



PUC

EDUARDO SETTON SAMPAIO DA SILVEIRA

*UM SISTEMA DE MODELAGEM BIDIMENSIONAL CONFIGURÁVEL
PARA SIMULAÇÃO ADAPTATIVA EM MECÂNICA COMPUTACIONAL*

DISSERTAÇÃO DE MESTRADO

Departamento de Engenharia Civil
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Rio de Janeiro, Agosto de 1995

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

N. Chamada: 624 / 5857 / TESE UC

Título: Um sistema de modelagem bidimensional co



0 0 9 3 7 3 0

Ex: 1-CENTRAL

9232

EDUARDO SETTON SAMPAIO DA SILVEIRA

**Um Sistema de Modelagem Bidimensional Configurável para
Simulação Adaptativa em Mecânica Computacional**

Dissertação apresentada ao Departamento de
Engenharia Civil da PUC-Rio como parte dos
requisitos para a obtenção do título de Mestre
em Ciências em Engenharia Civil: Estruturas.

Orientador: Luiz Fernando Ramos Martha

Departamento de Engenharia Civil
Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 30 de Agosto de 1995

64359

uc-00065354-1



624
5857
TESEUC

Aos meus pais Carlos Alípio e Vera Lúcia (*in memoriam*), a minha avó Pauline (*in memoriam*), aos meus irmãos Rico, Leo e Carla e a minha esposa Daniella com muito amor o meu muito obrigado por tudo.

Agradecimentos

Ao professor Luiz Fernando Martha pela orientação, pelo incentivo constante e principalmente pela dedicação e amizade que foram fundamentais ao longo do desenvolvimento deste trabalho.

Ao amigo Marcelo Tílio Monteiro de Carvalho pelo apoio e orientação em todos os momentos do desenvolvimento deste trabalho.

Ao amigo Joaquim pelo incentivo, pelos conselhos, pela colaboração em muitas das etapas deste trabalho e principalmente pela amizade consolidada em tão pouco tempo de convívio.

Ao meu grande amigo Leo, que tão prematuramente nos deixou, pelas conversas, pelo humor constante que nos deu tanta alegria e principalmente pela nossa amizade que jamais será esquecida.

Aos meus grandes amigos de Maceió: Guga, Marquinho, Valmir e Eduardo Martins pelas conversas e pelos bons e inesquecíveis momentos vividos juntos.

Ao professor Roberaldo Carvalho de Souza e ao grupo PET de Engenharia Civil da Universidade Federal de Alagoas pelos ensinamentos e pelos bons momentos vividos ao longo da graduação.

Ao professor e amigo Adeildo pelos conhecimentos transmitidos no período de graduação e a Viviane pelo agradável convívio durante estes anos.

Aos amigos da PUC: Evandro, Áurea, Alexandre, Lucas, Stoessel, Ana Marta, Eduardo Thadeu, Roque, Andréia, Marquinho, Nobre, Kubrusly, Claudinha, Denyse, Márcio, William e Beбето.

Aos amigos do TeCGraf: Ivan, Waldemar, Lula, Camilo, Cacá, João Luiz, Raquel, Costa, Renato Cerqueira, Borges, Cassino, Clínio, Gil, Rodacki, Arlindo, Sedrez, Peter, Nick, Mônica, Ovídio e Anderson.

Aos amigos botafoguenses Lucas, Stoessel, Camilo, Lula, Thadeu e Rogério pelos grandes e inesquecíveis momentos vividos juntos com o nosso querido Botafogo.

Aos funcionários do Departamento de Engenharia Civil e do TeCGraf, principalmente a Ana Roxo pelo grande apoio dado durante todo o mestrado.

À CAPES pelo apoio financeiro durante a graduação e em todo curso de mestrado.

Esta tese apresenta um sistema extensível e configurável para simulação adaptativa de problemas bidimensionais de mecânica computacional pelo método dos elementos finitos. O sistema possibilita a configuração dos atributos a serem utilizados de acordo com o tipo de simulação desejada. Esta configuração é feita pelo usuário através de uma linguagem interpretada relativamente simples. O usuário com algum conhecimento define, através de um arquivo de configuração, os tipos de atributos específicos ao problema a ser analisado. O sistema também provê uma ferramenta interativa para manipulação deste arquivo.

A arquitetura do sistema é dividida em três módulos independentes. O primeiro é um modelador genérico para simulação adaptativa. Este módulo é completamente genérico no que se refere aos atributos do problema a ser analisado. Ele incorpora todas as etapas de modelagem deste tipo de simulação: modelagem geométrica, geração automática de malha, estratégia de refinamento de malha e visualização de resultados. O segundo módulo faz a análise por elementos finitos do modelo discretizado e estima o erro numérico de discretização. No caso, foi usado um programa baseado em programação orientada a objetos. O terceiro módulo é responsável pela configuração e gerenciamento dos atributos da simulação.

O principal objetivo deste trabalho é descrever, implementar e verificar a arquitetura do módulo de configuração e integrá-lo ao modelador de simulação adaptativa bidimensional. A configuração é feita através de uma interface baseada em diálogos para captura de dados e de uma linguagem acoplada e interpretada, com recursos de orientação a objetos. Isto possibilita uma extensão genérica e o acesso programado de modeladores de mecânica computacional. Exemplos de simulação e configuração são apresentados.

Abstract

This dissertation presents an extensible and customizable system for two-dimensional, adaptive simulation in computational mechanics through the finite element method. The system allows for the configuration of the desired simulation attributes. The customization is performed through a relatively simple interpreted language. A knowledgeable user defines, through a configuration file, the types of attributes which are specific for the problem to be analyzed. The system also provides a browser for manipulating this file.

The system architecture is divided in three modules. The first one is a generic modeler for adaptive simulation. It is generic with respect to the attributes of the problem to be analyzed. It incorporates all the modeling stages of this type of simulation: geometric modeling, automatic mesh generation, mesh refinement strategy, and visualization of results. The second module performs the finite element analysis and estimates the discretization numerical error. In this case, a program based on object oriented programming was used. The third module is responsible for the configuration and management of simulation attributes.

The main objective of this work is the description, implementation, and verification of the architecture of the configuration module and the integration of this module with the two-dimensional adaptive simulation modeler. The configuration is performed through an interface based on dialogs for data acquisition and through an embedded and interpreted language, with object oriented capabilities. This allows for generic extensions and programmable accesses of computational mechanics modelers. Examples of simulation and configuration are shown.

1 INTRODUÇÃO	1
1.1 OBJETIVOS	3
1.2 DEFINIÇÕES E CONCEITOS	4
1.2.1 ATRIBUTOS EM MECÂNICA COMPUTACIONAL	4
1.2.2 PROGRAMAÇÃO ORIENTADA A OBJETOS	5
1.3 ORGANIZAÇÃO DA DISSERTAÇÃO	7
2 SIMULAÇÃO ADAPTATIVA EM MECÂNICA COMPUTACIONAL	8
2.1 SISTEMA ADAPTATIVO BASEADO EM MODELAGEM GEOMÉTRICA PARA SIMULAÇÃO DE PROBLEMAS DE ELASTICIDADE PLANA	10
2.2 MODELAGEM GEOMÉTRICA E ESPECIFICAÇÃO DE ATRIBUTOS	12
2.2.1 MODELAGEM GEOMÉTRICA	12
2.2.2 ESPECIFICAÇÃO DOS ATRIBUTOS	13
2.3 GERAÇÃO AUTOMÁTICA DA MALHA INICIAL	15
2.4 MAPEAMENTO DOS ATRIBUTOS PARA MALHA	16
2.5 ANÁLISE NUMÉRICA	18
2.6 REFINAMENTO DAS FRONTEIRAS E GERAÇÃO DA NOVA MALHA	19
2.7 VISUALIZAÇÃO DOS RESULTADOS	20
3 UM SISTEMA EXTENSÍVEL PARA CONFIGURAÇÃO DE ATRIBUTOS EM MECÂNICA COMPUTACIONAL	22
3.1 ARQUITETURA DO AMBIENTE DE CONFIGURAÇÃO	23
3.2 DESCRIÇÃO DO SISTEMA	26
3.3 FERRAMENTAS UTILIZADAS	28
3.3.1 LINGUAGEM DE CONFIGURAÇÃO - LUA	28
3.3.2 O <i>TOOLKIT</i> EDG	29
3.4 <i>CORE CLASSES</i>	29
3.4.1 <i>ATTRIBUTE CLASSES</i>	30
3.4.2 <i>MODELING CLASSES</i>	31
3.4.2.1 CLASSE <i>MODEL</i>	32
3.4.2.2 CLASSE <i>GEOMETRY</i>	33
3.4.2.3 CLASSE <i>NUMERICAL MODEL</i>	34
3.4.2.4 CLASSE <i>NODE</i>	34
3.4.2.5 CLASSE <i>ELEMENT</i>	35
3.4.2.6 CLASSE <i>ELEMENT FEATURE</i>	37
3.5 FUNÇÕES DE ACESSO À TECNOLOGIA DA APLICAÇÃO	39
3.6 LIGAÇÃO ENTRE A PARTE CONFIGURÁVEL E O MODELADOR	40

3.7 SERVIÇOS OFERECIDOS PELO SISTEMA	40
3.8 ESAM <i>BROWSER</i>	43
4 SIMULAÇÃO ADAPTATIVA EXTENSÍVEL E CONFIGURÁVEL	46
4.1 INTEGRAÇÃO DO ESAM COM O MÓDULO DE MODELAGEM GEOMÉTRICA	47
4.1.1 FUNÇÕES MIS PARA O MODELADOR GEOMÉTRICO	48
4.1.2 SERVIÇOS DO ESAM UTILIZADOS PELO MODELADOR GEOMÉTRICO	50
4.2 INTEGRAÇÃO DO ESAM COM O GERADOR DO MODELO NUMÉRICO	54
4.2.1 FUNÇÕES MIS PARA O GERADOR DO MODELO NUMÉRICO	54
4.2.2 SERVIÇOS DO ESAM UTILIZADOS PELO GERADOR DE MALHAS	56
4.3 RESUMO DA ARQUITETURA PROPOSTA	57
4.3.1 MODELAGEM GEOMÉTRICA COM CONFIGURAÇÃO DE ATRIBUTOS	58
4.3.2 MAPEAMENTO DOS ATRIBUTOS PARA A MALHA DENTRO DE UM AMBIENTE CONFIGURÁVEL	59
4.3.3 FUNCIONALIDADE DA ARQUITETURA PROPOSTA	60
5 EXEMPLOS	63
5.1 ANÁLISE DE TENSÕES	64
5.2 ANÁLISE TÉRMICA	73
6 CONCLUSÃO	80
6.1 PRINCIPAIS CONTRIBUIÇÕES	81
6.2 SUGESTÕES PARA TRABALHOS FUTUROS	82
7 REFERÊNCIAS BIBLIOGRÁFICAS	84
APÊNDICE A - DESCRIÇÃO DAS CORE CLASSES	88
A.1 CLASSE <i>ATTRIBUTE</i>	88
A.2 CLASSE <i>MODEL</i>	89
A.3 CLASSE <i>GEOMETRY</i>	91
A.4 CLASSE <i>NUMERICAL MODEL</i>	91
A.5 CLASSE <i>NODE</i>	92
A.6 CLASSE <i>ELEMENT</i>	93
A.7 CLASSE <i>ELEMENT FEATURE</i>	93
APÊNDICE B - DESCRIÇÃO DAS FUNÇÕES MIS	95

Lista de Figuras

FIGURA 1.1 - TRÊS PROBLEMAS DISTINTOS PARA ANÁLISE MECÂNICA.....	4
FIGURA 2.1 - PROCEDIMENTO GENÉRICO PARA SIMULAÇÃO ADAPTATIVA BASEADA EM GEOMETRIA	10
FIGURA 2.2 - MÓDULOS E FUNCIONAMENTO DO SISTEMA APRESENTADO.....	11
FIGURA 2.3 - MODELAGEM GEOMÉTRICA E ESPECIFICAÇÃO DOS ATRIBUTOS.....	14
FIGURA 2.4 - GERAÇÃO AUTOMÁTICA DA MALHA INICIAL.....	16
FIGURA 2.5 - REFINAMENTO DAS FRONTEIRAS E GERAÇÃO DA NOVA MALHA.....	20
FIGURA 2.6 - VISUALIZAÇÃO DOS RESULTADOS.....	21
FIGURA 3.1 - APLICAÇÃO QUE UTILIZA O ESAM.....	23
FIGURA 3.2 - ARQUITETURA DO SISTEMA ESAM.....	26
FIGURA 3.3 - ORGANIZAÇÃO DAS <i>CORE CLASSES</i>	30
FIGURA 3.4 - DIÁLOGO DISPARADO PELO MÉTODO <i>CREATEATTRIBUTE</i> DA CLASSE <i>MODEL</i>	32
FIGURA 3.5 - RELAÇÕES ENTRE OS OBJETOS DAS CLASSES <i>NODE</i> , <i>ELEMENT</i> , <i>GEOMETRY</i> E <i>ATTRIBUTE</i>	38
FIGURA 3.6 - <i>ESAM BROWSER</i>	45
FIGURA 4.1 - INTEGRAÇÃO DO ESAM COM O MÓDULO DE SIMULAÇÃO.....	47
FIGURA 4.2 - FUNÇÃO QUE REGISTRA OS TIPOS DE ENTIDADES GEOMÉTRICAS UTILIZADAS PELO MODELADOR	49
FIGURA 4.3 - FUNÇÃO UTILIZADA PELO ESAM PARA REQUISITAR AS ENTIDADES GEOMÉTRICAS SELECIONADAS.....	49
FIGURA 4.4 - FUNÇÃO PARA INFORMAR AO MODELADOR QUAIS AS ENTIDADES GEOMÉTRICAS QUE CONTÊM INFORMAÇÕES DE ATRIBUTOS.....	50
FIGURA 4.5 - DIÁLOGO DISPARADO PELO SERVIÇO DE ESPECIFICAÇÃO DE ATRIBUTOS.....	51
FIGURA 4.6 - DIÁLOGO DISPARADO PELO SERVIÇO DE CRIAÇÃO DO ATRIBUTO.....	52
FIGURA 4.7 - DIÁLOGO DISPARADO PELO SERVIÇO DE CONSULTA.....	53
FIGURA 4.8 - FUNÇÃO PARA REGISTRO DOS TIPOS DE ELEMENTOS FINITOS UTILIZADOS PELO MODELADOR	55
FIGURA 4.9 - FUNÇÃO QUE PASSA PARA O ESAM OS IDENTIFICADORES DOS ELEMENTOS DA MALHA	55
FIGURA 4.10 - FUNÇÃO PARA CAPTURA DAS COORDENADAS DE UM NÓ DA MALHA.....	55
FIGURA 4.11 - FUNÇÃO UTILIZADA PARA OBTENÇÃO DA ENTIDADE GEOMÉTRICA ASSOCIADA A UM ELEMENTO DA MALHA.....	56
FIGURA 4.12 - INTEGRAÇÃO DO SISTEMA PARA SIMULAÇÃO ADAPTATIVA BIDIMENSIONAL DE ELEMENTOS FINITOS COM O SISTEMA ESAM.....	58
FIGURA 5.1 - PLACA EM L.....	64
FIGURA 5.2 - UTILIZAÇÃO DO <i>BROWSER</i> PARA CRIAÇÃO DO ATRIBUTO NOVO TIPO DE ATRIBUTO (<i>MATERIAL ISOTRÓPICO</i>).....	65
FIGURA 5.3 - CÓDIGO GERADO PELO <i>BROWSER</i> PARA CONFIGURAÇÃO DO ATRIBUTO <i>MATERIAL</i> <i>ISOTRÓPICO</i>	66
FIGURA 5.4 - DIÁLOGO CRIADO PARA CAPTURA DOS DADOS REFERENTES AO <i>MATERIAL</i> <i>ISOTRÓPICO</i>	67

FIGURA 5.5 - GERAÇÃO DA GEOMETRIA E ESPECIFICAÇÃO DOS ATRIBUTOS.....	68
FIGURA 5.6 - MALHA INICIAL.....	69
FIGURA 5.7 - MÉTODO RESPONSÁVEL PELA GERAÇÃO DO ARQUIVO QUE CONTÉM O MODELO NUMÉRICO.....	70
FIGURA 5.8 - ARQUIVO CONTENDO AS INFORMAÇÕES SOBRE O MODELO NUMÉRICO INICIAL.....	71
FIGURA 5.9 - MALHA REFINADA.....	72
FIGURA 5.10 - VISUALIZAÇÃO DOS RESULTADOS.....	73
FIGURA 5.11 - CHAPA AQUECIDA.....	73
FIGURA 5.12 - UTILIZAÇÃO DO <i>BROWSER</i> PARA CRIAÇÃO DO ATRIBUTO TEMPERATURA.....	74
FIGURA 5.13 - CÓDIGO GERADO ATRAVÉS DO <i>BROWSER</i> PARA CONFIGURAÇÃO DO ATRIBUTO TEMPERATURA.....	75
FIGURA 5.14 - MODELO GEOMÉTRICO E ESPECIFICAÇÃO DOS ATRIBUTOS.....	76
FIGURA 5.15 - MALHA INICIAL.....	76
FIGURA 5.16 - ARQUIVO COM AS INFORMAÇÕES SOBRE O MODELO NUMÉRICO INICIAL.....	77
FIGURA 5.17 - REFINAMENTO DA MALHA.....	78
FIGURA 5.18 - VISUALIZAÇÃO DOS RESULTADOS.....	79

Entre os métodos numéricos existentes para a análise de problemas em engenharia o método dos elementos finitos tem sido um dos mais utilizados. Isto se deve principalmente aos bons resultados obtidos e à sua aplicação em uma grande variedade de problemas em mecânica computacional. Como exemplo, pode-se citar os problemas de análise estrutural de peças mecânicas (estudo de deslocamentos e tensões), problemas envolvendo mecânica dos fluidos e análise térmica, entre outros.

Todos estes tipos de análise têm em comum o fato de que se baseiam na solução de um problema de equações diferenciais parciais que na grande maioria das vezes não tem solução analítica, ou pelo menos é difícil de ser obtida, e sua solução tem que ser obtida via métodos numéricos. De uma forma geral, o método dos elementos finitos requer a geração de um modelo numérico que consiste, basicamente, de uma discretização do domínio do modelo e de atributos referentes ao tipo de problema a ser resolvido associados a esta discretização. Como a discretização do domínio pode ser executada de forma independente do tipo de problema, o que particulariza a simulação de mecânica computacional, no contexto da utilização do método dos elementos finitos, são os atributos associados ao tipo de análise. Em um problema de elasticidade plana, por exemplo, alguns dos atributos referentes a este problema são: condições de suporte, cargas concentradas, cargas distribuídas e deslocamentos prescritos. Um problema de análise térmica, por sua vez, tem como alguns dos seus atributos fluxo de calor, campo de temperaturas iniciais ou temperatura prescrita no contorno do modelo.

Os sistemas existentes utilizados para simulação numérica, normalmente, são muito específicos para um determinado tipo de análise, o que muitas vezes dificulta ou até mesmo inviabiliza a sua utilização em outros tipos de análise. Isto porque a consideração dos atributos da simulação está intrinsicamente ligada às atividades de modelagem, geração de malha, análise numérica e visualização. Um desses sistemas, denominado AMVIEW (*Adaptative Mesh View*), desenvolvido por Cavalcante (1994), será

apresentado no decorrer desta dissertação, pois serviu como base para o desenvolvimento deste trabalho.

