por sua vez, duas instâncias diferem entre si pelos valores atribuídos as suas variáveis. Desta maneira, as classes são a unidade básica de modulação num sistema por objetos.

Os objetos se comunicam através do envio de mensagens, as quais especificam apenas a ação a ser feita. É responsabilidade do objeto receptor interpretar a mensagem e executar o conjunto de operações correspondente, retornando o resultado para a instância que emitiu a mensagem. Logo, numa classe tem-se o encapsulamento da informação, a qual pode ser acessada apenas através do envio de uma mensagem para o método desejado. Observa-se ainda que uma mesma mensagem pode endereçar diferentes procedimentos dependendo da classe receptora. Por exemplo, a operação + para matrizes representa uma adição usual, enquanto para um conjunto de caracteres poderia significar concatenação. Esta característica é denominada polimorfismo.

Além disso, tem-se o conceito de herança permitindo especializar a informação ao longo de uma hierarquia de classes, onde aquelas situadas em níveis mais baixos herdam as características (dados + métodos) daquelas classes situadas acima. Assim, é possível estender ou reutilizar classes em várias aplicações, bastando acrescentar apenas as variáveis e métodos necessários para descrever a especialização desejada.

A principal diferença do paradigma por objetos é a proximidade com os conceitos do mundo real a ser implementado no programa. Ao contrário do modelo por procedimentos, onde se isolam os comportamentos através de subrotinas, tem-se entidades próprias descrevendo as suas principais características através de variáveis e métodos.

Estes conceitos têm sido aplicados na análise estrutural de problemas de engenharia, podendo-se citar alguns trabalhos iniciais[17,23,15,6,11,29]. Um dos primeiros programas implementados está discutido em[6], usando a linguagem Object NAP baseada em rotinas escritas em C e Pascal. Em[11], apresentam-se conceitos gerais de programação por objetos, incluindo aspectos sobre concorrência, processamento distribuído e banco de dados, além de um programa para análise por elementos finitos implementado com uma extensão da linguagem LISP, denominada Flavors.

Conceitos do modelo por objetos aplicados ao MEF são também discutidos em[29], apresentando-se uma extensão da hierarquia de classes da linguagem Smalltalk para o caso do MEF. Detalhes de implementação deste programa protótipo estão dados em[32]. Devido a baixa eficiência do ambiente Smalltalk, desenvolveu-se em[33] uma versão do mesmo programa em C++[25]. Uma extensão para o tratamento de problemas de plasticidade está dada em[22].

Aplicações da linguagem C++ com diferentes enfoques também podem ser encontradas em[5,27,28]. Em[12], tem-se uma hierarquia de classes para o tratamento de vários tipos de matrizes, tais como simétrica, esparsa, coluna, dentre outras, além de uma série de classes para o tratamento de entidades do MEF. Já em[3], tem-se um ambiente em C++ constituído de um interpretador e um módulo de execução. Através de uma linguagem interpretada, pode-se especificar interativamente os parâmetros do modelo de elementos finitos e da solução. Exemplos de problemas lineares e não-lineares são resolvidos.

Em[14,13], apresentam-se aspectos da programação por objetos aplicados a análise numérica, representação gráfica das classes, bancos de dados, incluindo ainda a aplicação destes conceitos na implementação de um programa para o tratamento de laminas de materiais compostos. Além disso, aspectos de inteligência artificial são também abordados em conjunto com a análise estrutural. Esta temática também é discutida em[30,31], apresentando uma série de ferramentas computacionais para automatizar o processo de análise,

fornecendo uma base de conhecimentos e sistemas especialistas para auxiliar a especificação de parâmetros da análise, interpretação dos resultados, procedimentos adaptáveis, tratamento de erros, dentre outros. O objetivo básico é acumular um conjunto de conhecimentos de um especialista, auxiliando o usuário comum na simulação computacional de problemas mecânicos, tornando o ambiente de análise mais *inteligente*.

Um grande esforço no desenvolvimento de programas para engenharia estrutural tem sido realizado por um conjunto de pesquisadores, começando com a linguagem Fortran[10], passando para C[1] e finalmente utilizando o modelo de programação orientada por objetos[18,9,8,23,16,2] através da linguagem C++. De forma geral, tem-se bibliotecas de classes e procedimentos para bancos de dados, tratamento de erros, estruturas de dados, manipulação de matrizes e vetores, rotinas matemáticas[1], além de conjuntos de classes para definição de contornos geométricos[16], geração automática de malhas[9,8], análise linear por elementos finitos[18,23], métodos multigrid[18,20], visualização de resultados[2,26], otimização estrutural[7], dentre outras.

A contribuição deste trabalho está relacionada a uma nova proposta de organização das classes para elementos finitos baseada no conceito de tipos parametrizados disponíveis em C++. Consideram-se ainda os sistemas SAFE[2] e SAT[18] para análise de problemas elásticos bidimensionais.

Neste trabalho, apresentam-se inicialmente alguns conceitos do modelo orientado por objetos, tais como modulação, abstração, classes, métodos, dentre outros. Posteriormente, considera-se um conjunto de classes para análise linear de problemas elásticos, empregando o conceito de tipos parametrizados da linguagem C++. Finalmente, discutem-se dois ambientes para análise de problemas bidimensionais com ferramentas para definição de contornos, geração automática de malhas, visualização de resultados e análise adaptável, integradas através de interfaces gráficas.

CONCEITOS DE PROGRAMAÇÃO POR OBJETOS

O modelo orientado por objetos, de maneira análoga aos vários paradigmas de programação, introduz alguns conceitos particulares, tais como objetos, mensagens, mecanismo de herança, dentre outros. Ressalta-se, porém, que nem todas as linguagens orientadas por objetos implementam todos os conceitos a serem descritos. Assim, a apresentação feita a seguir tem um caráter mais geral, não procurando destacar as particularidades de uma linguagem[17].

