VI SIMPÓSIO MINEIRO DE MECÂNICA COMPUTACIONAL
UNIVERSIDADE FEDERAL DE ITAJUBÁ
INSTITUTO DE ENGENHARIA MECÂNICA
Itajubá, 17 a 19 de Maio de 2004

IMPLEMENTAÇÃO DE MODELOS ESTRUTURAIS DE BARRAS COMO CASOS PARTICULARES DO MÉTODO DE ELEMENTOS FINITOS

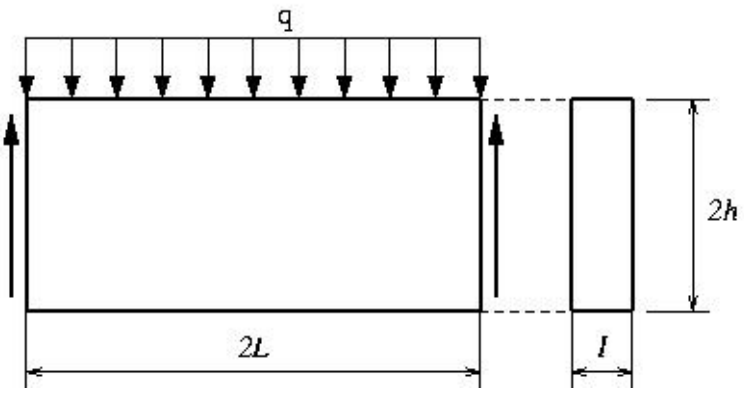
Roque Luiz Pitangueira[1]
Alcebíades de Vasconcellos Filho[2]
Flávio Torres da Fonseca[3]
Universidade Federal de Minas Gerais
Departamento de Engenharia de Estruturas
30110060 - Belo Horizonte - Minas Gerais
[1]roque@dees.ufmg.br
[2]avf@dees.ufmg.br
[3]flaviotf@dees.ufmg.br

Resumo: *O artigo apresenta parte do desenvolvimento de um sistema computacional que visa fomentar a pesquisa de modelos discretos de análise estrutural. Refere-se à parte do núcleo numérico do sistema relativa a modelos estruturais de barras. A concepção do sistema, como um conjunto de segmentos de aplicação, é apresentada, destacando-se as vantagens deste conceito. A escolha do paradigma de programação orientada a objetos e da linguagem de programação Java, como recursos de implementação, é discutida. Nesta discussão apontam-se as principais características das tecnologias escolhidas no que se refere às plataformas de desenvolvimento, execução e distribuição, bem como à segmentação, expansão e manutenção do código. A persistência dos dados compartilhados entre os vários segmentos do sistema também é discutida, ressaltando-se o uso de arquivos XML para criação dos protocolos de comunicação. A formulação dos modelos estruturais de barras, como casos particulares do Método de Elementos Finitos, é brevemente revisada. O projeto orientado a objetos para implementação da referida formulação é apresentado, indicando-se o segmento do núcleo numérico do sistema que a contém. Discute-se, então, a utilização desse recurso no ensino do Método de Elementos Finitos nos cursos de graduação em engenharia.*

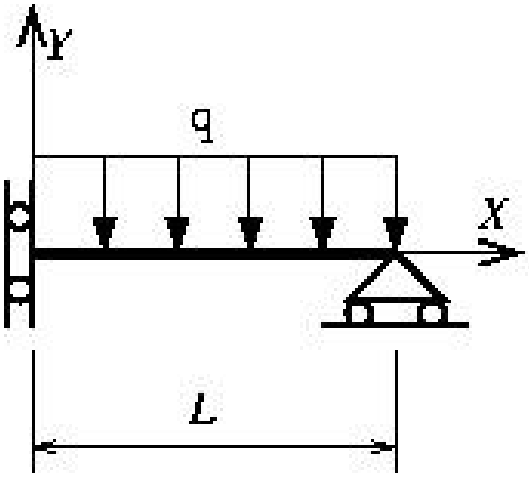
Palavas chave: *Modelos Estruturais de Barras, Método de Elementos Finitos, Programação Orientada a Objetos.*

1. INTRODUÇÃO

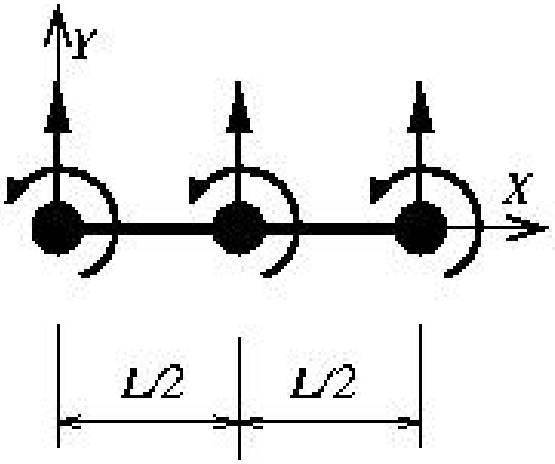
Conforme Soriano e Lima (1999), costuma-se dividir o processo de análise estrutural em três etapas (Figura 1). Orientando-se por experiência de projeto, o problema de meio contínuo da estrutura real (Figura 1(a)) é substituído por um modelo matemático utilizando-se hipóteses simplificadoras (Figura 1(b)). Tal modelo matemático é expresso por equações diferenciais (ordinárias ou parciais) cujas soluções, ditas soluções analíticas, são conhecidas apenas em alguns poucos casos simples. Para superar as limitações de tais soluções, adota-se um modelo numérico aproximado, dito modelo discreto (Figura 1(c)). Nos modelos discretos, as equações são algébricas e as grandezas são determinadas em um número finito de pontos, diferentemente das soluções analíticas cujas equações diferenciais, quando resolvidas, permitem avaliar as grandezas em um número infinito de pontos. Dentre os métodos discretos mais utilizados destacam-se o método de elementos finitos (MEF) e o método de elementos de contorno (MEC).



(a) Problema de Meio Contínuo



(b) Modelo Matemático



(c) Modelo Discreto

Figura 1: Modelamento de uma viga

A pesquisa na área de métodos numéricos e computacionais para os referidos modelos discretos procura um aprimoramento das hipóteses simplificadoras dos mesmos. Isto é feito de forma a ampliar complexidades a partir dos conceitos já consolidados. Entretanto,

observa-se sempre um recomeço do processo ao se recriarem as ferramentas relativas às tecnologias dominadas. Um exemplo ilustrativo deste fato é a (re)implementação computacional de algoritmos de solução de sistemas de equações algébricas lineares, toda vez que os mesmos são usados como parte do processo de aprimoramento de determinado modelo discreto.