Esta dissertação propõe um sistema que oferece um ambiente onde é possível a realização de simulações adaptativas pelo método dos elementos finitos com uma grande flexibilidade no que diz respeito à criação e manipulação dos atributos utilizados. Neste sistema é possível a realização de diversos tipos de simulação através da simples configuração dos atributos envolvidos na análise a ser realizada. A configuração destes atributos é realizada sem que haja a necessidade do conhecimento sobre a tecnologia utilizada no modelador e é feita de uma forma relativamente fácil e rápida pelo configurador da aplicação. A configuração dos atributos é realizada em tempo de execução, de forma que não há a necessidade de recompilação e religação do sistema. Além disso, o sistema pode ser executado em várias plataformas computacionais.

Avanços de diversas tecnologias em diferentes áreas da engenharia de *software* e mecânica computacional contribuíram para o desenvolvimento do sistema com a flexibilidade proposta:

- Linguagens acopladas para extensões e configurações de aplicações, como Tcl [Ousterhout 1994] ou Lua [Jerusalimschy *et al.* 1994, Figueiredo *et al.* 1994]. Estas linguagens proporcionam facilidade e rapidez na extensão e configuração das aplicações devido ao fato de serem linguagens interpretadas, ou seja, compiladas em tempo de execução. No caso da linguagem Lua, a simplicidade da sintaxe da linguagem também facilita muito a extensão e configuração das aplicações.
- Paradigma de programação orientada a objetos [Goldberg 1983, Cox-Novobilsky 1991] que oferece aos programadores um alto grau de reutilização de códigos já implementados. A modularização proporcionada por este paradigma também facilita o entendimento do código e sua expansão.
- Técnicas de modelagem geométrica baseadas em subdivisões planares, onde a inserção de curvas resulta no reconhecimento automático de regiões e relações de adjacências completas [Cavalcanti 1992].
- Análise pelo método dos elementos finitos utilizando programação orientada a objetos. O programa FEMOOP [Guimarães 1992] é uma boa ilustração deste tópico.

- Simulação adaptativa bidimensional pelo método dos elementos finitos baseada em geometria, onde os atributos são aplicados à entidades geométricas ao invés de aplicados à malha [Finnigam *et al.* 1989, Martha 1989, Vianna 1992 e Cavalcante 1994].
- Interfaces e sistemas gráficos, como o IUP [Levy 1993] e o GKS [TeCGraf 1989] respectivamente, que oferecem grande portabilidade, permitindo que os sistemas desenvolvidos com a utilização destas bibliotecas possam ser utilizados em qualquer plataforma de trabalho. E outros *Toolkits* que possibilitam o fácil desenvolvimento de interfaces gráficas para captura de dados. Como exemplo, tem-se o EDG [Celes 1995, Celes *et al.* 1995] e o Tk [Ousterhout 1994].

Alguns dos fatores citados foram combinados em uma arquitetura proposta por Carvalho (1995), que foi também tomada como base para este trabalho, para o desenvolvimento de simuladores de mecânica computacional extensíveis e configuráveis.

1.1 Objetivos

De acordo com o exposto acima, os objetivos deste trabalho são:

- Implementar um sistema extensível para a configuração de atributos de mecânica computacional baseado na arquitetura proposta por Carvalho (1995);
- Utilizar o sistema implementado em um sistema para simulação adaptativa [Cavalcante 1994], buscando testar e aprimorar o sistema extensível para a configuração de atributos;
- Integrar o gerenciador de atributos implementado com o sistema para simulação adaptativa de problemas bidimensionais através do método dos elementos finitos desenvolvido por Cavalcante (1994), obtendo com esta integração um sistema onde os atributos utilizados na simulação são configurados de acordo com o tipo de análise desejada.

1.2 Definições e conceitos

Esta seção apresenta algumas definições e conceitos que podem ser úteis para o entendimento deste trabalho.

1.2.1 Atributos em mecânica computacional

Uma simulação computacional de um problema de mecânica consiste na realização de uma análise numérica de um problema físico. Para realizar uma simulação computacional é necessária a geração de um modelo que represente consistentemente o modelo físico a ser analisado. Em simulações numéricas com a utilização do método dos elementos finitos, conforme dito, é necessária a geração de um modelo numérico, que consiste de uma discretização do domínio do modelo e de algumas propriedades associadas a este domínio. Estas propriedades serão referenciadas no decorrer deste trabalho como *atributos da simulação*. No contexto deste trabalho, pode-se entender atributos como sendo todas as informações adicionais à geometria e topologia do modelo que são necessárias à definição completa do problema.

Com o objetivo de caracterizar melhor estes atributos, são apresentados a seguir três problemas distintos de mecânica.

Para os três problemas ilustrados na Fig. 1.1 pode-se observar que a geometria que define o domínio do problema é a mesma. O que varia para cada um dos problemas apresentados são os atributos associados aos respectivos modelos.

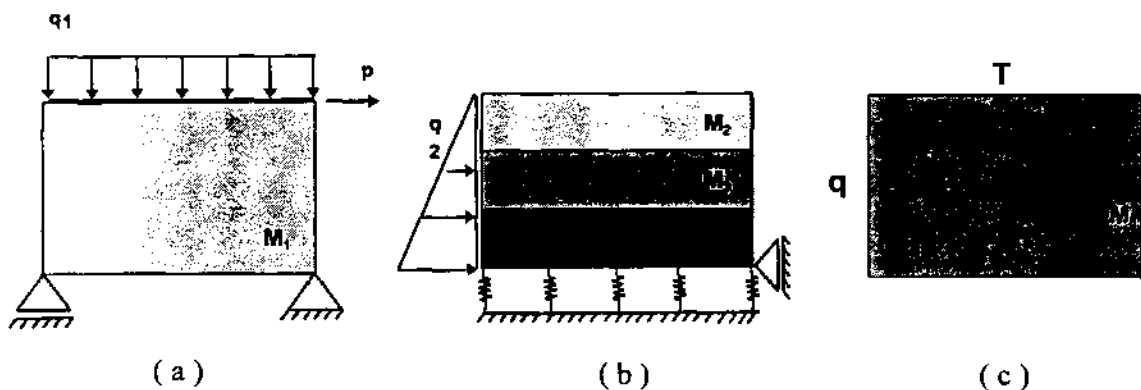


Figura 1.1 - Três problemas distintos para análise mecânica.

A Fig. 1.1-a apresenta um problema de análise estrutural onde os atributos específicos apresentados neste problema são os seguintes:

- Material (M_1) definido por suas propriedades físicas.
- Carregamento linearmente distribuído (q_1) definido por seu valor escalar e sua direção.
- Carga concentrada (p) aplicada no extremo superior da peça.
- Condições de suporte que restringem os deslocamentos dos pontos extremos inferiores em determinadas direções.

O tipo de problema apresentado na Fig. 1.1-b utiliza diferentes tipos de material, um carregamento triangular e um tipo de condição de suporte diferente da utilizada no problema da Fig. 1.1-a.

Na Fig. 1.1-c é mostrado um problema que diz respeito a uma análise térmica, onde os atributos utilizados para este tipo de análise são completamente diferentes dos dois primeiros problemas apresentados:

- Material (M_5) definido por algumas propriedades físicas adicionais que devem ser fornecidas para este tipo de problema, diferentes dos materiais utilizados nos outros casos. Por exemplo, a condutividade térmica do material.
- A temperatura (T) definida por um valor escalar.
- O fluxo de calor (q_3) definido também por um valor escalar.

1.2.2 Programação orientada a objetos

Segundo o paradigma de programação orientada a objetos, os programas podem ser vistos como um conjunto de tipos de dados abstratos e uma estratégia de controle que manipula estes dados para produzir o resultado esperado. Cada tipo de dado abstrato tem sua própria representação interna e um apropriado conjunto de rotinas que operam sobre os seus dados [Cox-Novobilsky 1991].

A seguir são descritos conceitos fundamentais de programação orientada a objetos, com objetivo de definir alguns termos que aparecem no trabalho.

Neste contexto, uma classe pode ser definida como uma categoria, ou grupo, composta por dados e procedimentos. Para formação deste grupo, é fundamental que seus elementos apresentem características comuns. Um outro conceito importante é o conceito de objetos. Objetos podem ser vistos como elementos de uma classe, também chamados instâncias. Estes objetos são componentes compostos de dados e procedimentos, que foram definidos na classe da qual eles fazem parte. A relação entre um objeto e a sua classe é a mesma que se estabelece entre uma variável declarada e o seu tipo.

Operações são realizadas sobre os objetos através de mensagens emitidas para eles. Quando um objeto recebe uma mensagem, executa um dos seus procedimentos, que são chamados de métodos. Esses métodos operam sobre os dados do objeto, de modo que os detalhes de implementação do objeto são escondidos do resto do programa que usa este objeto. Isto proporciona uma maior facilidade e rapidez na expansão do código, visto que esta filosofia de programação permite que novos procedimentos ou até mesmo novos componentes possam ser inseridos no código existente, sem a alteração da estrutura anterior e com a reutilização de toda tecnologia implementada neste código.

Um outro conceito importante utilizado é o conceito de herança. Dentro desta filosofia de programação é permitido derivar uma classe a partir de outras classes já existentes, que neste trabalho serão referenciadas como superclasses ou classes-base. Desta forma, quando um método não é encontrado em uma classe, é procurado, recursivamente, nas suas superclasses. Isto também facilita o re-uso do código pois uma classe nova, que difere apenas em alguns procedimentos de uma existente, é derivada da classe existente e somente os novos métodos são implementados.

1.3 Organização da dissertação

Esta dissertação foi dividida em 6 capítulos. Esta seção dá uma visão geral do trabalho descrevendo resumidamente o conteúdo de cada capítulo.

O capítulo 2 descreve as etapas envolvidas em uma simulação adaptativa em mecânica computacional baseada em geometria, mostrando a funcionalidade de cada uma destas etapas e descrevendo resumidamente cada uma delas. É apresentado, ainda no capítulo 2, um sistema para simulação adaptativa de problemas de elasticidade plana desenvolvido em outros trabalhos desta linha de pesquisa [Cavalcanti 1992, Guimarães 1992, Cavalcante 1994] e que foi tomado como base para o desenvolvimento deste trabalho.

O capítulo 3 apresenta um ambiente extensível para simulações de mecânica computacional proposto por Carvalho (1995). Neste capítulo descreve-se a arquitetura deste ambiente e o sistema extensível para gerenciamento de atributos que foi implementado baseado nesta arquitetura. É feita uma descrição detalhada do sistema, desenvolvido segundo uma disciplina de programação orientada a objetos, mostrando sua organização de classes e sua funcionalidade dentro de um processo de simulação de mecânica computacional baseada em geometria.

No capítulo 4 é apresentado o sistema configurável para simulação adaptativa proposto obtido a partir da integração do sistema descrito no capítulo 2 com o sistema extensível para configuração de atributos detalhado no capítulo 3. É mostrado como foi feita a integração destes sistemas. Descreve-se ainda a funcionalidade do sistema configurável para simulação adaptativa obtido, ressaltando as diferenças entre este sistema e o sistema apresentado no capítulo 2.

No capítulo 5 são mostrados dois exemplos de utilização do sistema proposto, demonstrando a sua generalidade.

Finalmente, no capítulo 6 são apresentadas as conclusões deste trabalho e sugestões para trabalhos futuros.

Capítulo 2

Simulação Adaptativa em Mecânica Computacional

Este capítulo tem por objetivo caracterizar um ambiente de simulação adaptativa em mecânica computacional baseada no método dos elementos finitos. São discutidas as principais características deste tipo de simulação, dentro do contexto da linha de pesquisa onde esta dissertação está inserida [Cavalcanti 1992, Guimarães 1992, Cavalcante 1994] de forma a se poder caracterizar melhor as contribuições deste trabalho. É descrito um sistema automático para simulação adaptativa de problemas de elasticidade plana, no qual as principais características são a modelagem baseada na descrição geométrica completa da estrutura e a estratégia de programação orientada a objetos.

Um sistema para simulação adaptativa de um problema através do método dos elementos finitos é considerado automático quando o processo é dirigido pelo usuário através de uma interface gráfica amigável e a geração e refinamento da malha de elementos finitos ocorre com um mínimo de intervenção do analista.

O processo de simulação pode ser dividido em algumas etapas conforme mostrado na Fig. 2.1. Este processo é iniciado com a especificação da geometria e dos atributos do modelo. O modelo geométrico é formado por componentes com geometria bem definida e por relações de conectividade entre estes componentes (topologia). Além da geometria e da topologia são necessárias ainda informações adicionais para a descrição completa do problema físico a ser analisado. Essas informações são referenciadas no decorrer deste trabalho como atributos do modelo ou da simulação. Grande parte destes atributos é aplicada diretamente sobre as entidades do modelo geométrico criado. Este procedimento de associar atributos diretamente às entidades geométricas do modelo caracteriza uma *modelagem baseada em geometria* [Shephard e Finnigam 1988]. Este tipo de abordagem

oferece algumas vantagens, entre as quais um melhor suporte para a automação completa do processo e a simplificação da geração do modelo para a simulação. Essas vantagens ficarão evidentes ao longo do trabalho e a medida que as fases subseqüentes do processo, como a geração automática da malha e análise adaptativa, forem devidamente caracterizadas.

A partir da definição da geometria, o processo continua com a geração automática da discretização do modelo geométrico, resultando na criação dos nós e elementos que, associados aos seus respectivos atributos, compõem o modelo numérico a ser analisado. Em seguida, é feito o mapeamento dos atributos associados ao modelo geométrico para a malha gerada. Este mapeamento é feito automaticamente, onde as entidades da discretização (nós e elementos) herdam os atributos associados ao modelo geométrico [Shephard-Finnigam 1988, Finnigam *et al.* 1989].

Gerado o modelo discretizado, procede-se então com a análise numérica do problema, onde são processados os dados para a obtenção dos resultados da simulação. De posse dos resultados, faz-se uma verificação dos valores obtidos, e caso estes não sejam satisfatórios, refina-se a malha utilizada, onde mais uma vez os atributos serão herdados do modelo geométrico pela nova discretização, gerando um novo modelo numérico para que seja efetuada nova análise. Continuando-se neste ciclo até que um determinado critério de convergência seja atendido. A estratégia de refinamento pode envolver uma subdivisão dos elementos (refinamento do tipo "h"), um aumento no poder de interpolação nos elementos (refinamento do tipo "p"), uma relocação dos nós (refinamento do tipo "r"), ou uma combinação destas técnicas [Cook 1989].

As etapas envolvidas em um sistema automático para simulação são bem definidas e para cada uma delas existem diversas tecnologias a serem aplicadas. Um exemplo disto são os diversos algoritmos existentes para a geração automática da malha.

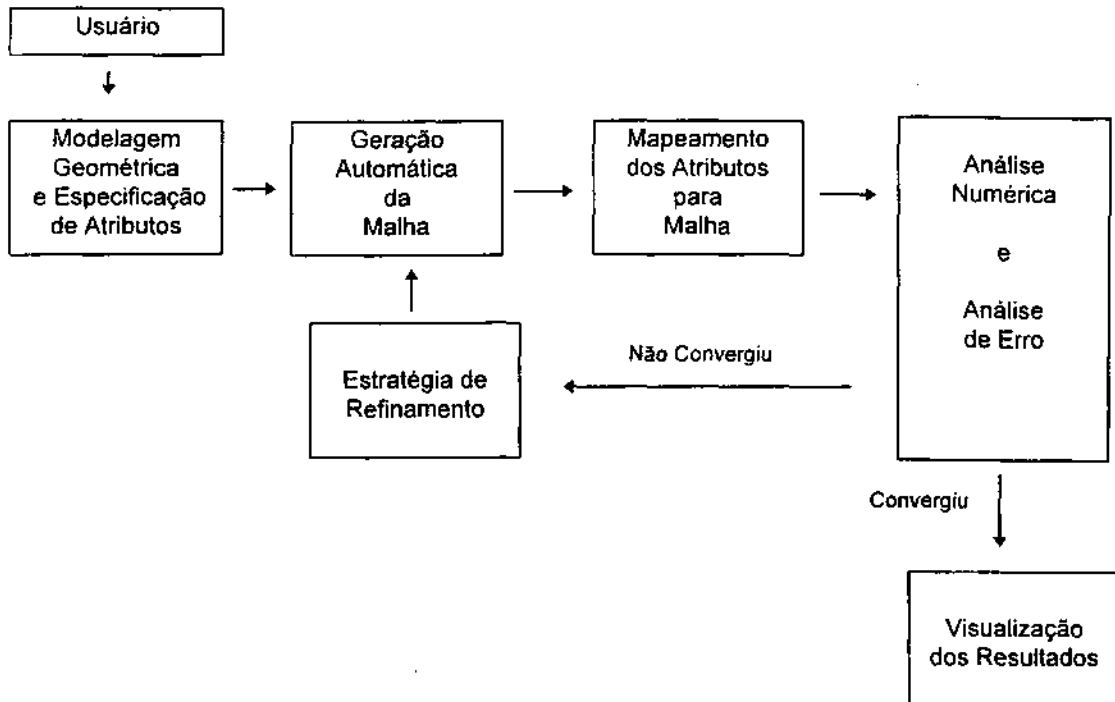


Figura 2.1- Procedimento genérico para simulação adaptativa baseada em geometria.

2.1 Sistema adaptativo baseado em modelagem geométrica para simulação de problemas de elasticidade plana

Conforme mencionado anteriormente, a título de ilustrar o processo descrito na introdução deste capítulo será apresentado e discutido um sistema para simulação adaptativa bidimensional [Cavalcante 1994] resultado de diversos trabalhos desenvolvidos nos últimos anos nesta linha de pesquisa [Cavalcanti 1992, Guimarães 1992, Cavalcante 1994]. A Fig. 2.2 mostra cada um destes trabalhos indicando suas respectivas funcionalidades dentro do sistema.

O modelador EDP (Editor de Diagramas Planares) [Cavalcanti 1992] é responsável pela geração do modelo geométrico e aplicação dos atributos associados a este modelo (Fig. 2.3). As informações sobre o modelo geométrico gerado são gravadas em um arquivo que é lido pelo programa AMVIEW (*Adaptive Mesh View*) [Cavalcante 1994] responsável

pela geração automática da malha inicial, o que é feito a partir da geometria e de um parâmetro de controle dado. Gera-se assim o modelo numérico inicial a ser analisado (Fig. 2.4). Em seguida, o modelo é analisado pelo programa FEMOOP (*Finite Element Method Oriented Object Programming*) [Guimarães 1992], que gera um arquivo contendo os resultados obtidos na análise. A partir destes resultados, o AMVIEW verifica a convergência do processo. Caso os resultados não tenham convergido, a malha é refinada gerando um novo modelo numérico (Fig. 2.5) que será analisado pelo FEMOOP. Este ciclo continua até que os resultados obtidos sejam aceitáveis, segundo um critério de erro numérico definido pelo usuário. O AMVIEW permite ainda que os resultados obtidos em cada análise realizada sejam visualizados graficamente (Fig. 2.6).