Modulação: o conceito de modulação é de certa forma intuitivo: dado um problema complexo, subdivide-se o mesmo em subproblemas menos complexos visando obter a solução global.

No entanto, a aplicação do conceito de modulação não pode ser realizada indefinidamente no desenvolvimento de um programa. Ao mesmo tempo em que o esforço de desenvolvimento diminui substancialmente quando se aumenta o número de módulos, o esforço de interfaceamento destes módulos cresce na mesma proporção. Alguns tipos de módulos são encontrados em linguagens de programação, como por exemplo a declaração `class` em C++. Este módulos são componentes de programas que combinam abstrações de dados e procedimentos, incentivando assim, o desenvolvimento de programas modulares.

Ocultamento de informação: a aplicação do conceito de ocultamento de informação no desenvolvimento de um programa permite construir módulos onde a interdependência entre os mesmos é pequena. Além disso, aumenta-se a confiabilidade e as modificações são efetuadas localmente dentro de cada módulo, preservando assim, a disseminação das alterações ao longo de todo o sistema.

Para se alcançar uma modulação efetiva, define-se um conjunto de módulos interdependentes, os quais se comunicam entre si apenas através das informações necessárias para

acessar uma determinada característica. O estado de um módulo é descrito por variáveis locais, visíveis apenas dentro do escopo deste módulo, e um conjunto de procedimentos que manipulam estes dados. Observa-se que o uso de abstrações de dados permite definir e utilizar o conceito de ocultamento de informação no desenvolvimento de programas.

Abstração: para problemas onde se aplicam o conceito de modulação, vários níveis de abstração podem ser considerados. No nível mais alto de abstração, a solução adotada para o problema é colocada em termos de uma linguagem próxima ao ambiente do problema. Nos níveis mais baixos, descrevem-se os procedimetos a serem implementados. Assim, o uso de abstração permite ao programador concentrar-se em um problema, considerando um nível de generalização qualquer, sem se preocupar com detalhes irrelevantes ao problema.

Várias linguagens de programação, tais como *Ada*, *Modula* e *Smalltalk*, permitem a criação de tipos abstratos de dados, os quais consistem de uma representação interna para os dados e um conjunto de procedimentos para acessar e manipular os dados.

Ligação dinâmica: nas linguagens convencionais, tais como *C*, *Pascal* e *FORTRAN*, a partir do conhecimento dos tipos de variáveis e constantes utilizadas em uma declaração, o compilador gera o código de máquina correspondente. Este processo de definir os tipos de dados aplicáveis a um operador em uma declaração, anterior a sua execução, é denominado *ligação estática*.

Entretanto, algumas linguagens, como por exemplo C++, permitem a flexibilidade de redefinir operadores convencionais para os tipos declarados pelo usuário. Assim, pode-se definir o tipo `Vetor`, onde a soma de dois vetores `A` e `B` é expressa por `A+B`. No entanto, esta ligação entre tipos e operandos é realizada ainda, em tempo de compilação do código fonte.

A ligação dinâmica permite associar um tipo de dado a um operando em tempo de execução do programa. Em C++, esta característica está implementada através de declarações do tipo `virtual`.

Objeto: um objeto se constitui na unidade básica de modulação no modelo orientado por objetos, constituindo-se um tipo abstrato de dados.

Para se manipular a informação representada por um objeto, deve-se solicitar ao mesmo que execute uma de suas operações, através do envio de uma mensagem. O objeto que recebe a mensagem é denominado receptor, devendo responder esta mensagem através da seleção da função correspondente, executar esta operação e retornar o controle para o objeto emissor da mensagem.

Mensagens: constituem-se nas especificações das operações de um objeto. Assim, quando um objeto recebe uma mensagem, deve determinar como manipulá-la para obter a resposta requerida. Uma mensagem inclui um seletor, descrevendo o tipo de manipulação desejada, e argumentos, os quais podem ser outros objetos ou valores das variáveis de um objeto.

A característica principal do mecanismo de mensagem é que o seletor é um nome de uma operação, descrevendo apenas a ação a ser executada. Portanto, uma mesma mensagem pode ser interpretada de maneiras distintas. Assim, por exemplo, considere as classes `Vector` e `String`. Definindo-se os objetos `Vector A,B` e `String Str1,Str2`, as instruções `A+B` e `Str1+Str2`, apesar de utilizarem o mesmo operador, possuem significados diferentes.

Classes e instâncias: vários sistemas orientados por objetos fazem uma distinção entre um objeto e a sua descrição. Assim, definem-se os conceitos de classe e instância.

Uma classe é uma descrição geral de um conjunto de objetos semelhantes, provendo todas as informações necessárias para a criação e utilização dos objetos. Uma instância, por sua vez, é um objeto descrito por uma classe particular. Cada objeto é instância de uma classe. Assim, no exemplo anterior `A,B` são instâncias da classe `Vector`.

Todas as instâncias de uma classe utilizam o mesmo método para responder a uma mensagem particular. A diferença na resposta obtida para duas instâncias distintas é resultado dos diferentes valores armazenados nas variáveis. Portanto, pode-se dizer que um

sistema orientado por objetos é desenvolvido a partir da criação das classes que descrevem os objetos constituintes do sistema.

Métodos: são procedimentos invocados pelo envio de mensagens para as instâncias de uma classe. Portanto, um método, como um procedimento, é a descrição de uma sequência de ações a serem executadas. De forma análoga aos procedimentos, os métodos devem conhecer os tipos de dados que manipulam.

Mecanismo de herança: permite compartilhar as informações entre objetos. Supondo uma hierarquia, os objetos situados em um nível inferior herdam todas as características (dados e operações) dos objetos situados em níveis superiores.