Ao longo do tempo, algumas iniciativas de desenvolvimento de software pela comunidade acadêmica resultaram em produtos dependentes de sistema operacional, pouco amigáveis, escritos em linguagens de programação não apropriadas, de expansão, manutenção e distribuição difíceis, desenvolvidos por equipes fechadas, com documentação deficiente, entre outras limitações. Tais fracassos podem ser creditados à falta de disposição da comunidade em se apropriar das tecnologias emergentes ou mesmo à inexistência das mesmas.

Esta constatação confronta-se com o surgimento e aprimoramento de soluções tecnológicas para desenvolvimento de software, como programação orientada a objetos, linguagem Java, XML (eXtensible Markup Language), padrões de projeto de software (Gamma et al. 1995), entre outras. Soluções estas que permitem o desenvolvimento de sistemas computacionais segmentados, amigáveis a mudanças e escaláveis em complexidade (Alvim, 2003).

Portanto, o desafio de desenvolver sistemas utilizando estes recursos é condição obrigatória para aprimoramento da agilidade e criatividade da pesquisa na área.

Este artigo apresenta parte do desenvolvimento de um sistema computacional cujo principal objetivo é fomentar a pesquisa de modelos discretos de análise estrutural.

2. UM SISTEMA SEGMENTADO

A utilização de modelos discretos de análise estrutural compreende três etapas principais inter-relacionadas: (1)criação do modelo, (2)montagem e resolução do modelo e (3)avaliação de resultados. Na criação do modelo, o analista informa as hipóteses simplificadoras relativas à geometria, material, carregamento e condições de contorno e estas são representadas com entidades matemáticas apropriadas, gerando assim o que se denomina malha e os atributos do modelo. Na etapa de montagem e resolução do modelo, combinam-se as informações matematicamente representadas, de modo a produzir equações algébricas lineares que, quando solucionadas, permitem obter as diversas grandezas. Na avaliação de resultados, o analista faz uma análise crítica e verifica a adequação dos mesmos ao problema em estudo.

Para disponibilização deste processo em computadores, normalmentente a referida divisão é adotada através de programas de pré-processamento (para a criação dos modelos com recursos gráficos interativos), processamento (para a montagem e resolução numérica do modelo) e pós-processamento (para visualização gráfica de resultados).

As possibilidades que os recursos tecnológicos para desenvolvimento de software oferecem para cada uma das três etapas constituem amplo campo de pesquisa na área de métodos numéricos e computacionais aplicados à engenharia.

O domínio destes recursos e a aplicação dos mesmos no aprimoramento progressivo dos modelos, sem ter que recomeçar o processo a cada novo aperfeicoamento, requer um ambiente computacional segmentado, amigável a mudanças e escalável em complexidade.

O ambiente computacional que aqui se apresenta pretende ter estas características, conforme mostra o projeto preliminar da Figura 2. Como pode ser visto na figura, o sistema é constituído de três aplicações denominadas `JMESH`, `NUCLEUS` e `JPOST`. `JMESH` e `JPOST` são aplicações gráficas interativas, implementadas na linguagem Java (Rowe, 2001), que disponibilizará, respectivamente, ferramentas de pré e pós-processamento de diferentes

modelos discretos. NUCLEUS é uma aplicação, também implementada em Java (Flanagan, 2000) que representa o núcleo numérico do sistema. Este núcleo é responsável pela obtenção dos resultados de diferentes modelos discretos de análise estrutural. A persistência dos dados compartilhados pelas três aplicações é alcançada através de uma interface baseada em arquivo(s) XML.