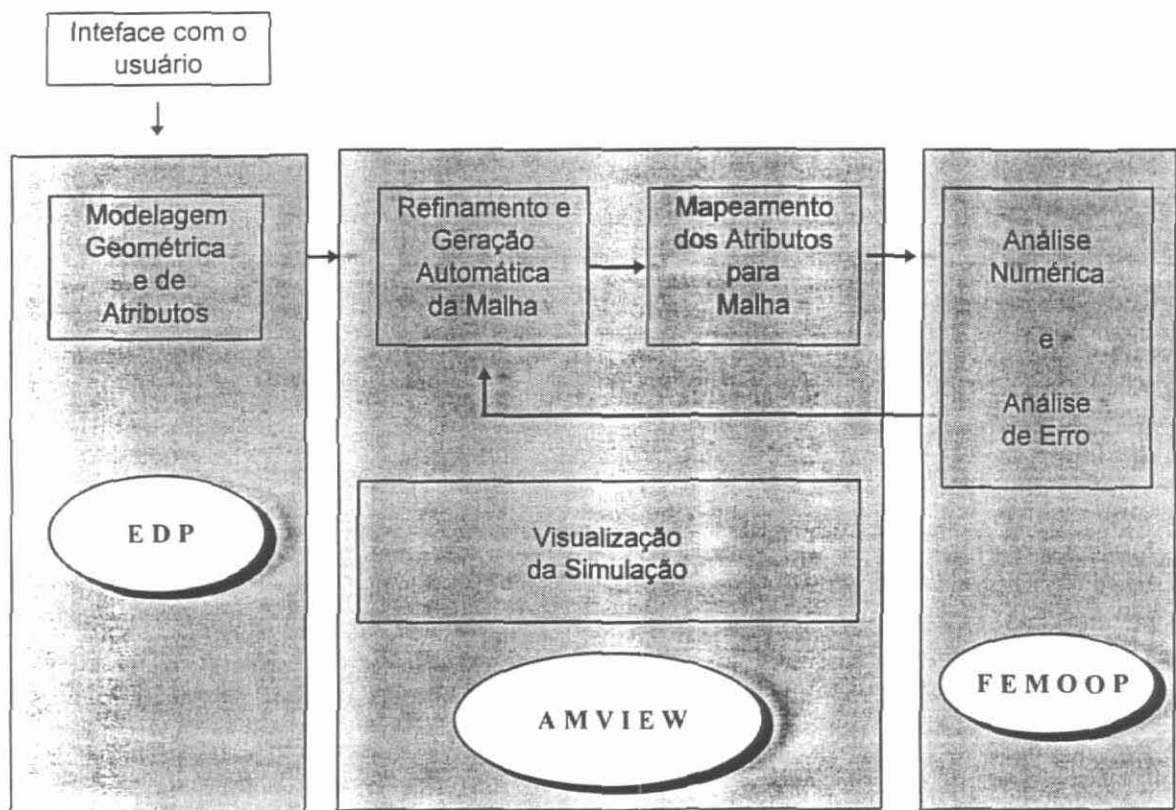


Figura 2.2 - Módulos e funcionamento do sistema apresentado.

Nas seções que se seguem serão discutidas as etapas do processo baseadas no sistema apresentado, indicando ainda as tecnologias que foram utilizadas para a implementação de cada um dos trabalhos envolvidos.

2.2 Modelagem geométrica e especificação de atributos

Conforme mostrado na Fig. 2.2, o módulo EDP é responsável pela geração do modelo geométrico e pela aplicação dos atributos associados a cada entidade geométrica do modelo. A Fig. 2.3 ilustra a interface gráfica do EDP, onde aparece um modelo geométrico construído e a especificação de um atributo do tipo material.

2.2.1 Modelagem geométrica

A descrição da geometria é uma das fases mais importantes no processo de automação, visto que tem uma participação direta em quase todos os aspectos que envolvem a modelagem para a simulação numérica de um problema físico.

Modelagem geométrica pode ser definida como a área que lida com o problema de criar, em computador, representações para objetos geométricos e de construir algoritmos que processam estas representações [Cavalcanti 1992].

O modelador geométrico utilizado pelo EDP consiste basicamente de um gerenciador de subdivisões do espaço bidimensional e busca facilitar ao usuário a criação e manutenção da geometria de um modelo, da forma mais natural possível. Para a implementação deste modelador utilizou-se a biblioteca HED (*Half-Edge Data Structure*)[Martha 1993], que consiste na implementação da estrutura de dados topológica *half-edge* [Mäntylä 1988].

A biblioteca HED faz o gerenciamento genérico de subdivisões planares, permitindo ao programador da aplicação o acesso a uma estrutura de dados relativamente complexa com

um alto nível de abstração, escondendo deste as complexidades envolvidas em um tratamento topológico e geométrico sofisticados. Isto é feito através de um conjunto de funções oferecidas pelo HED, como, por exemplo, funções para criação e destruição de um modelo, inserção de um vértice ou uma curva, redefinição de atributos associados as entidades geométricas, realização de *undos* e *redos*, entre outras.

Esta biblioteca apresenta algumas características que facilitam bastante o desenvolvimento de uma plataforma para a geração de um modelo geométrico, tais como:

- (a) Tratamento interno de consistência do modelo criado.
- (b) Relações de adjacências entre as entidades do modelo (vértices, curvas e regiões), são reconhecidas automaticamente e obtidas de uma forma bastante eficiente.
- (c) Criação automática das entidades que são geradas a partir da inserção de outras entidades no modelo.

O processo de criação da geometria dentro deste sistema é bastante simples e eficiente, oferecendo ao usuário muitos recursos para a geração da geometria do modelo a ser analisado [Ferraz 1993].

2.2.2 Especificação dos atributos

A biblioteca HED, além dos recursos citados, possui ainda um mecanismo para a associação dos atributos às entidades geométricas. Como esta biblioteca é independente da aplicação *cliente* e cada uma possui seus atributos específicos, o HED internamente possui um campo em cada entidade da estrutura de dados (vértice, aresta, face e sólido) que é reservado para que o cliente associe atributos a estas entidades. Como a biblioteca desconhece a natureza destes atributos, cabe à aplicação desenvolver funções específicas para manipulá-los. Desta forma o módulo responsável pela geração da geometria permite também que o usuário possa aplicar os atributos relativos a cada entidade geométrica criada.

Nesta versão do sistema são considerados atributos relativos ao problema de elasticidade plana, como, por exemplo, condições de suporte, carga concentrada, carga distribuída, deslocamentos prescritos, etc. Para a definição de um tipo de problema diferente é necessária a inclusão de novos tipos de atributos, como fluxo de calor e temperatura para uma análise térmica, por exemplo. A inserção de novos tipos de atributos requer uma modificação no código da aplicação para adição de outros algoritmos para o tratamento destes novos atributos. Para isto é necessário um conhecimento mais detalhado do programa, de sua estrutura de dados e da biblioteca HED.

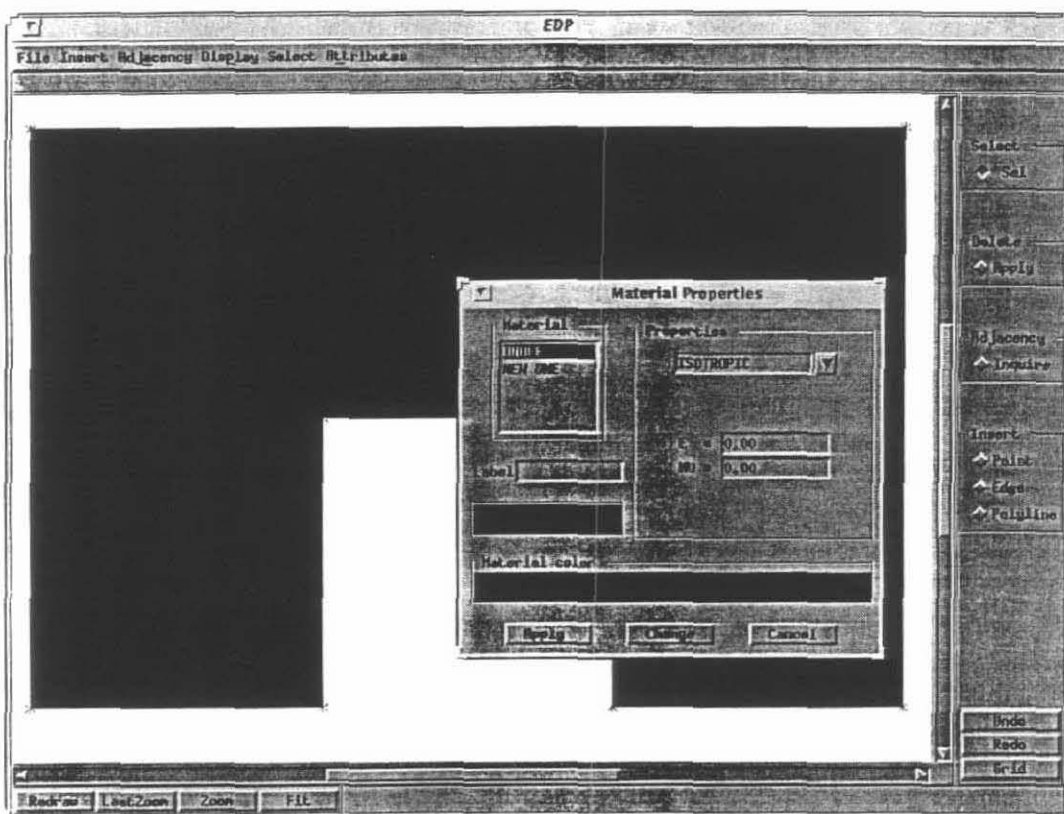


Figura 2.3 - Modelagem geométrica e especificação dos atributos.

2.3 Geração automática da malha inicial

Com a descrição completa da geometria e dos atributos associados, inicia-se então a fase seguinte do processo que é a geração automática da malha inicial.

Dentro do contexto da automação em uma análise por elementos finitos, a geração automática da malha é uma tecnologia imprescindível ao sistema. Entende-se por geração automática de uma malha de elementos finitos, a capacidade de geração de um modelo válido de elementos finitos a partir de um domínio arbitrário. No caso do sistema apresentado, a definição do domínio que é passada para o algoritmo de geração da malha é o modelo geométrico construído na primeira fase do processo.

O algoritmo utilizado pelo AMVIEW foi desenvolvido por Cavalcante (1994). A idéia do algoritmo consiste na combinação das técnicas de enumeração espacial recursiva (*quadtree*) [Samet 1984], para a geração dos elementos no interior da região, e triangulação de Delaunay [Preparata 1985], para a geração dos elementos entre o contorno da região e a malha gerada por *quadtree*. Um aspecto fundamental deste algoritmo é a utilização das informações de adjacência da estrutura *quadtree* também para o processo de triangulação de Delaunay.

A geração da malha inicial (Fig. 2.4) é feita a partir do modelo geométrico criado e é baseada em um parâmetro de controle fornecido pelo usuário. Esse parâmetro de controle é um valor que define a discretização do contorno da geometria do modelo. A necessidade de uma discretização prévia do contorno surge porque o algoritmo de *quadtree* se baseia nesta discretização para a geração da malha.

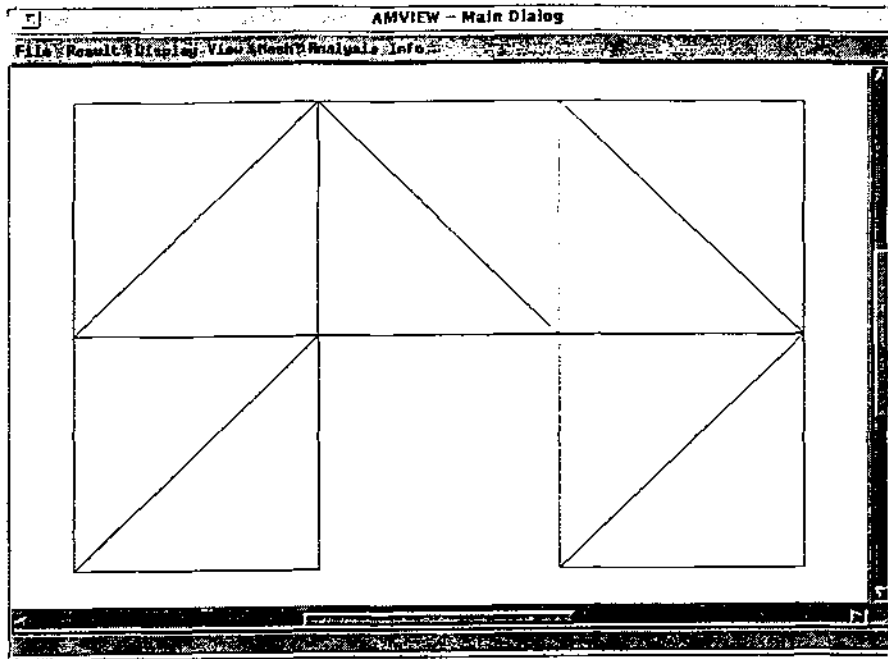


Figura 2.4 - Geração automática da malha inicial.

2.4 Mapeamento dos atributos para a malha

Nesta fase fica evidente uma das vantagens na adoção de uma modelagem baseada em geometria, conforme citado anteriormente, onde o mapeamento dos atributos do modelo geométrico para a discretização (nós e elementos) é feito automaticamente através da herança destes atributos. Ou seja, a discretização herda, automaticamente, os atributos do modelo geométrico. Os atributos no problema real são associados às características físicas do modelo e não a abstrações matemáticas, como são nós e elementos de uma malha.

Dentro do sistema apresentado foi utilizada uma estrutura de dados relativamente complexa para gerenciar todo o processo de geração adaptativa. Esta estrutura de dados, denominada HED, além de guardar informações com as descrições dos modelos geométrico (vértices, curvas e regiões) e numérico (nós e elementos), armazena também todas as informações relativas aos atributos associados a cada entidade geométrica. Essas informações são utilizadas para que a herança dos atributos pelo modelo numérico seja

feita de forma automática e consistente toda vez que uma nova malha é gerada. A estrutura de dados por este módulo utilizada não é a mesma que foi utilizada pelo módulo de modelagem geométrica.

O mapeamento dos atributos do modelo geométrico para a discretização considera quatro casos [Calvalcante 1994]: atributos aplicados em vértices (nível geométrico) que serão herdados automaticamente pelos nós (nível de malha), atributos aplicados sobre as curvas que serão herdados pelos nós que estão sobre estas, atributos sobre as curvas que vão para os lados dos elementos adjacentes às curvas e, finalmente, atributos que são aplicados às regiões que são herdados pelos elementos que pertencem a estas regiões.

Os atributos que são aplicados diretamente sobre vértices geométricos são condições de suportes, deslocamentos prescritos e forças concentradas por exemplo. Vértices geométricos correspondem aos pontos extremos de curvas e sobre estes vértices estão nós que serão sempre mantidos na estratégia de refinamento utilizada.

Os nós do segundo caso citado acima são gerados sobre curvas geométricas do modelo, cada vez que uma nova malha é gerada. Os atributos aplicados sobre as curvas são herdados pelos nós que estão sobre as mesmas. Neste caso, através de consultas a uma lista de nós que estão sobre uma curva, o módulo adaptativo transfere automaticamente e de forma consistente os atributos da curva para os nós correspondentes.

Para o terceiro caso, onde os atributos aplicados às curvas são mapeados para os lados dos elementos adjacentes a elas, o tratamento dos atributos é feito de forma semelhante. Percorre-se uma lista de elementos adjacentes existente na estrutura de dados da curva e aplica-se os atributos da curva no lado do elemento correspondente.

Um procedimento muito parecido é utilizado para o último caso, onde os elementos herdam os atributos associados às regiões em que eles estão inseridos. Neste caso é

percorrida uma lista de elementos contida na estrutura de dados da região, e cada elemento herda os atributos aplicados a esta região.

2.5 Análise numérica

Gerado o modelo de elementos finitos, o sistema está preparado para a fase de análise numérica do problema. O módulo de análise por elementos finitos usado no sistema foi desenvolvido sob uma filosofia de programação orientada a objetos em trabalho anterior desta linha de pesquisa [Guimarães 1992]. Inicialmente este módulo foi implementado em linguagem C, que não é uma linguagem formal para a utilização deste tipo de programação. Posteriormente, este módulo foi inteiramente traduzido para a linguagem de programação C++, o que permite um melhor entendimento do código e uma maior facilidade na expansão do programa. O autor desta dissertação foi um dos participantes da conversão deste módulo para a linguagem C++.

O programa FEMOOP (*Finite Element Method Object Oriented Program*) inicialmente tratava apenas alguns tipos de análise, como estado plano de tensões e estado plano de deformações. A utilização do paradigma de programação orientada a objetos facilitou a rápida expansão do programa, onde foram implementados novos tipos de análise, como análise de cascas, placas, análise térmica, entre outras. Além disso, foi feita a expansão do módulo que faz a análise de erro de discretização, utilizada para adaptatividade da simulação, desenvolvida por Cavalcante (1994). Este módulo inicialmente tratava apenas problemas de elasticidade plana e para este trabalho foi necessária a sua expansão para tratar também problemas térmicos. Isto foi facilmente implementado, sem alteração da estrutura anterior do programa.

2.6 Refinamento das fronteiras e geração da nova malha

A estratégia de geração de malha do sistema AMVIEW tem como uma das principais características o refinamento *a priori* das curvas do modelo geométrico. Isto resulta em uma discretização mais regular do domínio próximo ao contorno do modelo [Cavalcante 1994].

Para o refinamento das fronteiras das regiões (Fig. 2.5) foi utilizado um algoritmo semelhante ao algoritmo de *quadtree* utilizado para o refinamento do domínio, sendo que na versão unidimensional. Neste caso, o refinamento de cada curva na fronteira utiliza uma técnica de enumeração espacial recursiva baseada em uma estrutura de dados de árvore binária (*binary tree*). Como o algoritmo de *quadtree* concebido se baseia na discretização do contorno para refinar o domínio, o refinamento das fronteiras é quem controla a geração da *quadtree* que será utilizada no processo de geração automática de malha utilizado.

Este refinamento consiste numa discretização das curvas das fronteiras em função do tamanho característico de cada elemento da malha adjacente a estas curvas. Esses tamanhos característicos são calculados baseados na estimativa de erro de cada elemento. Uma descrição mais detalhada sobre o refinamento das fronteiras pode ser encontrado na referência [Cavalcante 1994].

Conforme mencionado anteriormente, a malha no interior do domínio (Fig. 2.5) é gerada pela combinação das técnicas de *quadtree* e de triangulação de Delaunay [Cavalcante 1994].

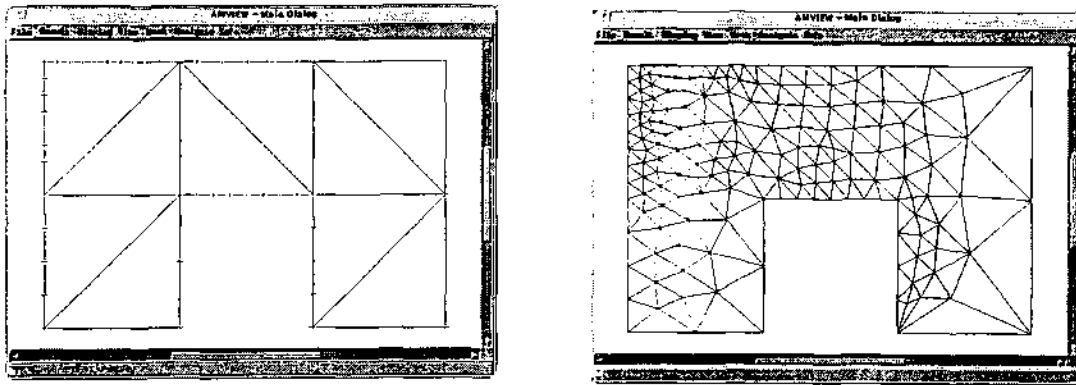


Figura 2.5 - Refinamento das fronteiras e geração da nova malha.