A maioria das linguagens orientadas por objetos implementam o mecanismo de herança entre as classes do sistema. Uma classe pode ser alterada para criar uma outra. Nesta relação, a primeira classe é denominada *superclasse* e a segunda *subclasse*. Uma subclasse pode adicionar novas variáveis e métodos, assim como redefinir os dados e operações da superclasse.

CLASSES PARA ELEMENTOS FINITOS

O objetivo, neste caso, foi implementar um conjunto de classes para análise linear de estruturas modeladas por elementos finitos isoparamétricos. Estas classes foram organizadas em 4 níveis como ilustrado na Figura 1. Observa-se que foram utilizados procedimentos do sistema ACDP[1] para o tratamento de erros, gravação e recuperação de dados em disco, assim como rotinas para manipulação de vetores e matrizes.

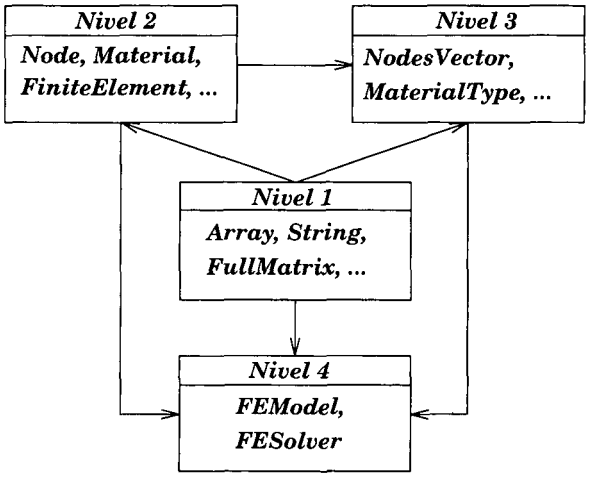

Figura 1. Níveis de organização das classes

NÍVEL 1

Envolve a definição das classes básicas do programa, compreendendo as estruturas de dados empregadas nos demais níveis. Métodos para alocação dinâmica de memória, manipulação de banco de dados e procedimentos matemáticos estão disponíveis.

Matrix: define uma matriz de elementos reais. Tem como principais variáveis os números de linhas e colunas, além de um vetor com os elementos da matriz. Implementa métodos para inicialização da matriz, busca de informação (máximo, mínimo, norma, ordem, elemento), bem como operações envolvendo matrizes, onde estão implementados algoritmos de transposição, inserção de elementos, adição, multiplicação e multiplicação por escalar.

Implementa, ainda, métodos diretos e iterativos para a solução de sistemas de equações, tais como Gauss e Gauss-Seidel.

SymmetricMatrix: classe análoga a **Matrix**, mas restrita a matrizes simétricas, armazenando, portanto, apenas a parte inferior da matriz. Considera ainda métodos diretos, iterativos estacionários e baseados em gradiente conjugado incluindo vários tipos de pré-condicionadores.

SymmetricSkyline: classe análoga a **Matrix** para matrizes simétricas do tipo skyline. É aplicada para definição da matriz global do sistema de equações. Armazena a ordem da matriz, altura das colunas e os elementos da parte inferior. Possui métodos para superposição de matrizes simétricas e resolução de sistemas de equações através de métodos diretos e iterativos.

SymmetricSparse: é semelhante a classe a **SymmetricSkyline**, considerando a estrutura comprimida por linhas para matriz esparsa[18].

Vector: caso particular de **Matrix** constituída de apenas uma coluna. Além dos procedimentos para adição, subtração e multiplicação, contém métodos para produto escalar, normas e multiplicação de matrizes por vetor.

String: classe responsável pelo manuseio de cadeias de caracteres ao longo do programa. Possui métodos de acesso a elementos, concatenação e comparação de caracteres.

Array: Tem por objetivo armazenar conjuntos de objetos. Na sua implementação, utilizou-se classes parametrizadas, onde o tipo de dado é também um parâmetro. Em C++, o comando `template` permite a definição de classes e funções parametrizadas. Pode-se definir um Array de nós e de strings pelas declarações:

```
Array<Node> Nodes(5) Array<String> Strings(10)
```

Foram implementados métodos para acesso e alteração do número de elementos, dentre outros. O objetivo desta classe é gerenciar conjuntos de qualquer tipo de dado ao longo do programa. Por questões de eficiência, consideraram-se as especializações `Array<long>` e `Array<double>` para o tratamento de conjuntos de números inteiros e reais, respectivamente.

NÍVEL 2

Neste nível, consideram-se as classes relativas ao modelo de elementos finitos.

Node: possui como principais atributos o número do nó e as coordenadas nodais. O número de coordenadas nodais depende da dimensão do problema em estudo, ou seja, uni, bi ou tridimemsional. Possui ainda uma variável para o número ótimo do nó obtido por um algoritmo de renumeração. Esta classe é empregada apenas para a passagem de parâmetros em alguns métodos.

DefinitionDOF: esta classe tem por objetivo armazenar os nomes dos graus de liberdade armazenados como uma variável do tipo `Array<String>`. Por exemplo, para um problema elástico bidimensional, os graus de liberdade dos nós são denominados UX e UY.

EliminatedDOF: esta classe constitui-se numa tabela de graus de liberdade a serem eliminados, correspondendo aos deslocamentos nulos, no processo de montagem do sistema de equações. Para isso, armazena o número dos nós e a cardinalidade dos graus de liberdade eliminados, utilizando variáveis do tipo `Array<long>`.

PrescribedBC: é nesta classe que são armazenadas as condições de contorno prescritas do problema a ser solucionado pelo modelo de elementos finitos. Como exemplo, podem-se citar as cargas concentradas, distribuídas e de corpo; gradientes de temperatura; e deslocamentos. Utilizam-se variáveis dos tipos `Array<double>` e `Array<long>` para armazenar os nós, os graus de liberdade e a intensidade dos carregamentos, deslocamentos e temperaturas.