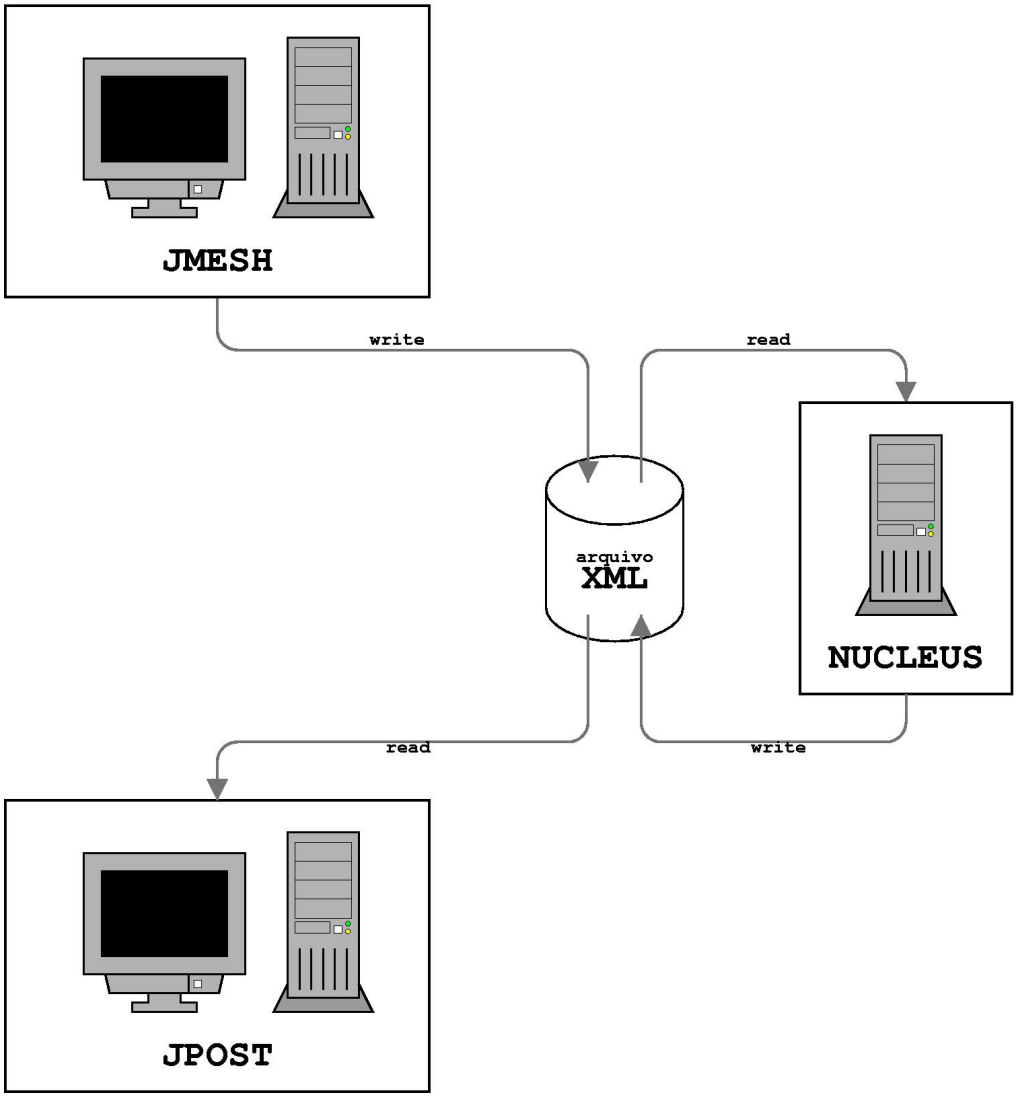


Figura 2: Projeto Preliminar do Sistema

Cada uma destas aplicações é implementada segundo o paradigma de programação orientada a objetos (POO). A programação orientada a objetos (POO) é uma técnica de programação baseada em classes e objetos. Os dados e métodos são encapsulados nos objetos, sendo eles intimamente amarrados entre si. Os objetos apresentam a propriedade de ocultar informações. Ou seja, apesar de eles se comunicarem uns com os outros através de interfaces bem definidas, geralmente um objeto não tem permissão para conhecer como outros objetos são implementados. Isto permite que os programas possam ser divididos em módulos independentes, permitindo o trabalho em conjunto de diversas pessoas em diferentes locais e épocas. Desta forma, a manutenção e expansão do código são muito mais fáceis de serem feitas em um programa desenvolvido com POO do que em um feito com linguagem estruturada. A linguagem de programação Java é uma linguagem orientada a objetos, apresentando todas as vantagens deste tipo de programação. Além disso, ela foi

desenvolvida de forma a ser independente de plataforma. Essa característica é denominada portabilidade e é um dos principais atrativos desta linguagem. Um programa desenvolvido em Java pode ser compilado em um sistema operacional e executado em outro, sem prejuízos. Devido a todas as vantagens citadas acima, escolheu-se utilizar a linguagem Java, que suporta o paradigma da programação orientada a objetos, para a implementação do sistema.

A aplicação `NUCLEUS` utiliza os diversos conceitos do paradigma de POO (classes, herança, polimorfismo, etc.) de modo a possuir a segmentação necessária ao aprimoramento progressivo do sistema. As aplicações gráficas interativas (`JMESH` e `JPOST`), também implementadas segundo POO, utilizam o padrão de projeto de software (Gamma et al. 1995) denominado `Model-View-Controller` (MVC). Este padrão é bastante apropriado uma vez que preconiza a separação do processamento da informação de sua representação gráfica (Pietro, 2001), facilitando assim os trabalhos de expansão e manutenção destas aplicações.

A persistência de dados entre os diversos segmentos da aplicação é feita através de arquivos XML, sigla esta que é uma abreviação de "Extensible Markup Language", ou seja, linguagem de marcação estendida. A XML é uma técnica para criar dados estruturados baseados em um arquivo texto. O que a difere de outras linguagens de marcação, como a HTML, é o fato de as regras de marcação serem definidas pelo programador, podendo ele fazê-las do modo que melhor o atenda. A "eXtensible Markup Language" (XML) está sendo adotada como padrão para troca de documentos através da internet. Com a tecnologia dos "WEB Services", praticamente qualquer software ou componente de software (orientado a objetos) pode ser utilizado remotamente, sendo necessário somente que as partes "conversem" em XML. A tecnologia SOAP ("Simple Object Access Protocol"), por exemplo, opera sobre a WEB de maneira que suas mensagens (requisições e respostas) sejam simplesmente documentos XML (Braz, 2003). Assim, a opção de fazer a persistência dos dados em arquivos XML (ver figura 2) e a segmentação propiciada pela utilização de POO permitirá que o sistema ou partes deste possa, futuramente, ser utilizado através da internet.

3. MODELOS DE BARRAS E ABSTRAÇÕES DE OBJETOS

Uma das tarefas mais difíceis do projeto de software orientado a objetos é estabelecer qual a decomposição do problema em sub-problemas mais adequada. Também não é nada fácil decidir quais categorias ou classes de objetos compõem o problema a ser resolvido.

Para modelos discretos de análise estrutural, esta tarefa pode ficar menos árdua se a formulação matemática dos mesmos for observada com cuidado. No caso do Método dos Elementos Finitos, formulado em termos cinemáticos, os deslocamentos em qualquer ponto (armazenados no vetor $\underline{u}$) são obtidos a partir dos valores nodais (armazenados no vetor $\underline{d}$), através da aproximação

$$\underline{u} = \underline{N}\ \underline{d} \tag{1}$$

onde $\underline{N}$ é a matriz que contém as chamadas funções de forma do elemento, uma para cada deslocamento permitido.

Utlizando a aproximação dada na equação 1, as componentes de deformação (vetor $\underline{\epsilon}$) podem ser calculadas através da relação

$$\underline{\epsilon} = \underline{B}\ \underline{d} \tag{2}$$

onde $\underline{B}$ é uma matriz de derivadas das funções de forma.

Adotando-se hipóteses relativas ao comportamento do material, as tensões (vetor $\underline{\sigma}$) podem ser calculadas por

$$\underline{\sigma} = \underline{\mathrm{C}}\ \underline{\epsilon} \tag{3}$$

onde $\underline{\mathrm{C}}$ é uma matriz contendo combinações de propriedades do material.

Combinando-se as relações acima (equações 1, 2 e 3) e informações relativas ao carregamento, obtém-se a equação matricial de equilíbrio de cada elemento, dada por

$$\underline{\mathrm{k}}\ \underline{\mathrm{d}} = \underline{\mathrm{p}} + \underline{\mathrm{f}} \tag{4}$$

onde

$$\underline{\mathrm{k}} = \int_V \underline{\mathrm{B}}^T\ \underline{\mathrm{C}}\ \underline{\mathrm{B}}\ dV \tag{5}$$

é a matriz de rigidez do elemento,

$$\underline{\mathrm{f}} = \int_V \underline{\mathrm{N}}^T\ \underline{\mathrm{b}}\ dV \tag{6}$$

é o vetor que contém o carregamento nodal equivalente a cargas não nodais (vetor $\underline{\mathrm{b}}$) e $\underline{\mathrm{p}}$ é o vetor de cargas nodais.