2.7 Visualização dos resultados

Nesta etapa o usuário tem acesso a informações qualitativas e quantitativas do modelo numérico e dos resultados obtidos na análise, em cada etapa ou ao final do processo adaptativo. A resposta quantitativa é feita através de funções de consulta, que fornecem informações como o número de nós da malha, ou de gráficos de resultados ao longo de linhas arbitrárias definidas interativamente sobre o modelo. A resposta qualitativa consiste basicamente na representação gráfica de resultados escalares (Fig. 2.6), representados nos pontos de Gauss ou nos pontos nodais, e de resultados vetoriais, obtidos por valores referentes às componentes de vetores. Os resultados vetoriais podem ser os deslocamentos nodais ou os vetores das tensões principais nos pontos de Gauss. Os resultados tensoriais (tensões ou deformações, por exemplo) são mostrados através da representação escalar dos seus componentes.

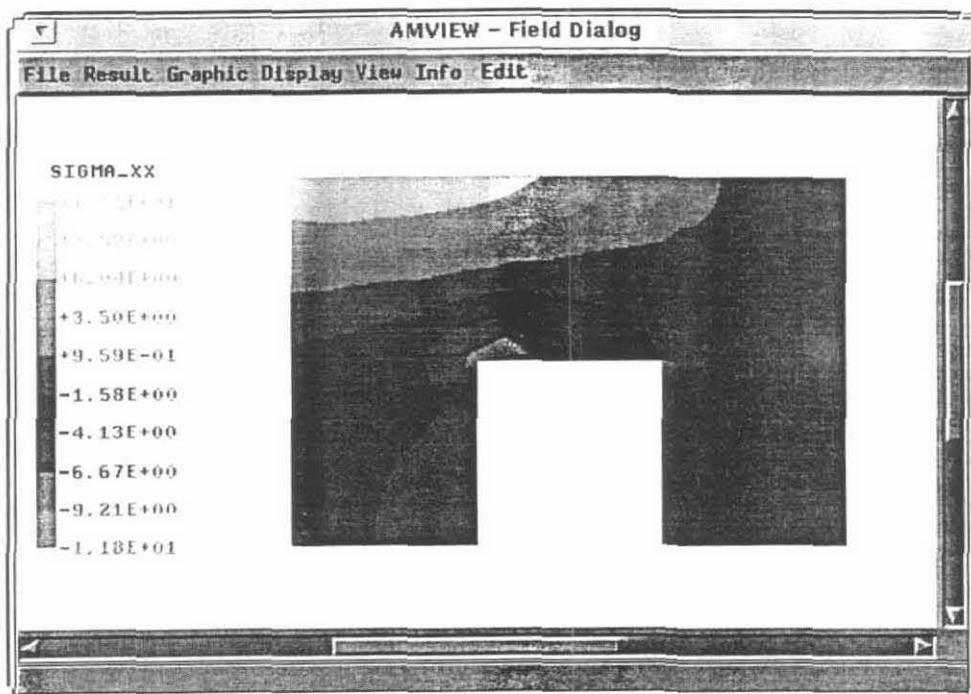


Figura 2.6 - Visualização dos resultados.

Capítulo 3

Um Sistema Extensível para Configuração de Atributos de Mecânica Computacional

Dentro de um processo de simulação de mecânica computacional utilizando o método dos elementos finitos existem algumas etapas que não dependem do tipo de problema a ser analisado. Exemplos disso são a modelagem geométrica e a discretização do modelo. Uma determinada discretização, por exemplo, pode ser utilizada para diferentes análises. O que determina o tipo de análise a ser efetuada são os atributos associados ao modelo.

Geralmente, os sistemas utilizados para simulações em mecânica computacional são muito grandes e genéricos, dificultando o seu uso e manutenção, ou são específicos para uma aplicação em particular. Neste segundo caso, é difícil a sua utilização em novas aplicações, pois adicionar novas funcionalidades para atender a diferentes tipos de simulações requer, normalmente, um conhecimento detalhado do código implementado.

Por outro lado, o conhecimento adicional necessário para o usuário final utilizar um sistema em novos tipos de simulações se refere somente à especificação de novos tipos de atributos, já que outros aspectos, como a modelagem geométrica, a geração da malha e a visualização dos resultados, independem do tipo de análise.

Neste sentido, Carvalho (1995) propôs uma arquitetura para um ambiente de configuração de atributos para simulações em mecânica computacional. Esta arquitetura adota a abordagem de modelagem baseada em geometria, tal como descrito no capítulo anterior, e tem como principal objetivo dar ao usuário a capacidade de estender um sistema para a simulação em mecânica computacional para diferentes tipos de análise. A configuração de atributos é feita através de um arquivo que contém instruções em uma linguagem simples. Como esta linguagem de

configuração é interpretada, ou seja, compilada em tempo de execução, a configuração é feita sem necessidade de recompilar e religar o sistema.

Uma parte importante desta dissertação consiste no desenvolvimento de um sistema que implementa a arquitetura proposta por Carvalho. Este sistema é denominado ESAM (*Extensible System of Attribute Management*) e seu desenvolvimento será detalhado ao longo deste capítulo. Por uma questão de completude deste trabalho serão repetidos aqui alguns conceitos apresentados por Carvalho. A Fig. 3.1 ilustra o sistema ESAM sendo utilizado em uma aplicação.

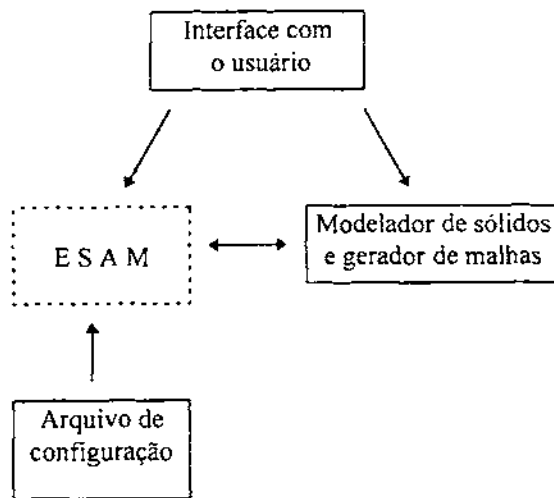


Figura 3.1 - Aplicação que utiliza o ESAM.

3.1 Arquitetura do ambiente de configuração

O ambiente proposto por Carvalho não se destina a tratar de detalhes referentes à modelagem geométrica ou à geração da malha, mas sim dos aspectos referentes à especificação dos atributos de simulação. Assume-se que os serviços de modelagem são fornecidos por algum modelador.

Seguindo a abordagem de modelagem baseada em geometria, os atributos são associados às entidades geométricas do modelo. De modo a permitir que a especificação dos atributos seja feita de forma independente do modelador

geométrico, a comunicação entre o gerenciador de atributos e os serviços de modelagem deve ser feita através de um mecanismo que mantenha a independência dos módulos envolvidos na tarefa. Ou seja, o módulo responsável pela definição dos atributos não deve ter conhecimento sobre as características específicas do modelador geométrico, assim como o modelador não deve saber da parte referente à especificação de atributos.

Neste sentido, a arquitetura proposta divide a aplicação em três partes. Uma parte representa os serviços ou tecnologia da aplicação, composta pelos aspectos que são independentes do tipo de análise, como modelagem geométrica, a geração da malha e a visualização dos resultados. Uma outra parte refere-se aos aspectos específicos a uma aplicação particular, ou seja, a definição dos atributos. A terceira parte, refere-se a um sistema responsável pela comunicação entre as duas partes anteriores. Esse sistema permite o acesso programável à tecnologia da aplicação com um alto grau de abstração.

As tarefas referentes à configuração dos atributos são feitas através de uma linguagem acoplada e interpretada. Conforme mencionado anteriormente, isto permite que a configuração dos atributos seja feita em tempo de execução, isto é, sem a necessidade de recompilar e religar o sistema. Essa linguagem será referida ao longo deste trabalho como *linguagem de configuração*. A linguagem utilizada para implementação dos serviços de modelagem será referida como *linguagem de programação*.

Uma aplicação construída com a arquitetura proposta consiste de (Veja Fig. 3.2):

- Um módulo para modelagem de sólidos com suporte para mecânica computacional. Este módulo representa os serviços da aplicação e é responsável pela geração do modelo geométrico, pela discretização deste modelo e pela visualização dos resultados. Deve-se entender por modelador, nesse contexto, a capacidade de descrição da geometria, geração da malha, análise numérica e visualização do modelo e de seus resultados.
- Uma linguagem para a configuração, como Tcl [Ousterhout 1994] ou Lua [Jerusalimschy *et al.* 1994, Figueiredo *et al.* 1994], estendida com mecanismos

para suportar a definição de classes, no contexto de programação orientada a objetos. A configuração da aplicação é feita através da redefinição dos métodos das classes existentes e da definição e implementação de novas classes, com seus respectivos métodos. O conjunto de instruções utilizando a linguagem de configuração constitui um módulo que é referido como a parte configurável da aplicação.

- Um *toolkit*, como Tk [Ousterhout 1994] ou EDG [Celes 1995, Celes *et al.* 1995], que oferece uma interface gráfica para acessar e manipular os dados das classes definidas.
- Uma coleção de classes básicas que definem, com alto grau de abstração, as entidades normalmente envolvidas no processo de geração dos modelos geométricos e numéricos e que sejam comuns a um grande domínio de simulações (*Material, Boundary Conditions, Edge, Vertex, Element, Node, etc.*). Estas classes são definidas como *Core classes*.
- Um conjunto de funções que são responsáveis pelo acesso à base de dados do modelador. Qualquer consulta a esta base de dados é feita através destas funções, que são chamadas de funções MIS (*Modeling Interface Specification*). Deste modo é criada uma camada de abstração sobre a base de dados, que esconde os detalhes de como as informações requisitadas são obtidas.
- Um conjunto de funções que são responsáveis pela ligação (*binding*) entre as *Core classes* e as funções mencionadas acima.

A Fig. 3.2 mostra uma aplicação desenvolvida com a arquitetura proposta, onde os retângulos com linhas tracejadas correspondem aos módulos que não fazem parte do sistema ESAM, ou seja, o módulo referente aos serviços da aplicação (modelador) e o arquivo de configuração dos atributos. Neste arquivo é feita a especificação de novos tipos de atributos para uma determinada simulação, onde podem ser redefinidos os métodos das classes existentes e criadas novas classes de atributos.

Os retângulos com linhas cheias na Fig. 3.2 representam o sistema proposto, composto dos módulos que permitem o acesso programável à tecnologia da aplicação (modelador). Nas seções seguintes serão discutidos detalhes sobre a funcionalidade e

o desenvolvimento destes módulos. Também são descritas as ferramentas utilizadas para o desenvolvimento do sistema ESAM.

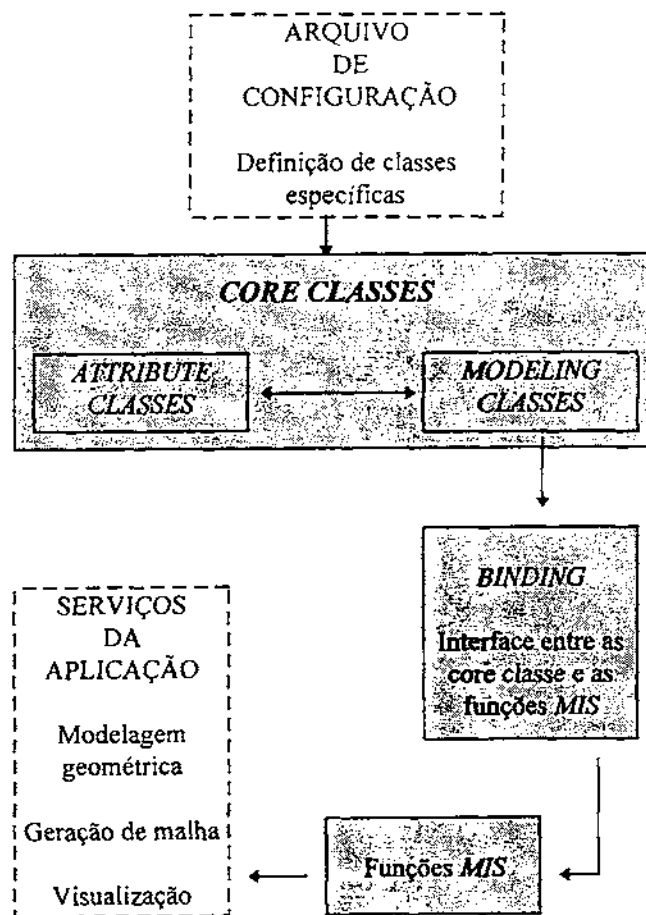


Figura 3.2 - Arquitetura do sistema ESAM.

3.2 Descrição do sistema

O ESAM consiste de uma coleção de classes que representa a biblioteca básica da parte configurável da aplicação (*Core classes*) e de um conjunto de funções responsáveis pela interface entre as *Core classes* e o modelador. O ESAM possui ainda um conjunto de funções que oferecem os serviços para a configuração da aplicação.

Os serviços do sistema são oferecidos através de um conjunto de rotinas que devem ser incorporadas à aplicação. Estes serviços representam a funcionalidade do sistema, como a possibilidade de especificação de um novo atributo ou associação deste atributo a uma entidade geométrica. A aplicação programa sua interface para que os usuários tenham acesso aos serviços do ESAM. Os serviços, entretanto, não incorporam a semântica da aplicação. Este capítulo mostrará algumas das funcionalidades destes serviços, bem como uma descrição detalhada de cada um deles.

As *Core classes*, por sua vez, representam uma abstração das entidades envolvidas no processo de geração de um modelo de elementos finitos, ou seja, representam as entidades geométricas e da malha (*Modeling classes*) e os atributos de simulação (*Attribute classes*).

As classes *Modeling* permitem que o usuário defina métodos correspondentes às ações sobre as entidades geométricas, tais como a aplicação de um atributo a uma região por exemplo. Desse modo, o configurador da aplicação pode controlar o comportamento das entidades geométricas com um alto grau de abstração, pois não precisa acessar o código que implementa estas entidades no modelador.

As classes derivadas da classe *Attribute* compõem uma biblioteca que representa uma base para um grande domínio de simulações. Esta biblioteca pode ser expandida através da construção de novas classes, derivadas das classes existentes, de acordo com a necessidade de cada simulação específica.

O acesso à hierarquia das classes existentes é feito através de uma ferramenta que permite ao usuário percorrer toda a estruturação de classes do sistema, com suas respectivas variáveis e métodos. Esta ferramenta, denominada *Browser*, permite ainda a construção de novas classes derivadas de classes de atributos, ou a redefinição de métodos de classes de geometria. O acesso ao arquivo de configuração, pelo usuário, pode ser feito inteiramente através do *Browser*.

3.3 Ferramentas utilizadas

De acordo com a arquitetura proposta, para a implementação do sistema são necessárias algumas ferramentas especiais, como uma linguagem de configuração e um *toolkit* que ofereça uma interface gráfica para acesso aos objetos definidos. Para o desenvolvimento do sistema apresentado foram utilizadas a linguagem de configuração Lua [Jerusalimschy *et al.* 1994, Figueiredo *et al.* 1994] e o *toolkit* EDG [Celes 1995]. Nesta seção serão mostrados alguns detalhes de cada uma destas ferramentas.

3.3.1 Linguagem de configuração - Lua

Lua é uma linguagem de extensão projetada para ser usada como linguagem de configuração, acoplada a um programa hospedeiro (escrito em C). As aplicações em geral podem acoplar códigos escritos em Lua, permitindo uma prototipagem rápida e acesso programável pelo usuário à tecnologia implementada pela aplicação.

A linguagem depende de um programa hospedeiro, que pode chamar funções escritas em Lua para serem executadas, ler e atribuir variáveis em Lua, e ainda registrar novas funções escritas em C para serem chamadas por funções escritas em Lua. Além disso Lua provê as construções fundamentais para definição de funções e controle de fluxo (*if-elseif-else-end*, *while-do-end*, *repeat-until*). A linguagem Lua também suporta o paradigma de programação orientada a objetos, permitindo a definição de métodos e herança entre classes.

Todos os mecanismos e recursos citados acima são facilmente acessados através de uma sintaxe muito simples, semelhante à da linguagem Pascal. A simplicidade é uma característica fundamental apresentada pela linguagem, que facilitou bastante o desenvolvimento do sistema ESAM e tornou-o simples de ser utilizado. Lua oferece ao programador algumas outras facilidades. Por exemplo em Lua o programador não precisa se preocupar com declaração de variáveis e o gerenciamento de memória.

Maiores detalhes sobre a linguagem podem ser encontrados nas referências [Jerusalimschy *et al.* 1994, Figueiredo *et al.* 1994].

3.3.2 O *toolkit* EDG

O *toolkit* EDG (Entrada de Dados Gráfica) é uma ferramenta para o desenvolvimento de aplicações para a captura de dados, permitindo a associação e manipulação de desenhos. O EDG é composto basicamente por dois conjuntos de objetos: objetos de interface e objetos gráficos. Os objetos de interface implementam abstrações, através de uma sintaxe simples, sobre objetos do sistema de interface com o usuário IUP [Levy 1993]. O EDG permite que o usuário crie diálogos com alto grau de interatividade a partir da composição de objetos simples. Com estes recursos podem ser construídos facilmente diálogos para a captura de dados.

O sistema EDG provê ainda recursos para a instanciação e manipulação de representações gráficas. Para o desenvolvimento do sistema apresentado neste trabalho não foram utilizados estes recursos. Utilizou-se apenas os recursos oferecidos para a construção de diálogos para toda parte de captura de dados do sistema.

O configurador de uma aplicação que utiliza o *toolkit* EDG faz a configuração através da linguagem Lua. Desta forma, é possível a criação de diálogos para a captura dos dados relativos aos atributos especificados pelo configurador.

3.4 *Core classes*

Esta seção descreve a funcionalidade e a implementação das *Core classes*, mencionadas nas seções anteriores.

As *Core classes* formam uma biblioteca de classes que representam as grandezas comuns envolvidas em uma simulação de mecânica computacional baseada em geometria. Estas classes representam uma abstração, no contexto do ambiente de

configuração, dos modelos geométrico e numérico, das entidades referentes a estes modelos e dos seus atributos. A hierarquia destas classes, mostrada na Fig.3.3, é organizada de modo consistente com a arquitetura proposta, ou seja, as classes referentes aos atributos (*Attribute classes*) são definidas independentemente das classes referentes aos modelos geométrico e numérico (*Modeling classes*).

A associação entre os atributos e as entidades geométricas é armazenada pelo ambiente de configuração, independentemente do modelador. O modelador não armazena e nem tem acesso aos objetos referentes aos atributos.

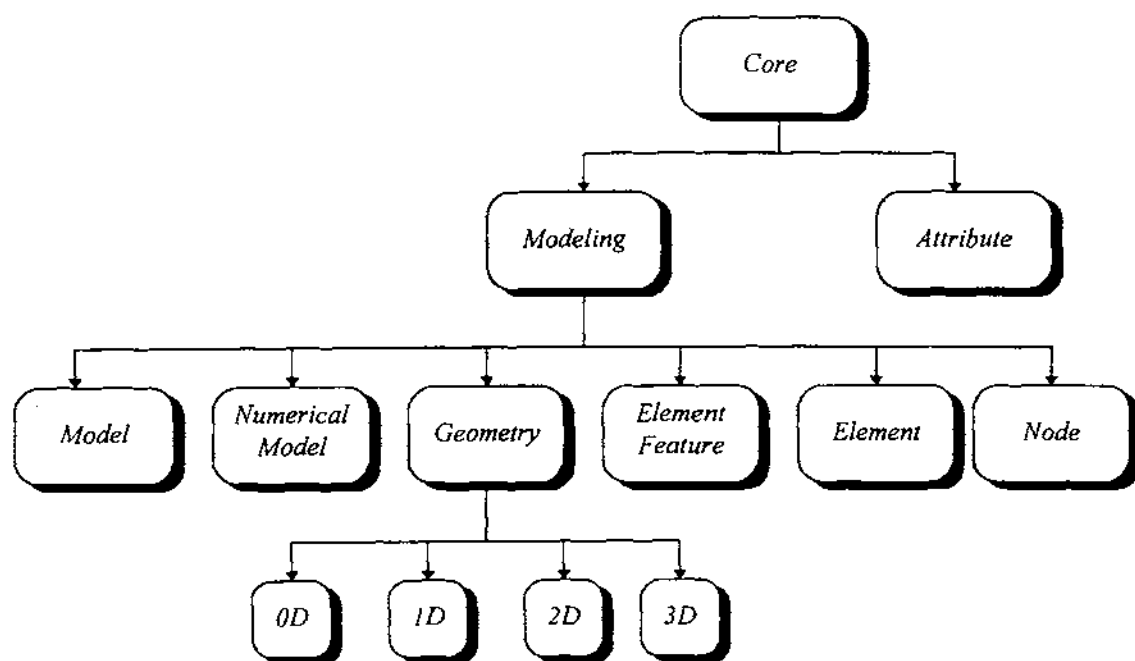


Figura 3.3 - Organização das Core classes.