DOFEquation: armazena para cada nó, o número ótimo determinado por um algoritmo de renumeração. Além disso, o número e a numeração dos graus de liberdade são armazenados nesta classe. Possui métodos para realizar a numeração dos graus de liberdade.

GeometricProperties: é uma tabela para armazenar as propriedades geométricas dos elementos, tais como espessura, momentos de inércia, área de secção, etc. Esta tabela é identificada pelo seu número armazenado com uma variável inteira.

ElasticMaterial: é uma classe genérica, definindo as características básicas de outras classes que implementam as informações referentes ao comportamento dos materiais elásticos. Possui uma variável para o número do material. Declara várias funções virtuais para inicialização e acesso às propriedades dos materiais e outras para obtenção das matrizes de elasticidade nos casos de estado plano de tensão, estado plano de deformação, sólidos axissimétricos e estado geral de solicitação. A partir desta classe, derivam-se outras duas: **ElasticIsotropicMaterial** e **ElasticOrtotropicMaterial**.

ElasticIsotropicMaterial: declara variáveis para o armazenamento de informações relativas aos materiais elásticos isotrópicos, como o módulo de elasticidade longitudinal, coeficiente de Poisson, coeficiente de expansão térmica e densidade. Implementa as funções declaradas virtuais na classe **ElasticMaterial**. A classe **ElasticOrtotropicMaterial** é análoga a esta classe, considerando no entanto, materiais ortotrópicos.

FiniteElement: constitui-se numa classe genérica de onde derivam-se cada um dos diferentes tipos de elementos finitos. Declara variáveis para número do elemento, número ótimo, número total de graus de liberdade, número do material, número da tabela de propriedades geométricas, número da tabela de sistemas locais de referência, número de pontos de integração e incidência.

Possui métodos para inicialização e acesso a estas informações, bem como métodos virtuais para o cálculo da matriz de rigidez, matriz de massa, tensões, deformações e erro em energia no elemento.

PlaneStressTriangular: é um dos tipos de elementos implementados. Através do mecanismo de herança, possui acesso a todos os atributos da classe **FiniteElement**. Implementa as operações declaradas virtuais em **FiniteElement** para o caso de estado plano de tensão.

Tem-se métodos de cálculo das matrizes de rigidez e de massa; tensores de tensão e deformação calculados nos pontos de intergração de Gauss-Legendere e nas coordenadas locais dos nós; estimador de erro; dentre outros.

Desenvolveram-se as classes **PlaneStrainTriangle** e **AxyssimetricTriangle**, semelhantes a esta classe, para os casos de estado plano de deformação e sólidos axissimétricos, respectivamente. Da mesma maneira, classes quadrados, cubos e tetraedros foram também implementadas.

TriangularShapeFunctions: implementa as funções de forma de Serendipty até o quarto grau para o caso de elementos finitos triangulares. Possui métodos para o cálculo das derivadas em relação às coordenadas locais e globais do elemento, assim como para a matriz e o determinante do Jacobiano.

TriangularGaussLegendre: armazena os pontos de integração e os coeficientes de ponderação para a integração de Gauss-Legendre utilizadas nas classes **PlaneStressTriangle**, **PlaneStrainTriangle** e **AxyssimetricTriangle**.

NÍVEL 3

Neste caso, consideram-se conjuntos das classes descritas no nível anterior. Constitui-se basicamente no grupo de classes que vai organizar as informações dos atributos do modelo de elementos finitos, permitindo a inicialização, modificação e acesso aos atributos das classes utilizadas. Os dados são lidos a partir de arquivos de dados no formato definido[2].

Nodes: possui como atributos principais uma variável para armazenar a dimensão do problema (uni, bi ou tridimensional) e um `Array<double>` para as coordenadas.

DOFBoundaryConditions: armazena as tabelas de definição dos nomes dos graus de liberdade, graus eliminados e prescritos. Trata as informações das classes **DefinitionDOF**, **EliminatedDOF** e **PrescribedBC**.

MaterialGroup: implementa um vetor de apontadores da classe **ElasticMaterial**, armazenando ainda o tipo dos materiais (isotrópico ou ortotrópico) aplicados na análise.

FiniteElementGroup: um grupo de elementos finitos é constituído por elementos do mesmo tipo. Esta classe declara um vetor do tipo **FiniteElement** e o nome do elemento, a fim de gerenciar os vários tipos de elementos da malha. Além disso, tem-se uma variável `Array<long>` para a incidência nodal dos elementos do grupo.

LoadCase: é um vetor da classe **PrescribedBC** para o tratamento das condições de carregamento aplicadas à estrutura.

NÍVEL 4

Neste nível, tem-se classes para o armazenamento de todos os atributos do modelo de elementos finitos para que o problema seja posteriormente resolvido.

FEModel: esta classe armazena todos os atributos do modelo de elementos finitos. Define variáveis dos tipos descritos no Nível 3 para os nós, materiais, condições de contorno, grupos de elementos e condições de carregamento. Tem-se ainda uma variável da classe **String** para armazenar o título do modelo. Assim, a partir de um arquivo de entrada, especificando as características do modelo e das operações implementadas nesta classe, inicializam-se todas as demais já apresentadas.

FESolver: é uma classe derivada de FEModel, implementando operações para a resolução do modelo. Possui variáveis para a matriz do sistema nos formatos skyline e esparso, sendo opção do usuário escolher a opção mais conveniente. Como métodos de solução, tem-se algoritmos diretos, iterativos e multgrid[18,19,20].

AMBIENTES PARA ANÁLISE DE PROBLEMAS ELÁSTICOS BIDIMENSIONAIS

A aplicação do MEF para a resolução de problemas práticos de engenharia tem apresentado um crescimento considerável. Estas ferramentas computacionais possibilitam otimizar projetos na sua fase inicial, assim como verificar o comportamento de um componente já existente, visando validar a concepção atual e permitir ainda a sua posterior otimização.