A combinação das rigidezes e carregamentos dos vários elementos leva à equação de equilíbrio do modelo

$$\underline{\mathrm{K}}\ \underline{\mathrm{d}} = \underline{\mathrm{P}} + \underline{\mathrm{F}} \tag{7}$$

Para modelos estruturais de barras, tratados como casos particulares do Método dos Elementos Finitos, as equações 5 e 6 ficam simplificadas nas formas

$$\underline{\mathrm{k}} = \int_L M\ S\ \underline{\mathrm{B}}^T\ \underline{\mathrm{B}}\ dx \tag{8}$$

e

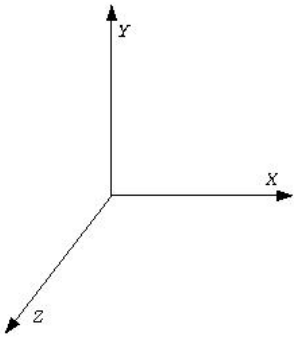
$$\underline{\mathrm{f}} = \int_L \underline{\mathrm{N}}^T\ \underline{\mathrm{b}}(x)\ dx \tag{9}$$

onde M é um escalar, propriedade do material, S é um escalar, propriedade geométrica da seção transversal e as integrais são realizadas ao longo do comprimento do elemento (L).

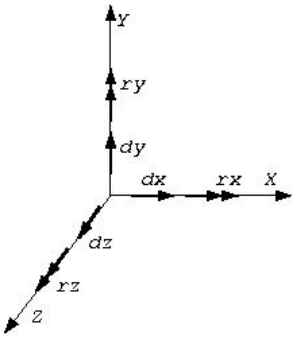
A partir da observação da formulação acima, é razoável supor, para o caso de modelos de barras, as categorias de objetos mostradas na figura 3. Um nó (figura 3(d)) é caracterizado por suas coordenadas (figura 3(a)), seus deslocamentos (figura 3(b)) e suas cargas (figura 3(c)). Uma barra (figura 3(h)), genericamente tridimensional, é composta por nós (figura 3(d)), material (figura 3(f)), seção transversal (figura 3(e)) e pode estar submetida a forças e momentos distribuídos em seu corpo, como ilustrado na figura 3(g). Quando parte de um modelo estrutural, cada barra tem seu estado deformado descrito por uma função de aproximação, como mostra a figura 3(i).

Estas categorias ou classes de objetos devem conter informações suficientemente gerais para caracterizá-los. Entretanto, é comum a particularização dos modelos estruturais de barras nas categorias: Treliça Plana, Viga, Pórtico Plano, Grelha, Treliça Espacial e Pórtico Espacial. A figura 4 mostra esta particularização, destacando as características de cada um dos modelos relacionadas ao material, seção transversal, carregamento e deslocamentos nodais.

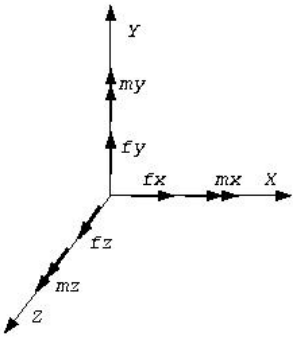
A seção a seguir detalha o projeto orientado a objetos para modelos estruturais de barras, inspirado nas abstrações de classes aqui discutidas.