3.4.1 *Attribute classes*

A classe *Attribute* representa a superclasse dos atributos da simulação. Subclasses derivadas desta classe devem ser criadas para atender aos requisitos específicos de cada simulação, como, por exemplo, condições de contorno e materiais.

As instâncias destas classes contêm as propriedades que definem os tipos de atributos, assim como referências para os objetos que representam as entidades geométricas às quais os atributos estão associados (objetos das subclasses da classe *Geometry*). Estas instâncias são referenciadas como *objetos atributo*.

Os objetos atributo estão associados a um identificador. Esse identificador representa um rótulo (*label*) implementado através de uma cadeia de caracteres. O *label* de cada objeto pode ser definido explicitamente pelo usuário ou pode ser implicitamente construído pelo sistema. As novas classes são criadas em um arquivo de configuração, que é lido pela aplicação e que pode, como dito anteriormente, ser modificado sem a necessidade de recompilação do sistema. No apêndice A.1 é mostrada a definição desta classe, apresentando suas variáveis e métodos.

3.4.2 Modeling classes

A classe *Modeling* é a superclasse que representa as entidades abstratas envolvidas no processo de geração dos modelos geométrico e numérico de uma simulação. Desse modo, a hierarquia desta classe inclui as subclasses: *Model*, *Numerical Model*, *Geometry*, *Element*, *Node* e *Element Feature* (Fig. 3.3).

No processo de geração do modelo geométrico, os objetos da classe *Attribute* devem ser associados às entidades do modelador e, além disso, a especificação dos atributos pode requisitar informações do modelador. As classes *Modeling* são responsáveis pela interface da comunicação entre o ambiente de configuração e o modelador. Toda ação referente à configuração dos atributos que requisitar informações do modelador deve ser feita através destas classes, que permitem, desse modo, o acesso programável às entidades existentes no modelador.

As classes *Model* e *Numerical Model* implementam as abstrações dos próprios modelos geométrico e numérico, e contêm as informações que devem existir nestes modelos. A classe *Model* contém uma coleção de objetos que representam as entidades geométricas do modelo e uma coleção dos objetos que representam os

atributos especificados (objetos das subclasses de *Geometry* e *Attribute*). A classe *Numerical Model* contém uma coleção de objetos que representam as entidades da discretização do modelo (objetos das subclasses de *Node* e *Element*).

3.4.2.1 Classe *Model*

Cada objeto da classe *Model* representa um modelo geométrico específico. Os métodos associados a esta classe implementam os serviços oferecidos pelo sistema referentes ao modelo geométrico, tais como serviços de especificação, de validação e de persistência de atributos do modelo geométrico.

Suponha, por exemplo, que o serviço para criação de um atributo tenha sido associado a um botão da interface de uma determinada aplicação. Ao ser acionado pelo usuário, este botão dispara o serviço, que por sua vez chama o método da classe *Model* para criar um atributo, que finalmente dispara um diálogo de interface mostrado na Fig. 3.4. A captura dos dados do atributo é feita utilizando-se o EDG.

A estruturação da classe *Model*, através de suas variáveis e métodos, é mostrada no apêndice A.2.

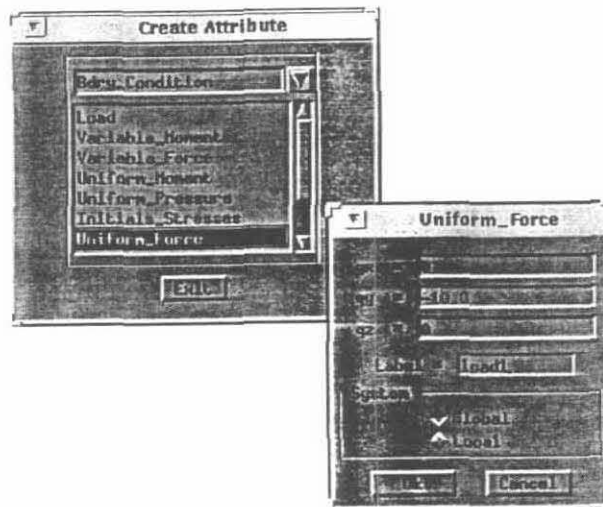


Figura 3.4 - Diálogo disparado pelo método *CreateAttribute* da classe *Model*.

3.4.2.2 Classe *Geometry*

Esta classe implementa a abstração das entidades geométricas para o sistema de configuração de atributos. A dimensão das entidades de um modelo geométrico são: 0 para um ponto, 1 para uma curva, 2 para uma superfície e 3 para um volume. Desse modo, existem classes (subclasses) da classe *Geometry* referentes a cada dimensão (Fig. 3.3).

As instâncias destas classes, referidas como *objetos geométricos*, contêm referências para as entidades geométricas do modelador e referências para os objetos atributo associados a estas entidades.

Um objeto geométrico, no contexto do ambiente de configuração, é criado somente quando se aplica um atributo a uma entidade geométrica. Desse modo, só existem objetos geométricos referentes às entidades que possuem atributos associados. A referência à entidade geométrica, armazenada pelo objeto geométrico, é feita através de um identificador, que pode ser qualquer valor que identifica a entidade no contexto do modelador.

A associação entre as entidades geométricas e os atributos é feita através de classes referentes a estas entidades, criadas automaticamente pelo sistema. Essas classes são específicas para cada modelador.

A classe *Geometry* implementa métodos para a manipulação dos atributos da entidade geométrica, como, por exemplo, adicionar e retirar um atributo da entidade, métodos para aplicação de atributos a entidade geométrica ou para consulta dos atributos aplicados a esta entidade.

Existem, ainda, métodos para validação de algumas operações sobre as entidades geométricas, como, por exemplo, métodos para validar a junção de entidades

geométricas (duas regiões podem apresentar materiais distintos que não podem ser combinados).

Esses métodos possuem uma implementação padrão (*default*) e são herdados por todas as subclasses de *Geometry*. Por exemplo, a implementação do método padrão *Attach* simplesmente adiciona o atributo corrente na lista de atributos do objeto em questão. Entretanto, o usuário configurador pode redefinir o método da subclasse *Vertex* para que, antes de adicionar o atributo na lista, verifique se já existe algum atributo do mesmo tipo associado ao vértice selecionado. É mostrado no apêndice A.3 a definição da classe com seus dados e métodos.

3.4.2.3 Classe *Numerical Model*

Esta classe representa uma abstração, no contexto do ambiente de configuração, do modelo numérico utilizado na simulação. Uma instância desta classe é criada para cada análise a ser efetuada e contém as informações necessárias à especificação do modelo numérico, ou seja, contém referências às entidades da discretização e um meio de mapear os atributos associados às entidades geométricas para as entidades da discretização.

Esta classe implementa os métodos para acesso aos atributos aplicados sobre o modelo numérico. Estes métodos retornam listas de todos os nós ou elementos que possuem um determinado tipo de atributo ou listas de todos os atributos de um determinado tipo aplicados sobre nós ou elementos. Estes métodos são usados, por exemplo, quando se deseja obter a lista de todos os elementos de uma malha que possuem material isotrópico ou todas as cargas aplicadas em nós. É mostrada no apêndice A.4 a estruturação desta classe.

3.4.2.4 Classe *Node*

Esta classe representa a abstração dos nós da discretização do modelo. É criado um objeto da classe para cada nó da malha. Um objeto desta classe referencia o objeto

geométrico correspondente à entidade geométrica, à qual o nó representado está associado. Por exemplo, se um nó da malha está sobre uma curva do modelo geométrico, o objeto da classe *Node* correspondente a esse nó referencia o objeto geométrico correspondente à curva.

De acordo com a estratégia de modelagem baseada em geometria, os atributos são associados somente aos objetos geométricos. Desse modo, para a geração do modelo numérico para análise, é necessário mapear os atributos dos objetos geométricos para os objetos da classe *Node*. Esse mapeamento é feito automaticamente pelo sistema ESAM, através da referência ao objeto geométrico.

É mostrado a seguir a definição desta classe, mostrando suas variáveis e métodos. Cada objeto da classe contém os campos *id*, que referencia o identificador do objeto (inicializado como *nil*, isto é, sem valor na sintaxe de Lua) e o campo *top*, que referencia o objeto geométrico correspondente ao nó. Esta classe implementa métodos para se obter as informações necessárias para a criação do modelo numérico para uma análise numérica (através de um arquivo, por exemplo). Para isso, existem métodos para manipulação dos dados relativos aos nós, dentre os quais estão: método para capturar as coordenadas de um nó, método para associar o objeto geométrico correspondente ao nó, método para acessar os atributos de um determinado tipo aplicado ao nó, entre outros apresentado no apêndice A.5.

3.4.2.5 Classe *Element*

Esta classe representa a abstração dos elementos da discretização do modelo. É criado um objeto da classe para cada elemento finito da malha.

Do mesmo modo como para os objetos da classe *Node*, é necessário se fazer o mapeamento dos atributos das entidades geométricas para os objetos da classe *Element*. O mecanismo para isso, no entanto, difere do utilizado pelos objetos da classe *Node*. Devido ao fato de no método dos elementos finitos ser possível a associação de atributos a partes de um elemento, como um lado de um elemento plano

ou uma face de um elemento sólido, não existe uma correspondência direta entre a entidade geométrica que contém o atributo e o elemento finito. Para se referenciar atributos associados a partes de elementos, foi criada uma classe (subclasse de *Modeling*) denominada *Element Feature*. Desse modo, cada objeto da classe *Element* contém uma lista de objetos da classe *Element Feature*. Esta classe é descrita na próxima seção.

As subclasses da classe *Element* se referem aos tipos de elementos existentes no modelador e são criadas automaticamente pelo sistema. Estas subclasses são específicas para cada modelador.

A classe *Element* implementa métodos para se obter as informações necessárias para a criação do modelo discretizado para uma análise numérica. Exemplos são métodos para obtenção da conectividade (lista de nós) de um elemento ou para acessar os atributos aplicados ao elemento ou em parte do elemento.

O método que fornece a conectividade do elemento assume uma numeração seqüencial dos nós do elemento no sentido anti-horário, fornecida pelo modelador através de uma das funções responsáveis pela comunicação entre o ESAM e o modelador. Esse método, no entanto, pode ser redefinido pelo usuário configurador, nas subclasses de *Element*, como por exemplo, para fornecer a conectividade do elemento no sentido horário.

Os métodos para acesso aos atributos retornam uma lista com todos os atributos de um determinado tipo. Esses métodos são usados, por exemplo, para se consultar todos os lados de elementos que possuem uma determinada carga distribuída. Segue no apêndice A.6 a organização desta classe mostrando suas variáveis e métodos.

3.4.2.6 Classe *Element Feature*

Esta classe existe para facilitar a identificação de atributos aplicados a partes de elementos. Um objeto da classe *Element Feature*, assim como na classe *Node*, referencia o objeto geométrico correspondente à entidade geométrica à qual a parte do elemento representada está associada.

Os métodos desta classe são referentes a informações de atributos de partes de um elemento finito, como métodos que retornam a conectividade da parte do elemento que contém um atributo ou métodos que retornam a lista de atributos de um determinado tipo aplicados à parte do elemento. Estes métodos são usados, por exemplo, para se obter uma carga distribuída aplicada a um lado de elemento.

A Fig. 3.5 ilustra as relações existentes entre os objetos das classes *Node* e *Element* com os objetos das classes *Geometry* e *Attribute*. Este relacionamento permite o mapeamento dos atributos das entidades geométricas para as entidades da malha.

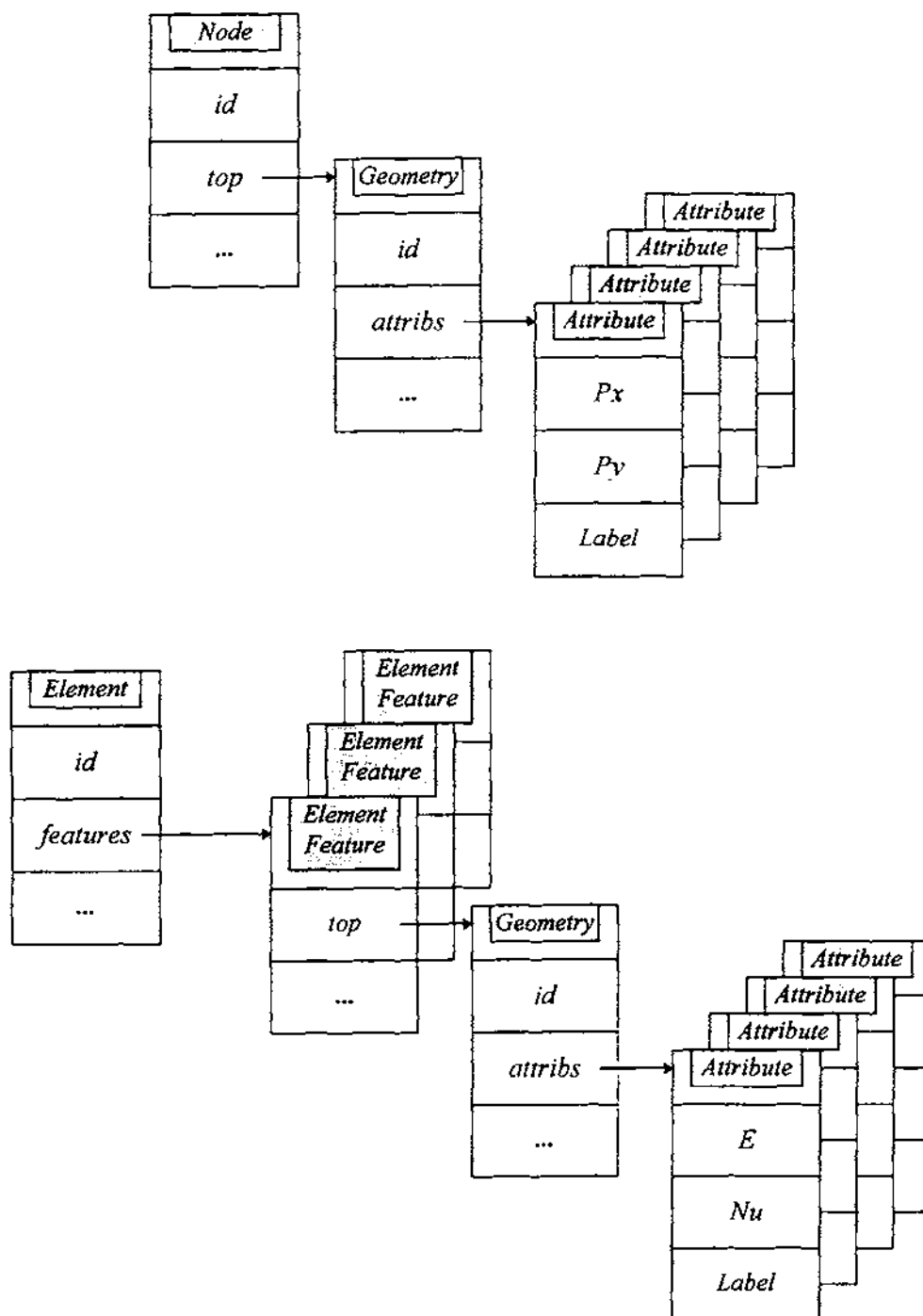


Figura 3.5 - Relações entre os objetos das classes *Node*, *Element*, *Geometry* e *Attribute*.

A organização da classe *Element Feature* com suas variáveis e métodos pode ser encontrada no apêndice A.7.

3.5 Funções de acesso à tecnologia da aplicação

A especificação de um atributo pode requisitar informações referentes ao modelador. Estas informações são armazenadas na base de dados do modelador, embora possam não estar disponíveis diretamente. O meio como as informações são obtidas é, portanto, dependente da implementação da base de dados. Neste sentido, é desejável que as informações sobre o modelador sejam obtidas através de chamadas procedurais e não através do acesso explícito a sua estrutura de dados. Ou seja, deve existir uma interface entre a base de dados do modelador e o restante da aplicação. Esta interface é feita através de um conjunto de funções, referidas como funções MIS (*Modeling Interface Specification*), que representam uma camada de abstração sobre a base de dados do modelador.

A comunicação é feita somente através da API (*Application Program Interface*), ou seja, das variáveis dos argumentos desses operadores, que não contêm nenhum tipo de estrutura de dados. O conjunto de funções deve ser suficiente para permitir a realização das tarefas referentes à especificação dos atributos requisitados para a simulação. No caso de se utilizar um outro modelador geométrico, uma das tarefas é reescrever as funções MIS para este novo modelador.

A especificação das funções MIS para atual versão do sistema ESAM podem ser classificadas em três grupos, de acordo com sua funcionalidade.

- **Funções de inicialização** - São funções responsáveis por informações fundamentais ao funcionamento do sistema. Essas funções são chamadas automaticamente pelo sistema ao ser inicializado.
- **Funções de seleção** - São funções responsáveis pelas informações referentes às entidades selecionadas pelo usuário.
- **Funções de acesso** - São funções responsáveis por informações que estão armazenadas na base de dados do modelador.

Uma relação das funções MIS utilizadas pelo presente sistema é mostrada no apêndice B.

3.6 Ligação entre a parte configurável e o modelador

O módulo que faz a ligação entre a parte configurável e o modelador, *binding* (Fig. 3.2), é a interface responsável pelo acesso da parte configurável ao modelador através das funções MIS. Na parte configurável, os métodos das *Modeling classes* representam a interface para acessar as informações do modelador. No modelador, as funções MIS disponibilizam o acesso a estas informações.

A forma como as funções do programa hospedeiro (escritas em C) são acessadas pela parte configurável depende da linguagem de configuração utilizada. Conforme mencionado anteriormente, a linguagem de configuração Lua permite o acesso a funções escritas em linguagem C. Para isto é necessário apenas que a função a ser chamada seja registrada como uma função em C [Celes *et al.* 1995].

Dessa forma obtém-se uma importante independência entre os grupos responsáveis pelo desenvolvimento da aplicação como um todo. Por um lado, o responsável pela implementação do modelador escreve as funções MIS sem conhecer características da linguagem de configuração utilizada e do sistema de configuração. Por outro lado, o configurador acessa os dados do modelador, sem conhecer o código, através de métodos das classes definidas no próprio ambiente de configuração.

3.7 Serviços oferecidos pelo sistema

Os serviços oferecidos pelo ESAM são funções, implementadas em C, que incorporam o ambiente de configuração de atributos à aplicação (modelador) e têm como objetivo permitir ao usuário final acessar às funcionalidades do ambiente de configuração, tais como especificar um atributo, aplicar um atributo a uma entidade geométrica, etc.