Vários programas comerciais estão disponíveis para esta finalidade. Sob o ponto de vista do usuário destes pacotes, exige-se um bom conhecimento da técnica de análise, assim como uma boa experiência na utilização do programa, visto que em geral a interface de comandos é um tanto complexa. Tais fatos limitam uma maior disseminação destes programas, pois o usuário deve possuir uma base sólida de conhecimentos para o efetivo emprego destes recursos a problemas de engenharia. Uma hipótese mal formulada, implica em resultados não refletindo o real comportamento mecânico do componente. Desta maneira, o programa deve ser de fácil utilização, permitindo ao usuário dedicar a maior parte do tempo na definição e simulação de modelos para a estrutura considerada.

Observa-se, em geral, que as tarefas de definição da geometria do componente e da respectiva malha de elementos finitos são responsáveis pela maior demanda em termos de tempo no estudo de um problema. Em relação aos pacotes existentes, observa-se uma maior integração entre programas de CAD (*Computer Aided Design*) e de análise. Empregam-se as ferramentas CAD para a definição da geometria e possivelmente da malha. A partir daí,

utiliza-se o programa de análise para a obtenção dos resultados. No entanto, em vários casos não se verifica uma efetiva integração ou onde esta ocorre tem-se uma interface complexa dificultando a tarefa do usuário.

Desta forma, algoritmos robustos e eficientes para a geração da geometria do componente e da respectiva malha de elementos, solução, estimação de erros e refinamento, acessíveis através de uma interface de comandos simples, são pré-requisitos fundamentais para a análise numérica por elementos finitos.

Dentro deste contexto, desenvolveu-se inicialmente o ambiente SAFE[2] para análise bidimensional de problemas elásticos planos. A Figura 2 ilustra a janela principal e alguns dos módulos do programa, tais como o editor para a especificação dos parâmetros da estrutura e da análise , o gerador de malhas, a visualização de resultados, além da malha refinada através do procedimento adaptável baseado no estimador Zhu-Zienkiewicz[21].

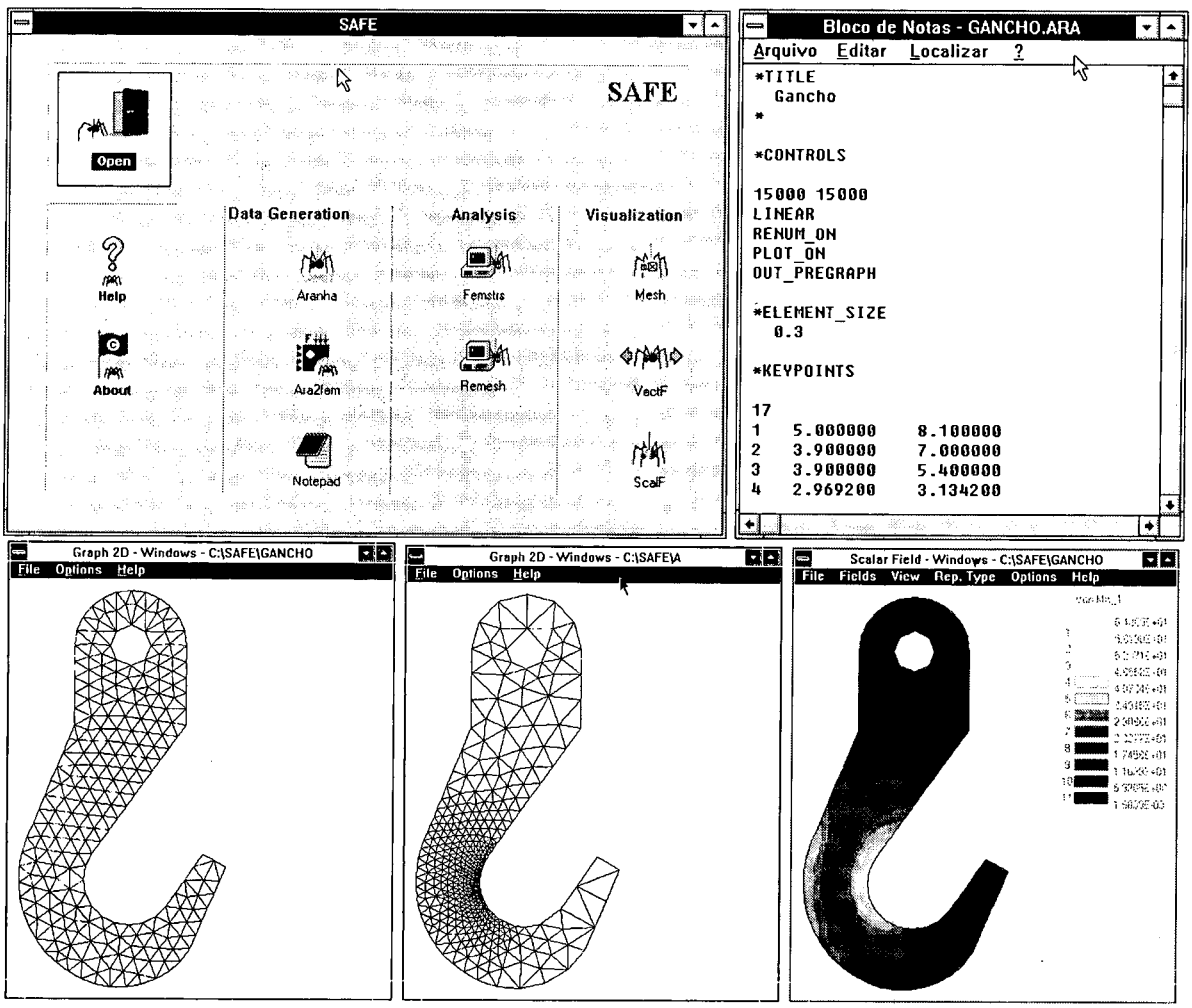

Figura 2. Módulos do programa SAFE

A especificação dos parâmetros é feita através de 2 arquivos de dados, com extensões **.ara** e **.a2f**, contendo várias palavras chaves[2]. O primeiro especifica o contorno do domínio, parâmetros de controle da malha e erro admissível. O outro arquivo considera as propriedades dos materiais, tipos de elementos, carregamentos, restrições, graus de liberdade e propriedades geométricas. Com o arquivo **.ara**, efetua-se a geração da malha através do programa ARANHA[8]. A partir daí, aplicam-se os parâmetros disponíveis no arquivo **.a2f**,

gerando como saída um outro arquivo com extensão **.fem** a ser submetido ao módulo de solução. Estas tarefas constituem a parte de geração dos dados do programa.