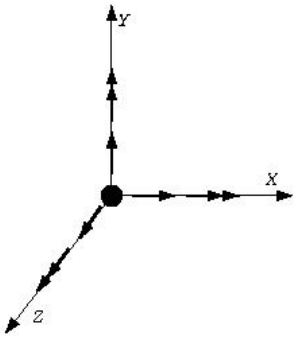
(a) Coordenadas



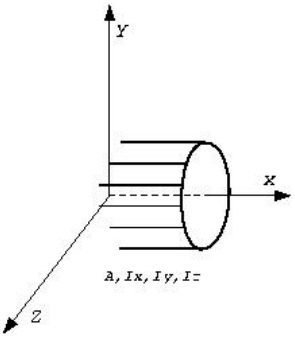
(b) Deslocamentos



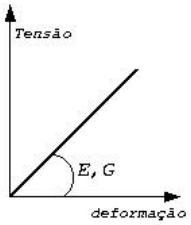
(c) Cargas



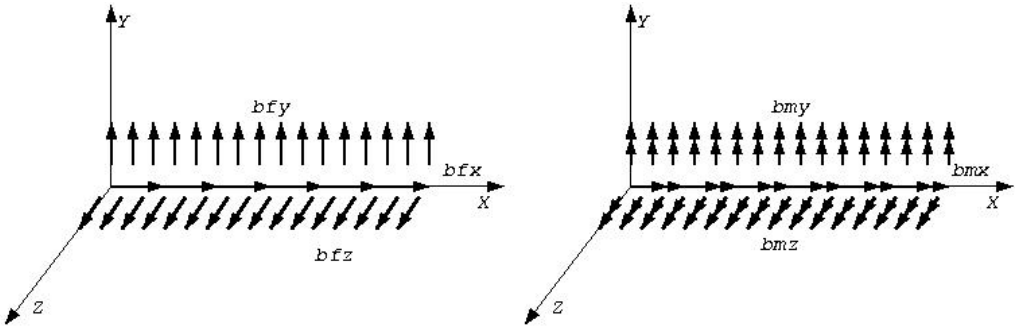
(d) Nó



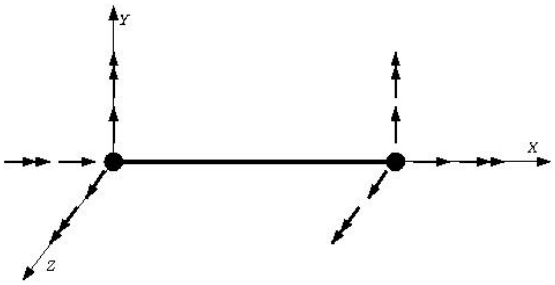
(e) Seção Transversal



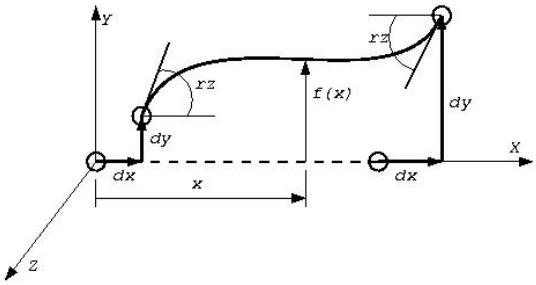
(f) Material



(g) Cargas em uma Barra

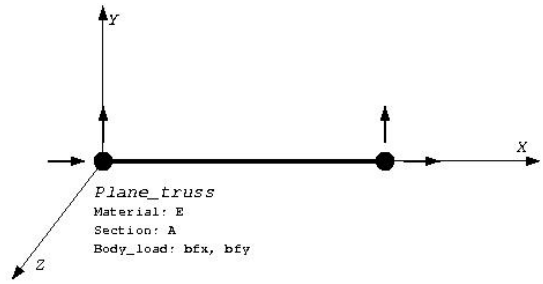


(h) Barra

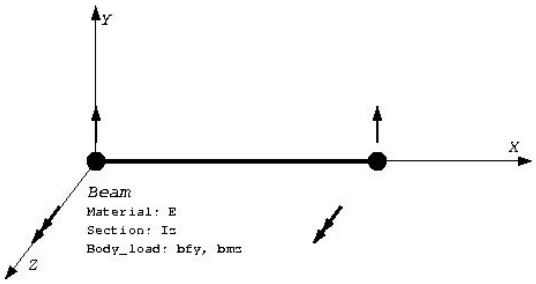


(i) Barra Deformada

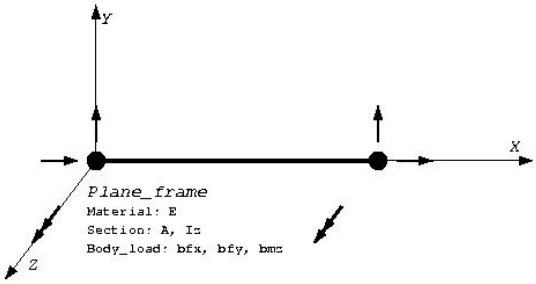
Figura 3: Grandezas de um Problema Estrutural modelado com Barras



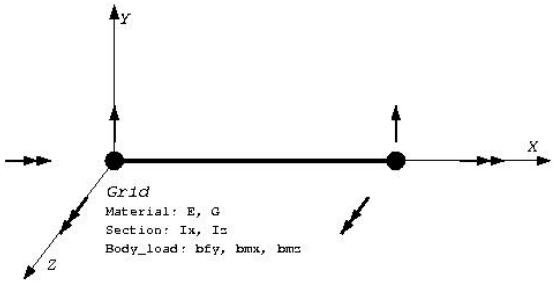
(a) Treliça Plana



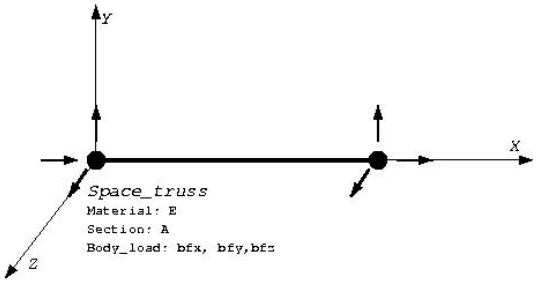
(b) Viga



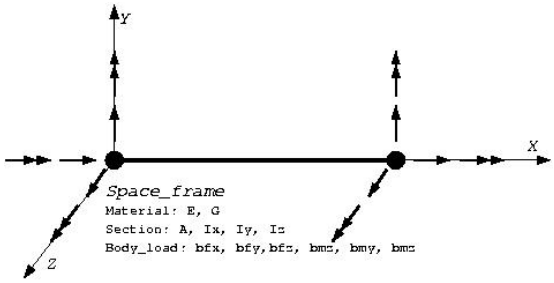
(c) Pórtico Plano



(d) Grelha



(e) Treliça Espacial



(f) Pórtico Espacial

Figura 4: Elementos Estruturais de Barras contemplados no Projeto Orientado a Objetos

4. PROJETO ORIENTADO A OBJETOS

Para subdividir a tarefa de modelagem e resolução de modelos discretos, optou-se, com base na proposta feita por Martha et al. (1996) e Pitangueira (1998), pela criação de três classes abstratas: `Driver`, `Model` e `Solution`. A classe `Driver` contém todos os métodos e atributos necessários para a montagem e resolução do modelo desejado. Um objeto `Driver` contém uma instância da classe `Solution` e uma da classe `Model` (figura 5).

A classe `Model` contém os dados relativos ao modelo que deve ser analisado, como sua geometria, propriedades dos materiais e o tipo de análise a ser realizada. `Solution` fornece as ferramentas matemáticas necessárias para a resolução do modelo.

`Driver` contém métodos que, a partir dos dados contidos em `Model` e utilizando as ferramentas de `Solution`, montam e resolvem o problema. Para o caso particular de modelos estruturais de barras, foram implementadas subclasses destas classes principais. Foram criadas as classes `StructuralMech`, `OnePointEq` e `FrameModel`, como subclasses de `Driver`, `Solution` e `Model`, respectivamente (figura 5).