Esses serviços podem ser associados, por exemplo, através de funções *callbacks* de eventos de interface com o usuário. Os serviços são agrupados da seguinte maneira:

- **Serviços de inicialização e finalização** - Estes serviços são responsáveis pelas tarefas de inicialização e finalização necessárias ao ambiente de configuração. Segue abaixo uma descrição de cada um destes serviços indicando a funcionalidade de cada um.

EsamOpen () - Este serviço inicializa o ESAM, carregando o sistema e realizando todos os procedimentos necessários a sua utilização.

EsamCreateModel () - Este serviço dispara o método *new* da classe *Model*, criando o objeto que representa a abstração do modelo geométrico a ser utilizado pelo sistema. Este serviço deve ser chamado todas as vezes que um novo modelo geométrico for criado.

EsamDeleteModel () - Este serviço chama o método *delete* da classe *Model*, eliminando o modelo. Este serviço deve ser chamado todas as vezes que um modelo geométrico criado pela aplicação for destruído.

EsamCreateNumericalModel () - Este serviço dispara o método *new* da classe *Numerical Model*, criando o objeto que representa a abstração do modelo numérico a ser utilizado pelo sistema. Este serviço deve ser chamado todas as vezes que um novo modelo numérico for criado.

EsamDeleteNumericalModel () - Este serviço chama o método *delete* da classe *Numerical Model*, eliminando o modelo numérico. Este serviço deve ser chamado todas as vezes que um modelo numérico criado pela aplicação for destruído.

EsamClose () - Este serviço encerra a utilização do sistema ESAM.

Serviços de especificação - Estes serviços são os responsáveis por tarefas como criar um tipo novo de atributo, modificar um atributo existente, etc. Segue uma descrição destes serviços.

EsamCreateAttribute () - Este serviço dispara o método *createattribute* da classe *Model*, relativo ao objeto corrente. Este serviço deve ser utilizado para que o usuário possa criar um atributo de um tipo definido no arquivo de configuração.

EsamAttachAttribute () - Este serviço dispara o método *attachattribute* da classe *Model*, relativo ao objeto corrente, que aplica o atributo corrente na(s) entidade(s) do modelo que estiverem selecionadas naquele instante.

EsamSetCurrentAttribute () - Este serviço dispara o método *showattributedlg* da classe *Model*, que permite ao usuário escolher entre os atributos existentes qual será o atributo corrente do modelo.

EsamModifyAttribute () - Este serviço dispara o método *modify* da classe *Model*, relativo ao objeto corrente, possibilitando ao usuário modificar os valores de atributos existentes.

EsamInquire () - Este serviço dispara o método *showinquiredlg* da classe *Model*, relativo ao objeto corrente, colocando ao usuário todos os serviços de consulta existente.

EsamSpecifyAttributes () - Este serviço dispara o método *specifyattributes* da classe *Model* relativo ao objeto corrente, colocando a disposição do usuário um conjunto contendo alguns dos serviços que serão descritos a seguir.

- **Serviços de cliente** - Estes serviços são chamados pela aplicação sempre que uma determinada ação sobre uma entidade geométrica for efetuada, como, por exemplo, a criação de uma curva. A parte configurável pode então tratar os atributos das entidades do modelador que são criadas ou eliminadas. Segue uma descrição destes serviços.

EsamCreateGeometryEntity (identity, type) - Este serviço chama o método *creategeometry* da classe *Model*, relativo ao objeto corrente, passando como parâmetros o identificador e o tipo da entidade a ser criada para a criação do objeto associado a esta entidade.

EsamDeleteGeometryEntity (identity, type) - Este serviço chama o método *deletegeometry* da classe *Model*, relativo ao objeto corrente, passando como parâmetros o identificador e o tipo da entidade a ser eliminada.

EsamSplitGeometryEntity (identold, identnew1, identnew2, type) - Este serviço chama o método *splitgeometry* da classe *Model*, relativo ao objeto corrente, passando como parâmetros os identificadores e o tipo das entidades envolvidas na divisão, ou seja, a entidade a ser dividida e as duas novas entidades criadas a partir da divisão da primeira.

EsamJoinGeometryEntity (identold1, identold2, idnew, type) - Este serviço chama o método *joingeometry* da classe *Model*, relativo ao objeto corrente, passando como parâmetros os identificadores e o tipo das entidades envolvidas na união, ou seja, as entidades a serem unidas e a nova entidade criada a partir da união das primeiras.

- **Serviços de validação** - Estes serviços são os responsáveis pela validação, no que se refere ao gerenciamento de atributos, de ações sobre as entidades geométricas do modelador, como a operação de união e divisão de entidades. Segue uma descrição destes serviços.

EsamCheckCreateGeometryEntity (identity, type) - Este serviço dispara o método *checkcreategeometry* da classe *Model*, relativo ao objeto corrente, passando como parâmetros o identificador da entidade a ser criada e o seu tipo. Este serviço verifica se esta entidade pode ou não ser criada.

EsamCheckDeleteGeometryEntity (identity, type) - Este serviço dispara o método *checkdeletegeometry* da classe *Model*, relativo ao objeto corrente, passando como parâmetros o identificador da entidade a ser eliminada e o seu tipo. Este serviço verifica se esta entidade pode ou não ser eliminada.

EsamCheckSplitGeometryEntity (identold, type) - Este serviço dispara o método *checksplitgeometry* da classe *Model*, relativo ao objeto corrente, passando como parâmetros o identificador da entidade a ser dividida e o seu tipo. Este serviço verifica se a divisão desta entidade pode ou não ser realizada.

EsamCheckJoinGeometryEntity (identold1, identold2, type) - Este serviço dispara o método *checkjoingeometry* da classe *Model*, relativo ao objeto corrente, passando como parâmetros os identificadores das entidades a serem unidas e os seus tipos. Este serviço verifica se a união destas entidades pode ou não ser efetuada.

- **Serviços de entrada e saída** - Estes serviços são responsáveis por tarefas como carregar o arquivo de configuração desejado ou gravação e leitura de um arquivo contendo informações a respeito de um modelo gerado. Segue uma descrição destes serviços.
-

EsamLoadAttributeFile (filename) - Este serviço carrega o arquivo de configuração cujo nome é passado como parâmetro.

EsamWriteEsamFile () - Este serviço é responsável pela gravação das informações relativas ao modelo geométrico (geometria e atributos) em um arquivo. Este serviço mostra um diálogo para a definição do nome deste arquivo e dispara os métodos *writeattrtofile* e *writetoptofile* da classe *Model*, relativos ao objeto corrente.

EsamReadEsamFile () - Este serviço é responsável pela leitura das informações relativas ao modelo geométrico (geometria e atributos) de um arquivo. Este serviço mostra um diálogo para a escolha deste arquivo e dispara o método *readattrfromfile* e *readtopfromfile* da classe *Model*, relativos ao objeto corrente.

EsamWriteAttributesEsamFile () - Este serviço é responsável pela gravação das informações relativas apenas aos atributos do modelo em um arquivo. Este serviço mostra um diálogo para a definição do nome deste arquivo e dispara o método *writeattrtofile* da classe *Model*, relativo ao objeto corrente.

EsamReadAttributesEsamFile () - Este serviço é responsável pela leitura das informações relativas apenas aos atributos do modelo de um arquivo. Este serviço mostra um diálogo para a escolha deste arquivo e dispara o método *readattrfromfile* da classe *Model*, relativo ao objeto corrente.

EsamWriteNumericalModelFile () - Este serviço é responsável pela gravação em arquivo das informações relativas ao modelo numérico criado. Este serviço exibe um diálogo que permite ao usuário definir o nome deste arquivo e o formato configurado no qual este arquivo deve ser escrito. Definidos nome e formato, o serviço dispara o método *writetofile* da classe *Numerical Model*.

EsamReadNumericalModelFile () - Este serviço é responsável pela leitura em arquivo das informações relativas ao modelo numérico criado. Este serviço exibe um diálogo que permite ao usuário definir o nome deste arquivo e o formato configurado no qual este arquivo deve ser lido. Definidos nome e formato, o serviço dispara o método *readfromfile* da classe *Numerical Model*.

3.8 ESAM Browser

O *Browser* do ESAM é uma ferramenta oferecida para a configuração da aplicação, ou seja, para a criação de novas classes ou redefinição de classes ou métodos existentes, de acordo com as necessidades específicas. Além da configuração da aplicação, o *Browser* permite ao usuário conhecer a hierarquia das classes do sistema. Através desta ferramenta, pode-se obter diversos tipos de informações sobre as estruturas das classes disponíveis. O *Browser* permite explorar uma classe desde a sua definição até a implementação de cada um dos seus métodos. É possível, ainda, se obter informações sobre como as classes se relacionam entre si.

Conforme mencionado anteriormente, existem duas hierarquias de classes no sistema: uma das classes referentes às entidades do modelador (*Modeling classes*) e uma das classes referentes aos atributos (*Attribute classes*).

O *Browser* trabalha com o conceito de objeto selecionado corrente e possui três colunas, tal como mostrado na Fig. 3.6. Cada coluna contém uma lista de classes ou métodos sobre o qual podem ser efetuadas algumas ações. As ações referem-se sempre ao item da lista que está selecionado. Na primeira coluna define-se qual das duas hierarquias de classes se deseja acessar (*Modeling* ou *Attribute*). Abaixo da classe escolhida são mostradas suas subclasses imediatas. Estas subclasses não podem ser modificadas pelo configurador. A ação disponível nesta coluna consiste na criação de uma subclasse da classe selecionada (botão *Add Class*).

A segunda coluna contém lista de subclasses da classe selecionada na primeira coluna. A subclasse criada é adicionada na lista da segunda coluna. As ações oferecidas por esta coluna consistem em se criar uma subclasse (botão *Add Subclass*) ou se modificar um método de uma classe selecionada (botão *Modify Method*).

A terceira coluna contém uma lista de métodos da classe selecionada na segunda coluna. Os métodos seguidos por um asterisco correspondem aos métodos que são definidos na classe corrente. Os outros métodos, portanto, são herdados das classes-base da classe corrente. Todas as listas são atualizadas automaticamente pelo *Browser*.

Deve-se ressaltar que, para o configurador da aplicação, fica transparente o fato das classes *Modeling* serem construídas automaticamente pelo sistema, parecendo que existem no arquivo de configuração do mesmo modo que as classes *Attribute*. Para estas classes construídas automaticamente, no entanto, não é permitido se criar subclasses, só é permitido redefinir os seus métodos. Isso se deve ao fato dessas classes se referirem às entidades geométricas, não gerenciadas pelo configurador. Desse modo, seria inconsistente permitir o configurador criar subclasses dessas classes.

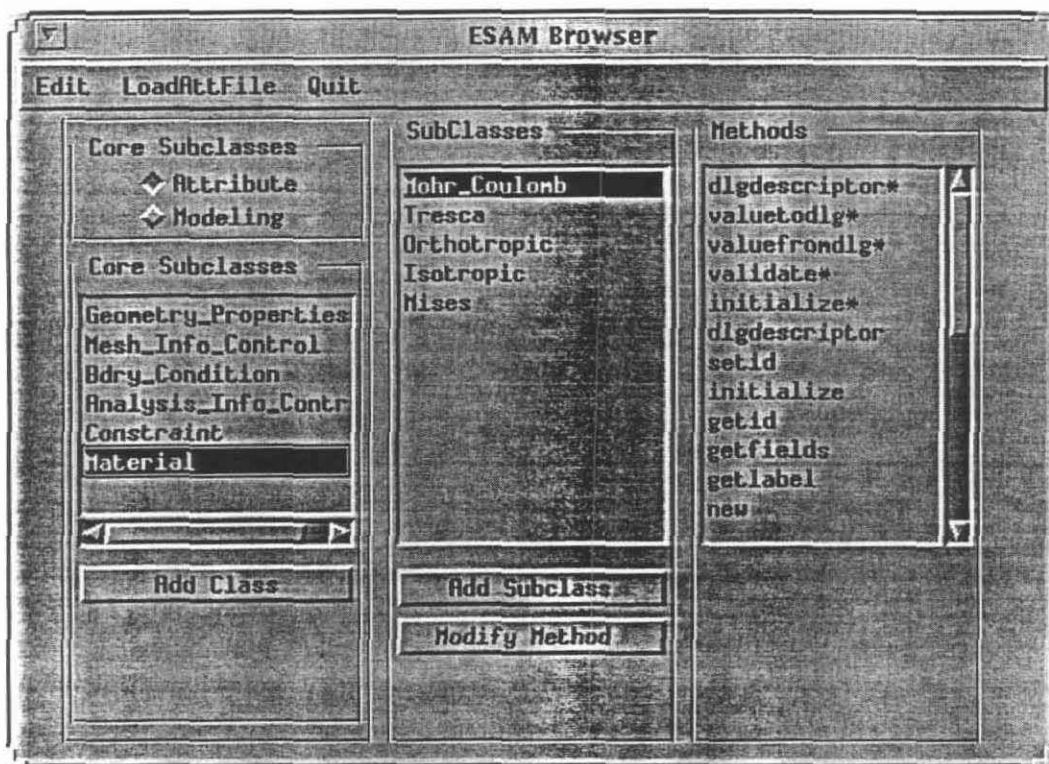


Figura 3.6 - ESAM Browser.

Capítulo 4

Simulação Adaptativa Extensível e Configurável

Este capítulo tem por objetivo descrever o sistema proposto neste trabalho, que é um ambiente, extensível e configurável para simulação adaptativa de problemas bidimensionais de mecânica computacional utilizando o método dos elementos finitos. O usuário tem o poder de configurar os atributos da análise de acordo com o tipo de simulação desejada, utilizando uma linguagem de extensão acoplada e interpretada, com alto poder de expressão.

O desenvolvimento deste sistema configurável tornou-se viável com a integração do sistema EDP/AMVIEW (módulo adaptativo), descrito no capítulo 2, com o sistema ESAM (*Extensible System of Attributes Management*), detalhado no capítulo anterior. A partir desta integração, o gerenciamento dos atributos utilizados, que era feito pelo próprio módulo adaptativo, passou a ser tarefa do ESAM. Em outras palavras, todas as informações relativas aos atributos do modelo foram retiradas do sistema anterior e passaram a ser gerenciadas, exclusivamente, pelo sistema ESAM. Uma das vantagens disso é permitir o acesso à tecnologia do EDP/AMVIEW sem o conhecimento dos detalhes desta tecnologia e nem mesmo da linguagem em que a aplicação foi implementada.

Para a integração do ESAM com uma aplicação, conforme descrito no capítulo anterior, dois procedimentos básicos devem ser seguidos. O primeiro consiste na implementação das funções responsáveis pela ligação da aplicação com o ESAM. Estas funções, denominadas funções MIS (*Modeling Interface Specification*), têm seus nomes e API (*Application Program Interface*) pré-definidos pelo ESAM. Uma lista completa destas

funções pode ser encontrada no apêndice B. O segundo procedimento consiste em acoplar os serviços oferecidos pelo ESAM à aplicação, colocando-os a disposição do usuário. Estes serviços, evidentemente, estão relacionados com a manipulação dos atributos da aplicação e também estão especificados no capítulo anterior. A Fig. 4.1 ilustra a integração do ESAM com o módulo adaptativo deste trabalho. Esta figura servirá como base para a descrição do sistema proposto nas seções deste capítulo.

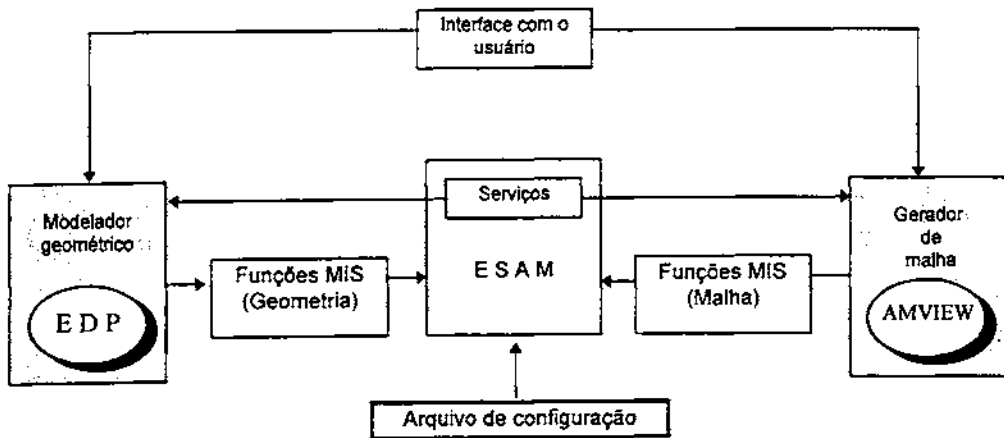


Figura 4.1 - Integração do ESAM com o módulo de simulação.

4.1 Integração do ESAM com o módulo de modelagem geométrica

No capítulo 2 ficaram caracterizadas as fases envolvidas em um processo de simulação adaptativa bidimensional de mecânica computacional, bem como os módulos do sistema existente responsáveis por cada uma destas fases [Cavalcante 1994].

A etapa de geração do modelo geométrico e associação dos atributos às entidades do modelo eram realizadas pelo modelador geométrico EDP. Para o desenvolvimento do sistema adaptativo configurável apresentado, toda a tecnologia utilizada para a criação e manipulação do modelo geométrico foi aproveitada. O tratamento de atributos da simulação, que era feito pela própria aplicação, agora é inteiramente gerenciado pelo

sistema ESAM. O EDP é responsável somente pelo gerenciamento dos serviços de modelagem geométrica existentes.

Esta seção descreve a integração do sistema ESAM com o módulo de modelagem geométrica (EDP). De acordo com a arquitetura do sistema ESAM, para a sua integração com qualquer modelador geométrico é necessário apenas que sejam escritas as funções MIS, que fazem a interface do modelador com o ESAM. Além disso, os serviços oferecidos pelo ESAM devem ser chamados diretamente pela aplicação de modo a se tornarem disponíveis para o usuário. Nas subseções que se seguem será mostrada a implementação de algumas funções MIS relativas à modelagem geométrica e alguns serviços do ESAM que foram colocados a disposição do usuário através do modelador.

4.1.1 Funções MIS para o modelador geométrico

Para a integração com o modelador geométrico foram implementadas as funções MIS associadas à geometria. Para a implementação destas funções é necessário ter um bom nível de conhecimento sobre a estrutura de dados utilizada pelo modelador.

Conforme mencionado no capítulo anterior, as classes referentes às entidades geométricas do modelador são criadas automaticamente pelo ESAM. Isto é feito através de uma função MIS de inicialização, chamada por um serviço de inicialização do sistema. Essa função registra cada tipo de entidade geométrica, fornecendo o nome (corresponde ao nome da classe criada pelo sistema), dado por uma cadeia de caracteres, a dimensão geométrica (representa a classe base da classe criada) e o identificador da entidade no contexto do modelador. Para o modelador geométrico utilizado, esta função foi implementada registrando três tipos de entidades: *vértice*, *curva* e *região*, que são as entidades utilizadas pelo modelador. A entidade geométrica *vértice*, por exemplo, foi registrada com nome "Vertex", seu identificador no contexto do modelador é o número 5 e sua dimensão (classe base) é 0 (zero). O pseudo-código com a implementação desta função para este modelador está mostrado na Fig. 4.2.

```

função MisRegGeomEntities (RegistraEntidade(nome, tipo, dimensão))

// Chama função recebida como parâmetro com os valores
// nome, tipo, dimensão
RegistraEntidade("Vertex", 5, 0)
RegistraEntidade("Curve", 4, 1)
RegistraEntidade("Region", 1, 2)

fim da função

```

Figura 4.2 - Função que registra os tipos de entidades geométricas utilizadas pelo modelador.