Na parte de análise, resolve-se o modelo de elementos finitos, determinando deslocamentos e tensões. Tem-se ainda o módulo para efetuar o procedimento adaptável gerando um novo conjunto de arquivos de entrada **.ara** e **.a2f**. Finalmente, pode-se efetuar a visualização da malha e de campos vetoriais e escalares. Observa-se que a comunicação entre os vários módulos é realizada através de arquivos e bancos de dados.

A partir da experiência do programa SAFE, desenvolveu-se um segundo ambiente, denominado SAT, onde todos os dados do modelo são especificados de forma interativa. A Figura 3 ilustra a janela principal e as interfaces dos módulos. Inicialmente, fornece-se o nome do projeto, definindo o nome dos bancos de dados a serem empregados nas demais partes do programa. Alguns argumentos genéricos do projeto, tais como título, data da última modificação, autor e observações podem ser colocadas neste arquivo de projeto.

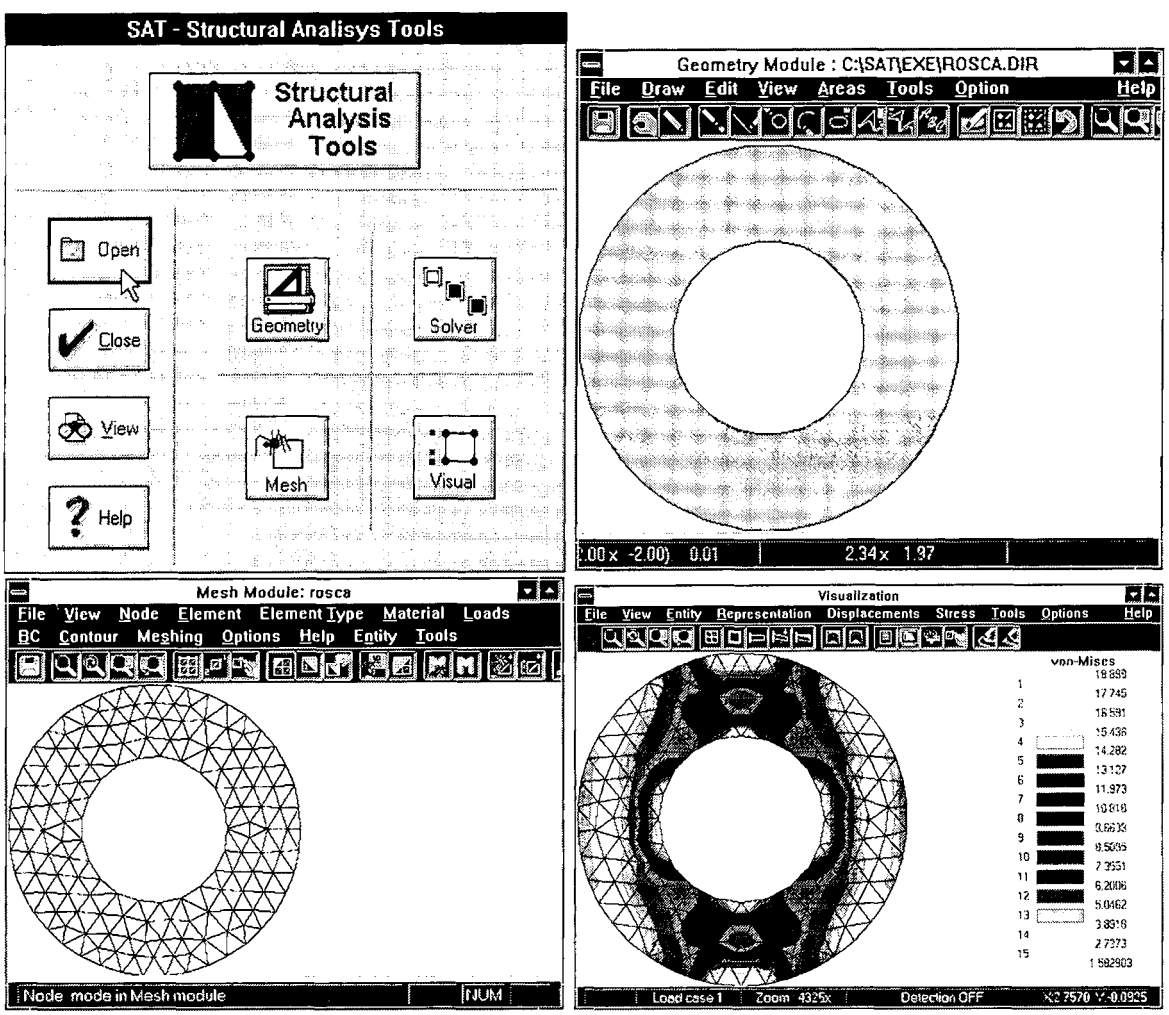

Figura 3. Módulos do programa SAT

O módulo GEOMETRY permite a definição do contorno da geometria do componente a ser analisado. Está baseado no conceito de NURBS (*Non-uniform Rational B-Splines*)[16], pondendo-se ainda importar arquivos no formato DXF. Possui como primitivas principais linha, arco, círculo, curva, dentre outras. Várias ferramentas de edição estão disponíveis

tais como rotação, translação, *zoom* e divisão. O objetivo principal é definir as áreas sobre as quais serão gerados os elementos, gravando-se ao final o arquivo com extensão **.ara**.