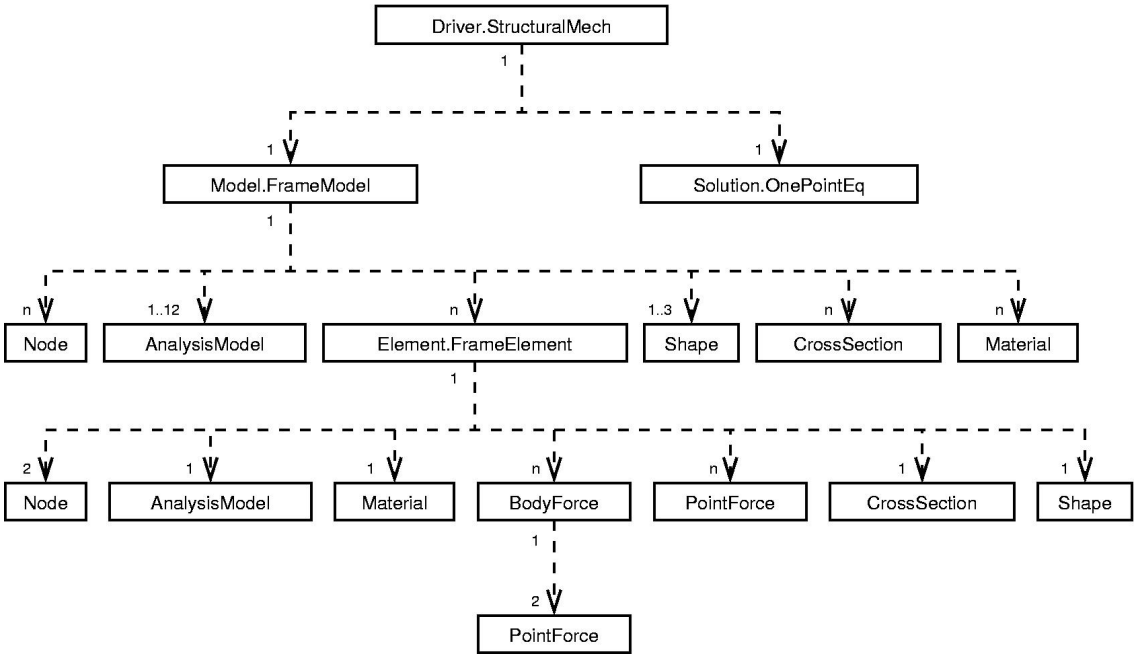


Figura 5: Intâncias de classes do programa

4.1 Classe `FrameModel`

`FrameModel`, como dito anteriormente, é a classe em que são armazenados os dados do modelo de barras. Estes dados são armazenados em listas, mais especificamente em instâncias da classe `java.util.ArrayList` da biblioteca padrão Java. O uso desta classe é justificado pelo fato de ela fornecer diversas ferramentas para a manipulação de listas e, principalmente, por não ser necessário informar um limite para o número de objetos que ela pode conter. Como um sistema pode apresentar infinitos nós, elementos, seções e etc., esta propriedade mostra-se bastante útil. As listas contidas em um `FrameModel` são as listas de nós, elementos, materiais, seções transversais, tipos de análise e funções de forma (ver figuras 3 e 5). Outros dois atributos desta classe são o número de equações do sistema e o número de restrições, sendo que estes dados são calculados durante a execução do programa. Os métodos da classe `FrameModel` são, em geral, métodos para manipulação das listas, como adicionar um objeto a uma lista, retornar o número de objetos contidos

em uma lista e retornar a própria lista.

4.2 Classe `OnePointEq`

A classe `OnePointEq` realiza a tarefa de obter um ponto de equilíbio do modelo. Possui três atributos e dois métodos principais.

Seus atributos são uma matriz, instância da classe `javax.vecmath.GMatrix`, e dois vetores, instâncias da classe `javax.vecmath.GVector`, sendo ambas estas classes fornecidas pela biblioteca `Java3D`. Os métodos principais desta classe são `kinematicSolver()` e `staticSolver()`, ambos sem parâmetros. O primeiro soluciona um sistema de equações lineares da forma $A * X = B$, em que A é uma matriz, X e B são vetores. O outro efetua a multiplicação de A por X, armazenando o resultado no vetor B. Para que estes métodos sejam executados corretamente, os valores da matriz e do vetor conhecido devem ser previamente atribuídos por método da classe `StructuralMech`. Além destes métodos e dos métodos para manipulação dos atributos, `OnePointEq` possui os métodos `subFromBVector()` e `addToBVector()`, definidos na classe base `Solution`, que recebem um `GVector` como parâmetro. Estes métodos são usados para se adicionar ou subtrair vetores de forças do vetor de forças do modelo.

4.3 Classe `StructuralMech`

A classe `StructuralMech` possui os diversos métodos numéricos para a montagm da matriz de rigidez e dos vetores de forças do modelo, bem como para outros cálculos porventura necessários. Seu principal método é `analyseModel()`, que chama outros métodos desta classe, atribuindo os devidos valores às variáveis de `OnePointEq`. Utiliza os métodos `kinematicSolver()` e `staticSolver()`, de `OnePointEq`, e armazena os resultados em `FrameModel`. O primeiro passo na execução dos cálculos é a numeração das equações dos nós e dos elementos. Isto é feito através dos métodos `numberNodeEquations()` e `numberElementEquations()`, ambos sem parâmetros. Durante a numeração das equações dos nós, são calculados e armazenados em `FrameModel` o número de equações e restrições do sistema. Deve-se, então, reduzir-se o vetor de equações dos elementos, para que eles contenham somente o número das equações válidas para o tipo de análise do elemento. Para isso, é chamado o método `reduceEquationsVector()`, sem parâmetros. Após isso, deve-se obter a solução cinemática do modelo, ou seja, calcular os deslocamentos incógnitos dos nós. Atribui-se à variável `A` de `OnePointEq`, uma `GMatrix`, a matriz de rigidez reduzida do sistema, calculada pelo método `reducedStiffnessMatrix()`, sem parâmetros. Atribui-se à variável `B` de `OnePointEq`, um `GVector`, o vetor de força nodal reduzido, calculado por `reducedNodalForceVector()`, sem parâmetros. Soma-se a este vetor os vetores de forças nodais equivalentes reduzidos devidos a cargas de corpo nos elementos e a deslocamentos prescritos nos nós, calculados pelos métodos `reducedEquivalentForceVector()` e `reducedPreDisplacementEquivalentForceVector()`, respectivamente, ambos sem parâmetros. Chamando-se o método `kinematicSolver()` de `OnePointEq`, obtém-se a solução cinemática do problema. Entretanto, é necessário armazenar em `FrameModel` os resultados obtidos. Para isso, é chamado o método `assignKinematicState()`, que recebe o vetor `X` de `OnePointEq` como parâmetro. Passa-se então à obtenção da solução estática do problema, objetivando-se calcular as reações de apoio e as ações nas extremidades de cada elemento. Atribui-se à variável `A` de `OnePointEq` a matriz de rigidez associada aos deslocamentos prescritos, calculada por `reactionsStiffnessMatrix()`, sem parâmetros. À variável `X` de `OnePointEq` é atribuído o vetor de deslocamentos reduzido do sistema, calculado por `reducedDisplacementVector()`,

sem parâmetros. Chamando-se o método `staticSolver()` de `OnePointEq`, obtém-se as forças internas atuantes em cada nó, armazenadas no vetor `B` de `OnePointEq`. Entretanto, deste vetor devem ser subtraídos os vetores de força nodal equivalente devido a cargas de corpo, de força nodal externa e de força nodal equivalente devido a deslocamentos prescritos, referentes às mesmas equações, calculados pelos métodos `reactionsEquivalentForceVector()`, `reactionsNodalForceVector()` e `reactionsPreDisplacementEquivalentForceVector()`, respectivamente, todos sem parâmetros. Os resultados obtidos são então armazenados em `FrameModel` pelo método `assignStaticState()`, que recebe o vetor `B` de `OnePointEq` como parâmetro. O último passo é o cálculo das ações nas extremidades de cada elemento, o que é feito pelo método `calculateActionsAtNodes()`, sem parâmetros.

4.4 Classe `AnalysisModel`

A classe abstrata `AnalysisModel` representa o tipo de análise escolhido para o modelo estrutural e para cada elemento específico. Seus atributos são o número de graus de liberdade de um elemento, as equações válidas e um identificador. Esta classe contém métodos para manipulação de atributos e dois métodos abstratos que devem ser implementados por suas subclasses. Estes métodos abstratos são ambos denominados `mountMatrixN()`, sendo que um recebe apenas um `GVector` como parâmetro e o outro dois, e são utilizados para a montagem da matriz $\underline{N}$ de um elemento.