As funções MIS de seleção, responsáveis por informações referentes a entidades geométricas selecionadas pelo usuário, também foram implementadas para o modelador em questão. Por exemplo, é mostrada na Fig. 4.3 a função que informa ao sistema ESAM quais as entidades que estão selecionadas na aplicação. Essa informação depende da maneira como o modelador foi implementado, e nesta figura é mostrada a implementação desta função para o caso do EDP.

```

função MisGetNSelect (nentidades, entidades, tipos)

para todas as regiões selecionadas
  atribui o identificador da região corrente a um campo do vetor
  entidades.
  atribui o identificador do tipo de entidade corrente a um campo do
  vetor tipos.
fim do para

...
// Repete-se o procedimento para os outros tipos de entidades
// existentes (vértices e curvas)

fim da função

```

Figura 4.3 - Função utilizada pelo ESAM para requisitar as entidades geométricas selecionadas.

É mostrada na Fig. 4.4 uma outra função MIS onde o fluxo de informações se dá no sentido inverso, ou seja, do ESAM para o modelador. Através desta função o ESAM fornece ao modelador uma lista de entidades existentes na estrutura de dados do modelador sobre as quais existe informação de atributos em um determinado momento.

Com esta informação o modelador pode realçar estas entidades de alguma forma, como por exemplo, trocando as cores destas entidades. Um exemplo da utilização desta função é o momento em que o serviço de consulta a atributos é acionado, no qual a aplicação deve mostrar quais as entidades que possuem um determinado atributo. Como o modelador não possui informações relativas aos atributos, cabe ao ESAM passar este tipo de informação para o modelador.

função MisSetEntities (*nentidades, entidades, tipos*)

```
para cada entidade do vetor entidades
conforme ( tipo )
  caso Vertex : seleciona vértice corrente
  caso Região : seleciona região corrente
  caso Curve : seleciona curva corrente
fim do conforme
fim do para
```

Fim da função

Figura 4.4 - Função para informar ao modelador quais as entidades que contêm informações de atributos.

A exemplo das funções mostradas, todas as funções MIS relativas à modelagem geométrica foram devidamente implementadas para que a integração entre os sistemas EDP e ESAM pudesse ser feita.

4.1.2 Serviços ESAM utilizados pelo modelador geométrico

Conforme dito anteriormente, além da implementação das funções MIS, é necessário também que o modelador coloque à disposição do usuário os serviços necessários à especificação dos atributos oferecidos pelo sistema ESAM. Grande parte destes serviços é formado por funções do ESAM que são chamadas pela aplicação (modelador) através da ação do usuário sobre um objeto de interface. Nesta seção serão mostrados alguns desses serviços.

Para a integração do ESAM com uma aplicação qualquer é necessária a inicialização do sistema. Esta inicialização é feita através da função *EsamOpen*, que consiste na inicialização de algumas variáveis utilizadas pelo sistema e na criação automática das classes referentes às entidades geométricas.

Conforme mencionado no capítulo anterior, o ESAM constrói abstrações dos modelos geométrico e numérico existentes na aplicação. A função *EsamCreateModel*, por exemplo, cria uma abstração relativa ao modelo geométrico a ser construído pela aplicação.

Para a manipulação direta dos atributos são utilizados alguns serviços de especificação, que são responsáveis por tarefas como criação de um novo atributo, modificação de um atributo existente, especificação de um atributo com atributo corrente, aplicação de um atributo a uma entidade geométrica ou consulta a um atributo de uma determinada entidade. O ESAM oferece um importante serviço que engloba um conjunto de outros serviços e é disparado pela aplicação através da chamada da função *EsamSpecifyAttributes*. O diálogo disparado por este serviço é mostrado na Fig. 4.5.

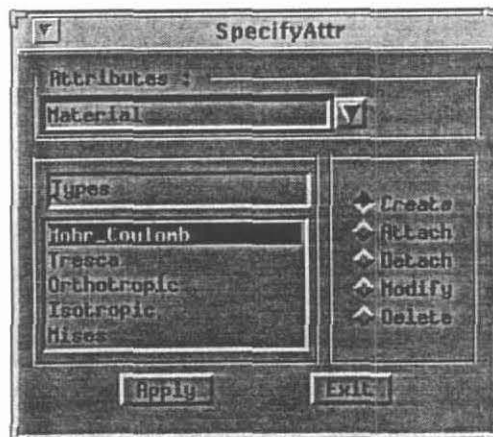


Figura 4.5 - Diálogo disparado pelo serviço de especificação de atributos.

Os serviços disponíveis através deste serviço de especificação também podem ser acionados isoladamente. Por exemplo, existe a função *EsamCreateAttribute* que permite a criação de um atributo de um determinado tipo. O diálogo disparado por este serviço está exemplificado na Fig. 4.6. Do mesmo modo, existe um outro serviço, a função *EsamAttachAttribute*, que corresponde ao serviço para aplicação do atributo corrente às entidades selecionadas.

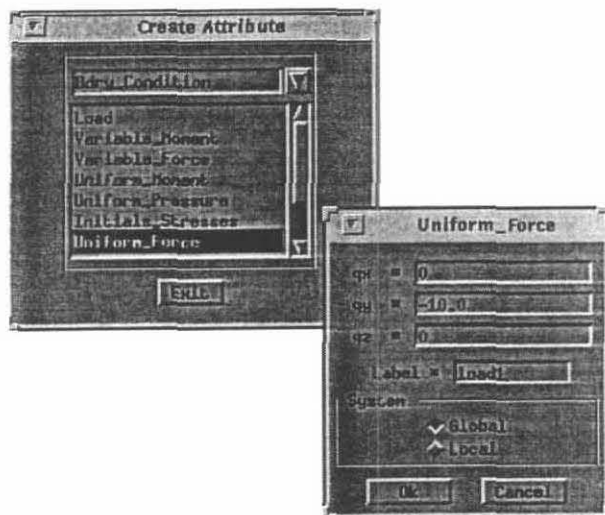


Figura 4.6 - Diálogo disparado pelo serviço de criação do atributo.

Além dos serviços de especificação, alguns serviços de consulta são utilizados pelo modelador. Entre eles está um serviço, análogo ao serviço de especificação, que contém um conjunto de serviços de consulta. Esta é a função *EsamInquire*, que oferece três serviços importantes de consulta: um serviço que mostra quais são os atributos associados a uma determinada entidade, outro serviço que mostra todas as entidades que contêm um determinado atributo e, finalmente, um serviço que mostra todos os atributos de um determinado tipo existente. O diálogo disparado pela função *EsamInquire* é mostrado na Fig. 4.7.

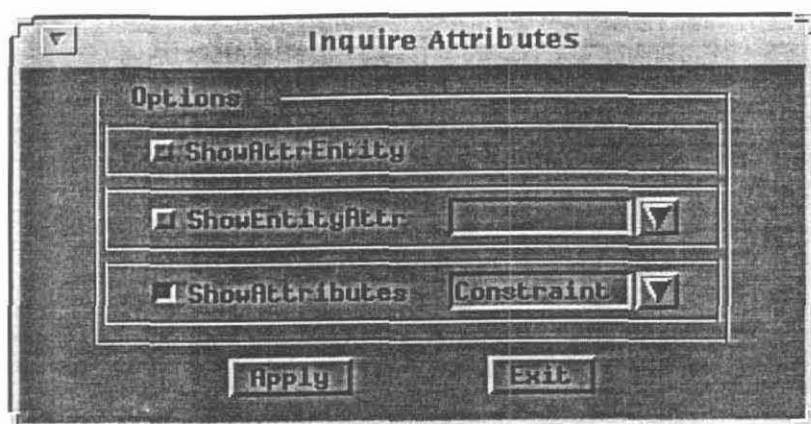


Figura 4.7 - Diálogo disparado pelo serviço de consulta.

Outros serviços importantes oferecidos pelo ESAM, e que foram utilizados pela aplicação em questão, são os serviços responsáveis pela persistência (armazenamento permanente) de um modelo dentro do contexto do ESAM. Isto é, são serviços para salvar em arquivo a abstração da geometria do modelo e os atributos da simulação. Entre estes está o serviço *EsamWriteEsamFile* que gera o arquivo contendo as informações relativas ao modelo criado.

Além destes serviços foram utilizados também os serviços disparados quando o modelador executa operações de divisão (*split*), união (*join*) e eliminação (*delete*) em uma entidade geométrica. Estes serviços são as funções: *EsamSplitGeometryEntity*, *EsamJoinGeometryEntity* e *EsamDeleteGeometryEntity*, que consistem em realizar estas ações sobre a abstração do modelo geométrico no ESAM. Conforme mencionado no capítulo anterior, para a realização destas tarefas o ESAM oferece alguns serviços de validação que também foram utilizados, como, por exemplo, as funções *EsamCheckJoinGeometryEntity* e *EsamCheckSplitGeometryEntity*.

4.2 Integração do ESAM com o gerador do modelo numérico

Da mesma forma que foi feito para o módulo de modelagem geométrica, toda tecnologia utilizada pelo módulo que faz a geração e refinamento da malha (AMVIEW) [Cavalcante 1994] foi utilizada no desenvolvimento do sistema de simulação adaptativa proposto. Apenas o tratamento dos atributos foi retirado deste módulo, o que passou a ser responsabilidade do sistema ESAM. Esta seção descreve a integração do sistema ESAM com o módulo AMVIEW.

4.2.1 Funções MIS para o gerador do modelo numérico

Para a integração com o módulo de geração do modelo numérico foram implementadas as funções associadas a informações sobre a malha de elementos finitos. Assim como para o modelador geométrico, a implementação destas funções requer um alto nível de conhecimento sobre a estrutura de dados utilizada pelo gerador de malha. Nesta subseção são mostradas como foram implementadas algumas destas funções.

As classes referentes aos tipos de elementos utilizados pelo modelador são criadas automaticamente pelo sistema. Isto é feito através de uma função MIS que registra estes tipos de elementos. Esta função está mostrada na Fig. 4.8, onde cada tipo de elemento é registrado passando o nome (corresponde ao nome da classe criada pelo sistema), dado por uma cadeia de caracteres, o número de nós do elemento, o identificador do tipo do elemento e o grau do polinômio de interpolação utilizado.

```
função MisRegElementTypes (RegistraElemento(nome, nnos, tipo, interp))

// Chama função recebida como parâmetro com os valores
// nome, número de nós, tipo e interpolação
RegistraElemento("T3" , 3, 0, 1)
RegistraElemento("T6" , 6, 1, 2)
RegistraElemento("Q4" , 4, 4, 1)
RegistraElemento("Q8" , 8, 3, 2)

fim da função
```

Figura 4.8 - Função para registro dos tipos de elementos finitos utilizados pelo modelador.

Foram implementadas também funções MIS responsáveis pelas informações relativas ao modelo numérico como um todo (malha). Uma destas funções é a função que permite ao ESAM obter uma lista com os identificadores dos elementos existentes na malha (Fig. 4.9).

```
função MisGetElemList (elemlist)

para todos os elementos da malha
coloca o identificador do elemento corrente na posição correspondente do
vetor elemlist
fim do para

fim da função
```

Figura 4.9 - Função que passa para o ESAM os identificadores dos elementos da malha.

Outra função MIS que foi implementada para o módulo que faz a geração de malha foi a função que dado o identificador do nó retorna suas coordenadas (Fig. 4.10).

```
função MisGetNodeCoords (node, coords)

atribui ao campo x do vetor coords a coordenada x do nó node
atribui ao campo y do vetor coords a coordenada y do nó node

fim da função
```

Figura 4.10 - Função para captura das coordenadas de um nó da malha.

Conforme dito anteriormente, o mapeamento dos atributos da geometria para a malha é feito pelo ESAM. Para isto ele necessita de algumas informações a respeito da associação das entidades de malha (nós e elementos) com as entidades geométricas (vértices, curvas e regiões) do modelo. Estas informações são passadas através de algumas funções MIS. Como exemplo de uma destas funções, é mostrada na Fig. 4.11 a função que dado o identificador de um elemento, informa ao ESAM em que entidade geométrica este elemento está contido.

```
função MisGetElemTopology (elem, type, geometry)
para todas as regiões do modelo faça
se o elemento estiver nesta região então
atribui a geometry o identificador da região
atribui a type o identificador correspondente ao tipo região
interrompe a função
fim do se
fim do para
fim da função
```

Figura 4.11 - Função utilizada para a obtenção da entidade geométrica associada a um elemento da malha.

Além das funções mostradas, todas as funções MIS relativas ao modelo numérico foram implementadas para a integração do AMVIEW com o ESAM.

4.2.2 Serviços ESAM utilizados pelo gerador de malhas

Os serviços do ESAM relativos ao modelo numérico foram utilizados pelo AMVIEW para sua integração desta com o gerenciador de atributos.

Para a criação da representação de um modelo numérico gerado, o ESAM oferece um serviço que é executado através da chamada da função *EsamCreateNumericalModel*. Esta função deve ser chamada pela aplicação todas as vezes que um modelo numérico novo for gerado. Este serviço cria a abstração do modelo numérico gerado pelo modelador.

Esta abstração consiste em uma instância da classe *Numerical Model*, e é formada por uma lista de nós (instâncias da classe *Node*) e uma lista de elementos (instâncias da classe *Element*). Estas são todas as informações necessárias ao ESAM para que este possa fazer a geração de um arquivo com todas as informações que definem perfeitamente o modelo numérico.

Outro serviço disponível consiste da operação de saída referente ao modelo numérico. A função *EsamWriteNumericalModelFile* dispara um diálogo que contém diferentes tipos de formato para o arquivo que será utilizado pelo módulo de análise numérica. Este diálogo contém todos os tipos de formatos de arquivos que foram configurados através do arquivo de configuração. O usuário pode, então, escolher o tipo de formato específico para sua necessidade.

4.3 Resumo da arquitetura proposta

Apresenta-se aqui o resultado da integração do sistema para simulação adaptativa, descrito no capítulo 2, com o sistema ESAM. O resultado desta integração é um sistema configurável para simulação de problemas bidimensionais adaptativos de mecânica computacional.

Esta seção tem por objetivo descrever a arquitetura do sistema configurável proposto, ressaltando as diferenças entre este sistema e o sistema anterior descrito no capítulo 2. A Fig. 4.12 mostra a arquitetura do sistema, ilustrando como é feita a comunicação entre os sistemas envolvidos. As linhas tracejadas indicam que uma comunicação é feita através de arquivos.

As modificações feitas sobre o sistema existente concentraram-se nas fases de modelagem geométrica e no mapeamento dos atributos para a malha, que serão discutidas nas subseções que se seguem.

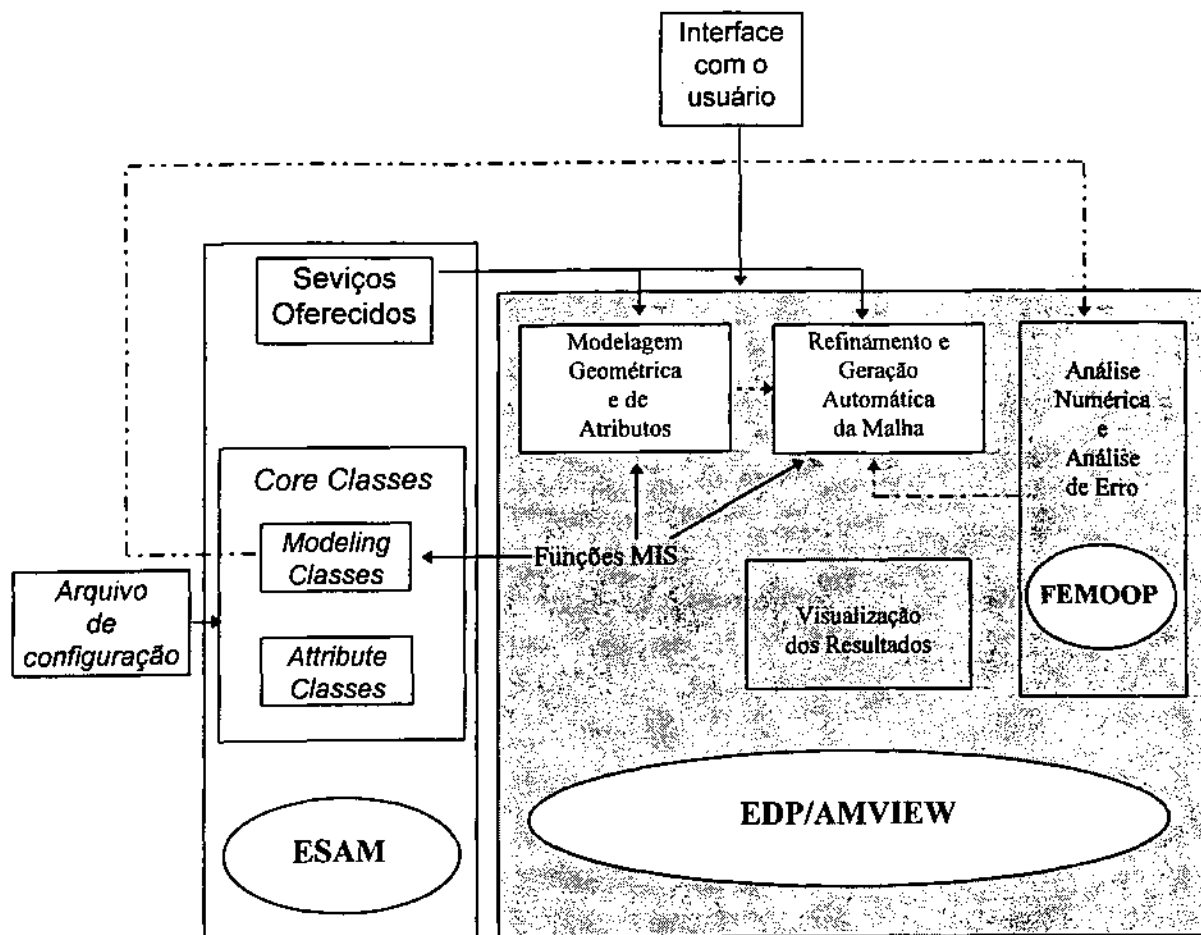


Figura 4.12 - Integração do sistema para simulação adaptativa bidimensional de elementos finitos com o sistema ESAM.

4.3.1 Modelagem geométrica com configuração de atributos

Para o sistema proposto, a etapa de modelagem geométrica utiliza toda a tecnologia desenvolvida no modelador EDP, ou seja, todo o processo de geração da geometria através da inserção de vértices e curvas, e a criação automática das regiões. A diferença fundamental está no gerenciamento dos atributos a serem associados ao modelo, que agora é feito pelo ESAM.

Conforme mencionado no capítulo 2, o tratamento dos atributos era feito pelo próprio modelador EDP que utiliza a biblioteca HED para gerenciamento de subdivisões planares. O HED possui internamente um campo em cada entidade topológica da

estrutura de dados reservado para que a aplicação que o utiliza associe atributos a estas entidades. Como a biblioteca desconhece a natureza destes atributos, cabe à aplicação desenvolver funções específicas para manipulá-los. Desta forma, para a inclusão de novos tipos de atributos era necessário um conhecimento mais detalhado do modelador e também da linguagem em que este foi implementada.

No sistema proposto ESAM, todas as informações associadas aos atributos da simulação, desde a sua definição e associação a entidades geométricas até o formato de entrada e saída das informações relativas aos atributos, são feitas pelo ESAM.