No caso do módulo MESH[4], partindo-se das áreas armazenadas no banco de dados do GEOMETRY, agregam-se informações de controle da malha, como por exemplo o tamanho médio dos elementos. A partir daí, executa-se o programa ARANHA[8] para a geração da malha. Todos os atributos do modelo de elementos finitos tais como carregamentos, propriedades dos materias e condições de contorno podem ser especificados de forma interativa através de diálogos, gravando-se ao final o arquivo **.a2f**. Com os arquivos **.ara** e **.a2f**, gera-se o arquivo **.fem** para ser submetido ao módulo SOLVER, o qual é o mesmo do programa SAFE, não tendo sido considerado ainda o estimador de erro.

Ao final, os resultados escalares e vetoriais podem ser apresentados de formas numérica e gráfica no módulo VISUALIZATION[26]. De forma análoga ao SAFE, tem-se mapas de cores em faixas e curvas de níveis, geometria original e/ou deformada, agregando ainda um procedimento de animação da configuração deformada. Um aspecto importante foi a organização dos campos escalares e vetoriais, onde os nomes e a quantidade dos mesmos são gravados no banco de dados do SOLVER. Assim, o programa determina dinamicamente os campos a serem apresentados, permitindo a sua aplicação para a visualização de resultados de outros tipos de análise.

Cada um dos módulos constitue-se num programa independente estando a comunicação entre os mesmos efetuada através de bancos de dados. A representação das entidades gráficas em todos os módulos é feita por uma mesma estrutura de dados baseada em NURBS. Isto permite uniformizar todas as operações efetuadas tais como rotação, translação, *zoom*, *fill*, dentre outras.

Os resultados obtidos para os dois programas foram bastante satisfatórios. Apesar de tratarem somente problemas elásticos bidimensionais, os programas apresentam ferramentas efetivamente integradas, desde a parte de geração de contornos e de malha, visualização dos resultados, estimação de erros e refinamento adaptável. As interfaces gráficas permitem um acesso simples e intuitivo aos vários módulos disponíveis.

Outros tipos de problemas podem ser facilmente integrados na estrutura já existente, criando-se por exemplo novos módulos de análise. Como exemplo deste procedimento, o módulo OPTIMIZATION para a otimização estrutural de componentes têm sido desenvolvido. Os procedimentos numéricos foram implementados usando as classes para elementos finitos discutidas neste artigo. Estão baseados num algoritmo de minimização de ponto interior e na fomulação contínua de análise de sensibilidade, tomando-se a espessura e a forma como variáveis de projeto[7]. A Figura 4 ilustra a otimização de forma de um componente bidimensional.

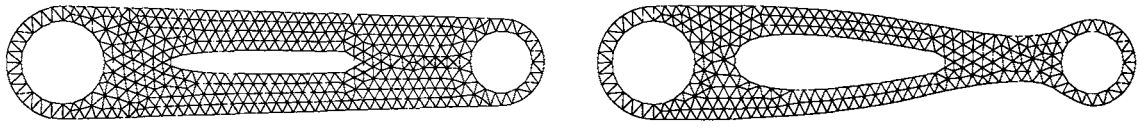

Figura 4. Exemplo de otimização de forma

COMENTÁRIOS FINAIS

A aplicação do modelo orientado por objetos tem permitido o desenvolvimento de programas com qualidade, eficiência, portabilidade e produtividade. Um aspecto importante, ao se aplicar este modelo, é realizar o projeto da hierarquia de classes, antes da fase de implementação. Esta é a tarefa mais complexa e deve ser efetuada com cuidado, procurando obter um conjunto de classes de fácil reutilização, extensão e manutenção.

Os ambientes de análise têm sido estendidos para o tratamento de problemas tridimensionais. Espera-se alcançar as mesmas facilidades de utilização e níveis de eficiência. No caso do modelamento geométrico no ambiente SAT tridimensional, tem-se usado a biblioteca ACIS. Trata-se de um poderoso conjunto de classes em C++ para o modelamento e visualização de objetos. Tem-se tornado um padrão na área de computação gráfica. Softwares comerciais, tais como Autocada e SolidEdge, têm sido desenvolvidos empregando a plataforma ACIS.

AGRADECIMENTOS

Os autores agradecem aos seguintes órgãos pelo apoio ao desenvolvimento do trabalho: CNPq (Proc. 523.982/94-1), CNPq-RHAE (Proc.610.035/94-0), FAPESP (97/97/0540-0), LNCC e UNICAMP. Os autores também agradecem as facilidades de software gentilmente fornecidas pelo Grupo TACSOM (www.lncc.br/~ tacsom).

REFERÊNCIAS

1 A.C.S. Guimarães y R.A. Feijóo, "The ACDP system (In portuguese)", 27/89, National Laboratory for Scientific Computation, Rio de Janeiro, Brazil, (1989).

2 A.C.S. Guimarães, E.A. Fancello, G.R. Feijóo y R.A. Feijóo, "SAFE - integrated system for structural finite element analysis - Version 1.0", National Laboratory for Scientific Computation, Rio de Janeiro, Brazil, (1994).

3 A. Cardona, I. Klapka y M. Geradin, "Design of a new finite element programming environment", *Engineering Computations*, Vol. **11**, pp. 365-381, (1994).

4 A.M. Thees, "Computational tools for a finite element structural analysis software (In portuguese), DPM/FEM, State University of Campinas, Brazil, Research Report, (1996).

5 B. Raphael y C.S. Krishnamoorthy, "Automating finite element development using object-oriented techniques", *Engineering Computation*, Vol. **10**, 1, pp. 267-278, (1993).