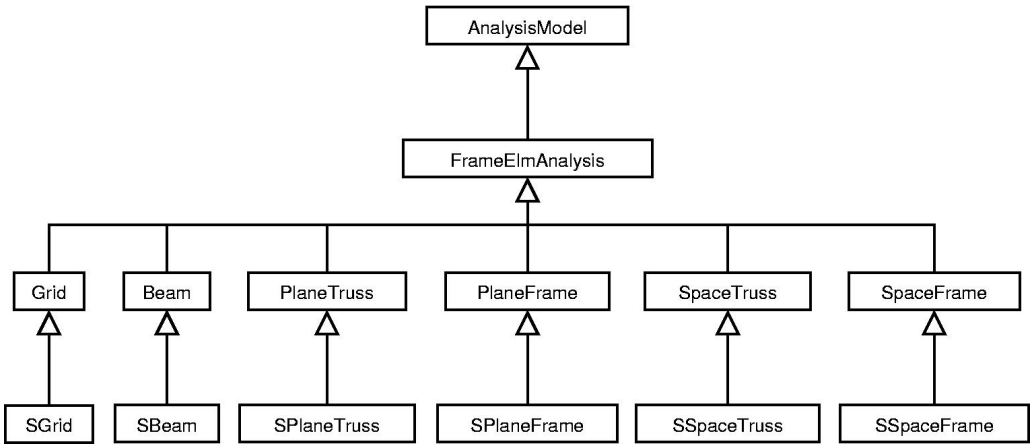


Figura 6: Hierarquia da classe `Analysis Model`

Para o caso de elementos de barra, foi implementada uma subclasse de `AnalysisModel` denominada `FrameElmAnalysis`, que também é abstrata. Esta classe tem como subclasses, classes que representam os seis tipos de análise de elementos de barra (ver figura 4), denominadas `Beam`, `Grid`, `PlaneTruss`, `PlaneFrame`, `SpaceTruss` e `SpaceFrame` (ver figura 6). Cada uma destas subclasses implementa os métodos abstratos definidos na classe base. Estes métodos retornam matrizes e vetores necessários para o cálculo das matrizes de rigidez e dos vetores de força do modelo. Um método muito importante é o `transformationMatrix()`, que recebe o próprio `FrameElement` como parâmetro, pois ele retorna a matriz de transformação do elemento que é usada para converter as matrizes e vetores calculados pelos outros métodos, das coordenadas locais do elemento para as coordenadas globais do modelo. Cada uma destas subclasses de `FrameElmAnalysis` tem uma subclasse que representa a análise de nós não-rígidos, denominadas `SBeam`, `SGrid`, `SPlaneTruss`, `SPlaneFrame`, `SSpaceTruss` e `SSpaceFrame` (figura 6).

4.5 Classe `PointForce`

A classe `PointForce` (ver figura 5) representa uma força concentrada. Seus atributos são um ponto tridimensional, instância da classe `javax.vecmath.Point3d` da biblioteca `Java3D`, um identificador e um `GVector` contendo as seis componentes possíveis da força. Seus métodos são apenas aqueles necessários para se acessar os atributos.

4.6 Classe `BodyForce`

A classe `BodyForce` (ver figura 5) representa um carregamento distribuído, composto por uma seqüência de objetos `PointForce`. Estes objetos da classe `PointForce` são armazenados em uma lista (`java.util.ArrayList`). Outro atributo de `BodyForce` é seu identificador. Seus métodos são apenas aqueles necessários para se acessar os atributos.

4.7 Classe `CrossSection`

A classe `CrossSection` (ver figura 5) representa a seção transversal de um elemento de barra. Seus atributos são um identificador, sua área, seus momentos de inércia, sua constante de torção e sua altura. Seus métodos são apenas aqueles necessários para se acessar os atributos.

4.8 Classe `Material`

A classe abstrata `Material` (ver figura 5) representa um material. Seus atributos são seu identificador e seu tipo, duas `Strings`, sendo que este último serve para identificar qual subclasse de `Material` foi criada. Seus métodos são apenas aqueles necessários para se acessar os atributos.

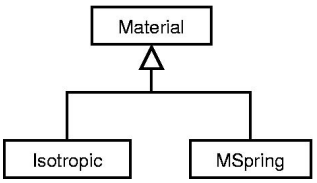


Figura 7: Hierarquia da classe `Material`

A classe `Isotropic`, subclasse de `Material`, representa um material isotrópico (ver figura 7). Seus atributos são os módulos de elasticidade longitudinal e transversal, o coeficiente de Poisson e o coeficiente de dilatação térmica. Seus métodos são apenas aqueles necessários para se acessar os atributos. A classe `Mspring`, subclasse de `Material`, representa o material de uma mola. Seu único atributo é um vetor que armazena os coeficientes de rigidez relacionados aos seis deslocamentos possíveis. Seus métodos são apenas aqueles necessários para se acessar este atributo.

4.9 Classe `Shape`

A classe abstrata `Shape` (ver figura 5) representa a função de forma de um elemento. Seu único atributo é seu tipo, que, assim como em `Material`, identifica qual subclasse foi criada. Seus métodos são apenas aqueles necessários para se acessar este atributo. Sua subclasse imediata é a classe `Shape1D`, também abstrata (ver figura 8). Esta classe define quatro métodos que devem ser implementados pelas suas subclasses. Estes métodos devem retornar a função de forma de um elemento avaliada em um ponto, a derivada desta função em um ponto, a integral desta função entre dois pontos e a integral desta função

multiplicada pela variável espacial (x), também avaliada entre dois pontos. As subclasses de `Shape1D` implementadas para o caso particular dos elementos de barra foram `Linear1D`, `Cubic1D` e `LinearCubic1D`, que representam, respectivamente, funções de forma lineares, cúbicas e lineares-cúbicas (figura 8). A classe `LinearCubic1D` foi criada somente para juntar as funções de forma linear e cúbica em uma única classe, de maneira a utilizar um único objeto `Shape` para cada tipo de análise.

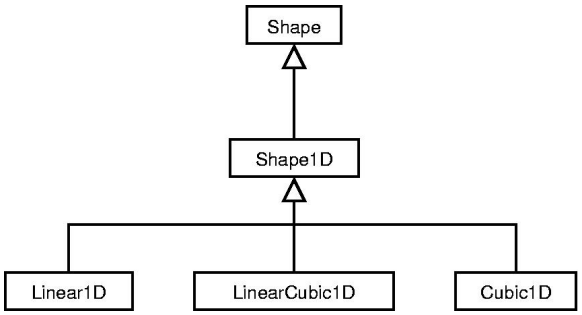


Figura 8: Hierarquia da classe `Shape`

4.10 Classe `Node`

A classe `Node` representa um nó (ver figura 5). Seus atributos, que devem ser informados antes da execução dos cálculos, são o identificador, as coordenadas, as forças externas atuantes sobre ele, as restrições, os deslocamentos prescritos, o ângulo de inclinação do nó em relação ao eixo X do sistema de coordenadas do modelo e os coeficientes de rigidez para o caso de o nó ser um apoio elástico. Durante a execução dos cálculos são numeradas as equações dos nós e preenche-se os vetores de deslocamentos e de reações de apoio. Seus métodos são apenas aqueles necessários para se acessar os atributos.