6 B.W.R. Forde, R.B. Foschi y S.F. Stiemer, "Object-oriented finite element analysis *Computer & Structures*, Vol. **34**, pp. 355 374, (1990).

7 C.A.C. Silva, "Object-oriented structural optimization and sensitivity analysis" (In portuguese), DPM/FEM, State University of Campinas, Brazil, (1997).

8 E.A. Fancello, A.C.S. Guimarães, R.A. Feijóo y M. Venere, "Automatic two-dimensional mesh generation using object-oriented programming" (In portuguese), Brasilian Association of Mechanical Sciences (Ed.), *Proceedings of 11th Brazilian Congress of Mechanical Engineering*, São Paulo, pp. 635 638, December, (1991).

9 E. Dari, "Contribuciones a la triangulación automática de dominios tridimensionales", Instituto Balseiro, Bariloche, Argentina, (1994).

10 R.A. Feijóo, "Relatórios do sistema SDP formado por 15 manuais" National Laboratory for Scientific Computation, Research Reports, Rio de Janeiro, Brazil, (1987).

11 G.R. Miller, "An Object-oriented approach to structural analysis and design", *Computer & Structures*, Vol. **40**, pp. 75-82, (1991).

12 G.W. Zeglinski, R.P.S. Han y P. Aitchison, "Object-oriented matrix classes for use in a finite element code using C++", *Int. J. for Num. Meth. in Engng.*, Vol. **37**, pp. 3921-3937, (1994).

13 G. Yu y H. Adeli, "Object-oriented finite element analysis using Eer model", *J. of Struct. Engng*, Vol. **119**, 9, pp. 2763-2781, (1993).

14 H. Adeli y G. Yu, "An integrated computing environment for solution of complex engineering problems using the object-oriented programming paradigm and a blackboard architeture", *Computer & Structures*, Vol. **54**, 2, pp. 255-265, (1995).

15 J.S.R.A. Filho y P. Devloo, "Object-oriented programming in scientific computations: the beginning of a new era, *Engineering Computations*, Vol. **8**, pp. 81–88, (1991).

16 M.C. Galvão, "A NURBS based environment for domain definition applied to mesh generation" (In portuguese), Research Report, DPM/FEM, State University of Campinas, Brazil, (1995).

17 M.L. Bittencourt, "Static and dynamic analysis by substructuring and object-oriented programming" (In portuguese), DPM/FEM, State University of Campinas, Brazil, (1990).

18 M.L. Bittencourt, "Adaptive iterative and multigrid methods applied to non-structured meshes" (In portuguese), DPM/FEM, State University of Campinas, Brazil, (1996).

19 M.L. Bittencourt y R.A. Feijóo, "Análise comparativa de métodos diretos e iterativos para a solução de sistema de equações", *Revista Int. de Mét. Num.para Cálculo y Diseño en Ing.*, Vol. **13**, 2, pp. 123-148, (1997).

20 M.L. Bittencourt y R.A. Feijóo, "Métodos multigrid em malhas não-aninhadas aplicados a problemas elásticos", *Revista Int. de Mét. Num. ypara Cálculo y Diseño en Ing.*, Vol. **14**, 1, pp. 3-23, (1998).

21 O.C. Zienkiewicz y J.Z. Zhu, "A simple error estimator and adaptative procedure for practical engineering analysis", *Int. J. for Num. Meth. in Engng*, Vol. **24**, pp. 337–357, (1987).

22 P.H. Menétrey y T. Zimmermann, "Object-oriented non-linear finite element analysis: Application to J2 plasticity", *Computer & Structures*, Vol. **49**, 5, pp. 767-777, (1993).

23 R.A. Feijóo, A.C.S. Guimarães y E.A. Fancello, "Algumas experiencias en la programación orientada por objetos y su aplicación en el método de los elementos finitos", Research Report 15/91, National Laboratory for Scientific Computation, Rio de Janeiro, Brazil, (1991).

24 R.S. Pressman, "*Software engineering - a practitioner's approach*", McGraw-Hill, New York, (1987).

25 S.B. Lippman, "*C++ primer*", Addison-Wesley, Reading, (1991).

26 S.H.P. Ramos, "Visualization of structural analysis results in windows environment" (In portuguese), Research Report, DPM/FEM, State University of Campinas, Brazil, (1996).

27 S.P. Scholz, "Elements of an object-oriented fem++ program in C++", *Computer & Structures*, Vol. **43**, 3, pp. 517–529, (1992).

28 T.J. Ross, L.R. Wagner y G.F. Luger, "Object-oriented programming for scientific codes: II. Examples in C++", *J. of Comp. in Civil Engng.*, Vol. **6**, 4, pp. 497–514, (1992).

29 T. Zimmermann, Y. Dubois-Pèlerin y P. Bomme, "Object-oriented finite element programming: I. Governing principles", *Comp. Meth. in Appl. Mech. and Engng.*, Vol. **98**, pp. 291-303, (1992).

30 W.W. Tworzydlo y J.T. Oden, "Towards an automated environment in computational mechanics", *Comp. Meth. in Appl. Mech. and Engng.*, Vol. **104**, pp. 87-143, (1993).

31 W.W. Tworzydlo y J.T. Oden, "Knowledge-based methods and smart algorithms in computational mechanics", *Engineering Fracture Mechanics*, Vol. **50**, 5, pp. 759-800, (1995).

32 Y. Dubois-Pèlerin, T. Zimmermann y P. Bomme, "Object-oriented finite element programming: II. A prototype program in smalltalk", *Comp. Meth. in Appl. Mech. and Engng*, Vol. **98**, pp. 361-397, (1992).

33 Y. Dubois-Pèlerin, T. Zimmermann y P. Bomme, "Object-oriented finite element programming: III. An efficient implementation in C++", *Comp. Meth. in Appl. Mech. and Engng*, Vol. **108**, pp. 165-183, (1993).