4.11 Classe `Element`

A classe abstrata `Element` representa um elemento finito qualquer (ver figura 5). Seus atributos são um identificador, sua incidência (uma lista de objetos `Node`), seus carregamentos, tanto concentrados quanto distribuídos (listas de `PointForce` e `BodyForce`, respectivamente), seu vetor de equações, seu material, sua forma e seu modelo de análise. Tanto a classe `Element` quanto suas subclasses não armazenam a maioria de seus atributos, como os nós e o material, elas apenas fazem referência aos objetos armazenados nas listas de `Model`. Além dos métodos necessários para se acessar os atributos, esta classe tem métodos abstratos que retornam a matriz de rigidez e o vetor de força nodal equivalente reduzido do elemento, ambos no sistema de coordenadas global. Há também um método abstrato que calcula as ações nos nós do elemento desde que os deslocamentos tenham sido previamente calculados.

A classe `FrameElement`, subclasse de `Element`, representa um elemento de barra (ver figura 9). Seus atributos são seu comprimento, sua seção transversal, suas liberações nas extremidades, suas variações de temperatura (nas fibras superior e inferior e na altura do centro de gravidade), suas pré-deformações, as ações em suas extremidades e um vetor contendo forças nodais equivalentes para ser usado quando se desejar analisar uma estrutura submetida a um carregamento não previsto por esta implementação. Esta classe implementa os métodos abstratos de `Element` da maneira adequada a um elemento de barra, além de possuir os métodos para acessar seus atributos.

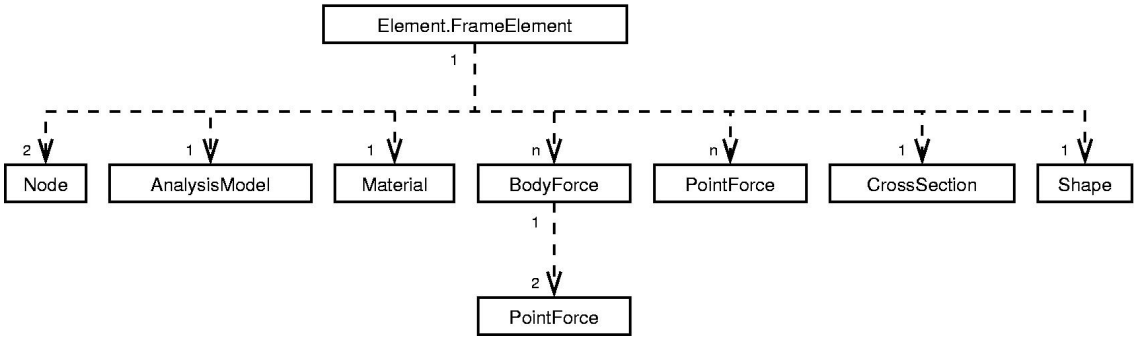


Figura 9: Hierarquia da classe `FrameElement`

5. CONSIDERAÇÕES FINAIS

O ensino do método dos elementos finitos (MEF) nos cursos de graduação de Engenharia contempla, inicialmente, os modelos estruturais de barras, para depois avançar na descrição de contínuos bi ou tridimensionais. O computador é normalmente utilizado como ferramenta auxiliar, através de programas que automatizam os procedimentos numéricos dos modelos. O estágio atual de evolução da computação gráfica permite dar saltos qualitativos relevantes no ensino do MEF, se forem utilizados programas computacionais que possibilitem a descrição dos dados de entrada (pré-processamento) e a visualização dos resultados de saída (pós-processamento) dos modelos, através de recursos gráficos interativos.

No estágio atual do desenvolvimento do sistema aqui apresentado, a parte da aplicação `NUCLEUS` relativa a modelos estruturais de barras (discutida neste artigo) já está implementada e exaustivamente testada. A persistência dos dados em arquivos `XML` também já foi implementada e testada.

A próxima etapa do trabalho é o desenvolvimento da aplicação `JMESH`, de maneira a disponibilizar recursos gráficos interativos para facilitar o pré-processamento dos dados de entrada da aplicação `NUCLEUS`. A implementação de `JMESH` que contempla elementos unidimensionais já está em andamento. Também já iniciou-se a implementação de geração de malhas de elementos finitos bidimensionais, baseada em mapeamentos. Outra tarefa, também já em andamento, é a expansão de `NUCLEUS` para contemplar elementos finitos paramétricos bi e tridimensionais.

Espera-se que o desenvolvimento do sistema aqui apresentado diminua as barreiras existentes entre o desenvolvimento teórico de modelos discretos de análise e sua aplicação. Também deseja-se que o sistema seja fomentador do desenvolvimento de novos modelos, evitando o recomeço do processo de implementação e permitindo maior agilidade e criatividade da pesquisa na área.

Agradecimentos

Ao apoio financeiro em forma de bolsa dado pela Pró-Reitoria de Graduação da UFMG, através do programa PAD, e pelo CNPq, através do programa PIBIC.

Referências

Alvim, P. (2003) Open Source: Os Novos Desafios de Negócios e a Indústria de TI, Developers Magazine, No. 80, pp 13-15.

Braz, M. R. (2003) Tecnologia de Web Services: Definições e Perspectivas, Developers Magazine, No. 80, pp 22-24.

Flanagan, D. (2000) Java, O Guia Essencial, Editora Campus.

Gamma, E., Helm, R., Johnson, R. e Vlissides, J. (1995) Design Paterns - Elements of Reusable Object-Oriented Software

Martha, L. F., Menezes, I. F. M., Lages, E. N., Parente Jr. E. P. e Pitangueira, R. L. (1996) "An OOP Class Organization for Materially Nonlinear Finite Element Analysis", XVII CILAMCE, Padova, Itália, pp 229-232.

Pietro, G. A. (2001) Utilização de Padrões de Projeto na Reengenharia de Sistemas, Dissertação de Mestrado - UFSCar, São Carlos - SP.

Pitangueira, R. L. (1998) Mecânica de Estruturas de Concreto com Inclusão de Efeitos de Tamanho e Heterogeneidade, Tese de Doutorado, Departamento de Engenharia Civil da PUC-Rio, Rio de Janeiro.

Rowe, G. W. (2001) Computer Graphics with Java, Palgrave.

Soriano, H. L. e Lima, S. S. (1999) Método de Elementos Finitos em Análise de Estrutura, Universidade Federal do Rio de Janeiro.