A utilização do ESAM possibilitou ao modelador a flexibilidade da configuração dos atributos a serem associados ao modelo através de uma linguagem acoplada extremamente fácil de ser aprendida. Desta forma, o acesso à tecnologia da aplicação é feito de maneira simples e sem a necessidade de um conhecimento mais detalhado da implementação do EDP e do HED.

4.3.2 Mapeamento dos atributos para a malha dentro de um ambiente configurável

Uma das principais etapas de uma modelagem baseada em geometria consiste no mapeamento dos atributos do modelo geométrico para a discretização, que é feito através de herança, onde a discretização herda, automaticamente, os atributos do modelo geométrico.

Conforme mencionado no capítulo 2, este mapeamento era responsabilidade do próprio módulo de refinamento e geração da malha. O AMVIEW guardava na sua estrutura de dados, além das informações a respeito dos modelos geométrico e numérico, todas as informações associadas aos atributos da simulação.

Após a integração com o ESAM, o mapeamento dos atributos utiliza a mesma filosofia do sistema AMVIEW em sua versão anterior. Ou seja, os atributos aplicados em vértices

são herdados, automaticamente, pelos nós que estão sobre eles, os atributos aplicados em curvas são herdados pelos nós que estão sobre elas e pelos lados dos elementos adjacentes a estas curvas e, finalmente, atributos que são associados às regiões são herdados pelos elementos que estão sobre elas. A diferença está em que este mapeamento agora é feito, integralmente, pelo ESAM. Este sistema, através das funções MIS, obtém todas as informações necessárias para executar este mapeamento.

Como o tratamento dos atributos envolvidos no problema era parte integrante do AMVIEW, e agora é feito pelo ESAM, toda a parte relativa aos atributos foi inteiramente retirada do gerador de malhas, diminuindo bastante o número de linhas de código e conseqüentemente simplificando muito a aplicação.

4.3.3 Funcionalidade da arquitetura proposta

Esta seção descreve a funcionalidade oferecida pelo sistema proposto, ressaltando as tarefas de cada módulo mostrado na Fig. 4.12 e a forma como é realizada a comunicação entre eles.

Para a realização de uma simulação deve-se determinar, inicialmente, quais os tipos de atributos envolvidos no problema a ser modelado. Caso o modelador ainda não tenha sido configurado para suportar estes tipos de atributos, pode-se então incluí-los na aplicação, através da configuração destes atributos. Eles podem ser incorporados ao modelador através do arquivo de configuração. Isto é feito com a criação de novos tipos (classes) de atributos, a implementação de alguns métodos associados ao tipo (classe) criado e a definição do formato de saída das informações relativas a este tipo de atributo.

Com um arquivo de configuração compatível com o tipo de simulação a ser efetuada, o processo de modelagem é iniciado com a geração do modelo geométrico do problema físico a ser simulado. Isto é feito através de uma interface amigável com o usuário, onde, através da manipulação de curvas e vértices, o modelo geométrico é definido. Além da

definição desta geometria, o usuário tem acesso aos serviços oferecidos pelo ESAM (incorporados à aplicação), que permitem a especificação dos atributos a serem associados às entidades do modelo geométrico.

Definido o modelo geométrico e os atributos associados, são gerados dois arquivos. Um arquivo, gerado pelo EDP, contém as informações relativas ao modelo geométrico, como a definição das curvas, vértices e regiões criados e os identificadores associados a cada entidade geométrica. O outro arquivo, gerado pelo ESAM, contém as informações relativas aos atributos criados e associados ao modelo geométrico, bem como a relação entre estes atributos e as representações das entidades geométricas dentro do ESAM. Esta relação é feita através dos identificadores das entidades geométricas. As informações relativas a esta associação são passadas através de uma função MIS implementada. A geração do arquivo de atributos pelo ESAM é disparada pela aplicação através da utilização do serviço *EsamWriteEsamFile*.

Para iniciar a simulação adaptativa, o AMVIEW faz a leitura do arquivos relativo à geometria do modelo e o ESAM procede com a leitura do arquivo de atributos, criando os atributos especificados e as abstrações associadas às entidades geométricas lidas.

Obtidas as informações do modelo gerado na fase inicial, o AMVIEW procede com a geração automática da malha inicial, a partir da geometria recebida e de um parâmetro de controle que é definido pelo usuário. Gerada a malha, o AMVIEW informa ao ESAM através de um outro serviço (*EsamCreateNumericalModel*) que foi gerado um modelo numérico para que o ESAM possa criar a sua abstração relativa a este modelo. Para criar esta abstração, o ESAM utiliza informações passadas pelas funções MIS associadas ao modelo numérico. Com as abstrações dos modelos geométrico e numérico, o ESAM realiza o mapeamento dos atributos, das entidades geométricas (vértices, curvas e regiões) para as entidades da malha (nós e elementos). Neste ponto o ESAM possui todas as informações necessárias relativas à definição de um modelo numérico, ou seja, nós e elementos que associados aos seus respectivos atributos compõem um modelo de

elementos finitos. O ESAM gera, então, um arquivo com o modelo numérico, através do serviço *EsamWriteNumericalFile*, que será lido pelo módulo de análise do sistema (FEMOOP).

O módulo de análise procede então com a análise numérica do problema gerando um arquivo contendo os resultados da análise, o que inclui as informações relativas à análise de erro. Este arquivo é lido pelo AMVIEW, que baseado no erro de discretização obtido na análise, procede com o refinamento e geração da nova malha. Gerada a nova malha, o AMVIEW informa ao ESAM, através do serviço *EsamDeleteNumericalModel*, que o modelo numérico existente anteriormente na parte configurável pode ser eliminado. Em seguida o AMVIEW indica ao ESAM, através da função *EsamCreateNumericalModel*, que este pode criar a abstração ao novo modelo numérico. O ESAM obtém, então, informações a respeito do novo modelo numérico, criando a sua abstração associada a este modelo. Considerando que não existe modificação na geometria, o ESAM realiza novamente o mapeamento dos atributos para a nova malha, gerando novo arquivo que será analisado pelo módulo de análise.

Este ciclo continua até que o erro obtido seja menor do que o erro admitido pelo usuário. A cada modelo analisado pode-se visualizar graficamente os resultados obtidos na análise.

Neste capítulo são apresentados dois exemplos de utilização do sistema proposto neste trabalho com o objetivo de ilustrar a generalidade obtida com a integração de um sistema extensível para configuração de atributos, ESAM, com o sistema para simulação adaptativa desenvolvido por Cavalcante (1994).

Os exemplos mostrados apresentam dois tipos de análises de mecânica computacional. O primeiro exemplo consiste de um problema envolvendo análise de tensões e o segundo exemplo um problema envolvendo uma análise térmica. O objetivo é caracterizar os diferentes tipos de atributos envolvidos em cada problema e mostrar a facilidade com que estes atributos podem ser acrescentados à aplicação.

São mostradas as fases do processo de simulação, ilustrando as diferenças entre os dois tipos de análise dentro do contexto da utilização do sistema e o procedimento que deve ser seguido para a configuração de novos atributos.

5.1 Análise de tensões

O primeiro exemplo apresentado é uma placa em “L” mostrada na Fig. 5.1, submetida a um carregamento lateral unitário. É considerado para solução numérica do problema estado plano de tensões, com módulo de elasticidade $E = 10^3$ e coeficiente de Poisson $\nu = 0.3$.

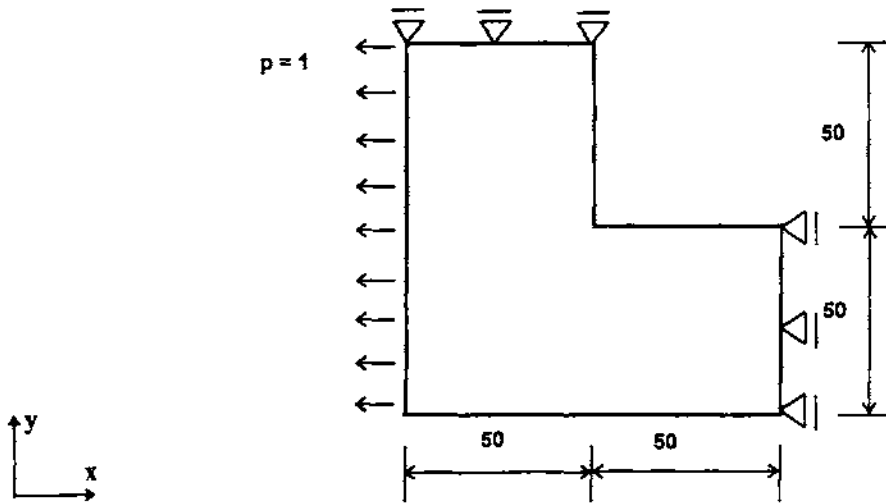


Figura 5.1 - Placa em L.

Os atributos do modelo físico envolvidos neste problema são os seguintes:

- Material isotrópico definido pelos valores do módulo de elasticidade e do coeficiente de Poisson.
- Carregamento uniforme no bordo da esquerda definido por um valor unitário na direção $-x$.
- Condições de suporte caracterizadas pela restrição ou liberdade de deslocamentos nas direções x e y .

Supondo que estes atributos não estejam definidos no contexto do modelador, o primeiro procedimento que deve ser seguido é tornar o modelador capaz de suportar estes tipos de atributos. Com a utilização do sistema de configuração de atributos, ESAM, isto é feito através de um arquivo que contém as definições destes atributos, denominado arquivo de configuração. Para facilitar a criação de novos tipos de atributos, alteração de outros

atributos já existentes, implementação de novos métodos associados às classes existentes, enfim para qualquer alteração no arquivo de configuração, o *Browser* oferecido pelo gerenciador de atributos pode ser utilizado. É ilustrado na Fig. 5.2 a utilização do *Browser* para a criação do atributo material isotrópico utilizado para o primeiro exemplo.

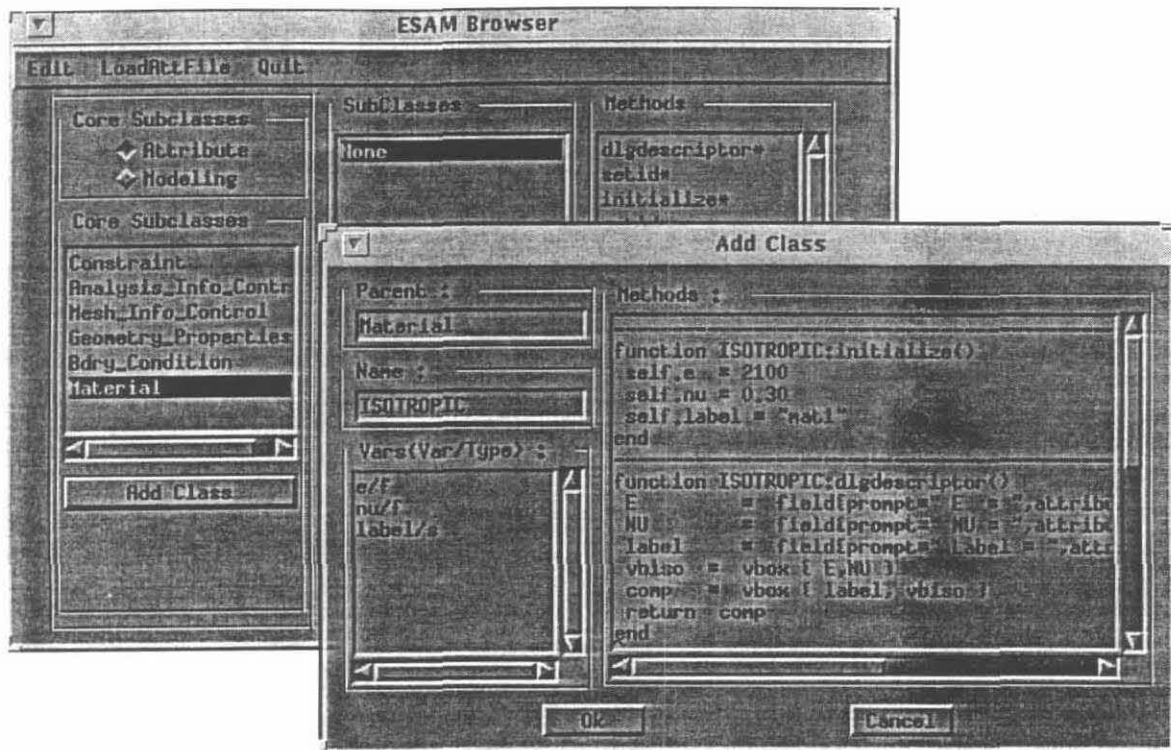


Figura 5.2 - Utilização do *Browser* para criação de um novo tipo de atributo (material isotrópico).

Para a criação de um novo tipo de atributo, cabe ao configurador determinar a superclasse do atributo a ser configurado, indicar o nome do tipo de atributo que ele está criando, especificar as variáveis do atributo e seus respectivos tipos, como, por exemplo, o módulo de elasticidade, definido pela letra *e*, que armazenará um número real. O tipo correspondente a um número real é representado pela letra *f* (*float*). Além disso, o configurador deverá implementar alguns métodos que são utilizados pelo ESAM para a manipulação do tipo de atributo que está sendo criado. É mostrado na Fig. 5.3 a implementação em Lua dos métodos necessários à configuração de um novo tipo de atributo, neste caso o material isotrópico. A variável *self* corresponde ao objeto corrente, em LUA.

```

-- Creation of class
ISOTROPIC      = crtclass { name = "Isotropic",
                           parent = MATERIAL,
                           vars = {"e","ni","label"},
                           ntype = {"f","f","s" }
                           }

-- Definição dos métodos :
-----
function ISOTROPIC:initialize()
  self.e = 2100
  self.nu = 0.30
  self.label = "mat1"
end
-----
function ISOTROPIC:dlgdescriptor()
  E      = field{prompt=" E = ",attributes='SIZE=75x12'}
  NU     = field{prompt=" NU = ",attributes='SIZE=75x12'}
  label  = field{prompt=" Label = ",attributes='SIZE=50x12'}
  vbiso  = vbox { E,NU }
  comp   = vbox { label, vbiso }
  return comp
end
-----
function ISOTROPIC:validate()
  if type(E.value) ~= "number" or E.nvalue <= 0 or
     type(NU.value) ~= "number" or NU.nvalue < 0 or NU.nvalue > 0.5 then
    return nil
  end
  return 1
end
-----
function ISOTROPIC:valuetodlg()
  E:set(self.e)
  NU:set(self.ni)
  label:set( self.label)
end
-----
function ISOTROPIC:valuefromdlg()
  self.e = E.nvalue
  self.ni = NU.nvalue
  self.label = label.value
  self.name = label.value
end
-----
function ISOTROPIC:writeprops( fname )
  write(self.e,"f>10.2\t")
  write(self.nu,"f>10.2\n")
end
-----

```

Figura 5.3 - Código gerado pelo *Browser* para configuração do atributo material isotrópico.

A definição do novo tipo de atributo e a implementação dos métodos associados a este foram inteiramente feitas através do *Browser*, que acessa o arquivo de configuração escrevendo neste arquivo o trecho de código em Lua mostrado acima, relativo à criação do atributo material isotrópico. Segue uma descrição mais detalhada da implementação de cada um dos métodos mostrados.

- **initialize** - Os valores do módulo de elasticidade (e), coeficiente de Poisson (nu) e o rótulo (*label*) do material foram inicializados com os valores 2100.00, 0.3 e “mat1”, respectivamente. Desta forma, todo atributo deste tipo que for criado terá suas variáveis inicializadas com estes valores.
- **dlgdescriptor** - Foi definido o diálogo que será utilizado para a captura dos dados relativos ao atributo do tipo material isotrópico a ser criado. Para a definição deste diálogo são utilizados alguns objetos de interface disponíveis no *toolkit* EDG [Celes 1995]. Este método retorna o diálogo construído. O diálogo criado para a captura dos dados do material isotrópico é mostrado na Fig. 5.4.

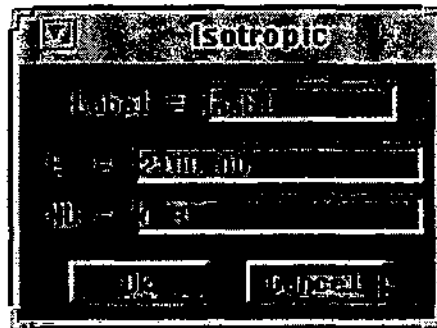


Figura 5.4 - Diálogo criado para captura dos dados referentes ao material isotrópico.

- **validate** - Para o material isotrópico a atribuição de um valor negativo ou de uma cadeia de caracteres para o valor do módulo de elasticidade ou coeficiente de Poisson invalida os dados fornecidos pelo usuário. Para o coeficiente de Poisson não é permitido ainda a atribuição de um valor menor que zero ou maior que 0.5.
- **valuetodlg** - Este método transporta os valores das variáveis e e nu do objeto corrente para os objetos de interface referenciados por E e NU. Em outras palavras, coloca nos respectivos campos do diálogo os valores das variáveis do atributo.
- **valuefromdlg** - Este método transporta os valores dos objetos de interface do diálogo, E e NU, para as variáveis e e nu do atributo corrente.
- **writetprops** - Este método imprime no arquivo de saída corrente os valores das variáveis do atributo, neste caso imprime os valores de e e nu do atributo corrente.

Para configurar o modelador para os demais atributos envolvidos nesta análise (carregamento distribuído e condições de suporte) seguiu-se um procedimento análogo ao que foi apresentado para o atributo material isotrópico.

Com os atributos configurados para o tipo de análise desejado, inicia-se então o processo de simulação propriamente dito com a fase de definição do modelo geométrico e da associação dos atributos às entidades geométricas. A Fig. 5.5 ilustra a definição deste modelo e a criação de um atributo. No caso é o atributo da aresta vertical da direita que está sujeita a restrição de deslocamento na direção x. O diálogo mostrado nesta figura para a captura dos dados relativos ao atributo condição de suporte, referenciado por *Fixity*, foi construído pelo método *dlgdescriptor* associado a este atributo.

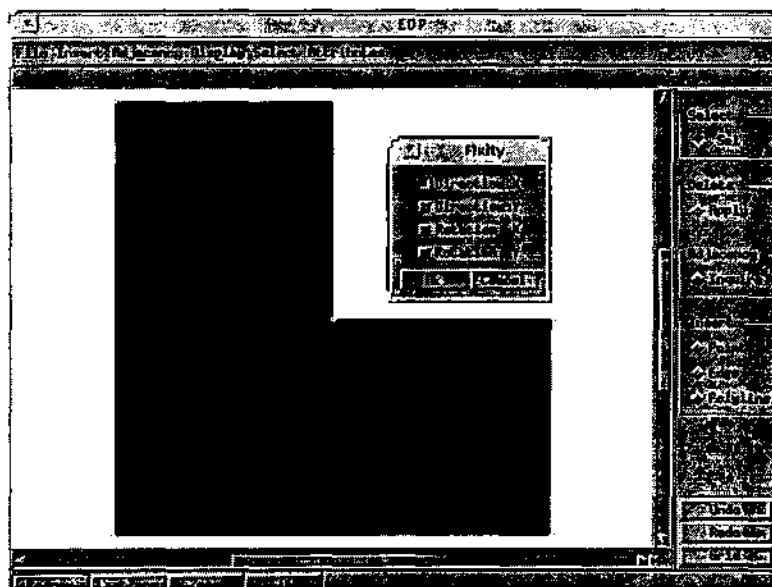


Figura 5.5 - Geração da geometria e especificação dos atributos.

Definido este modelo, o processo tem continuidade com a geração automática da malha inicial (Fig. 5.6) a partir da definição de alguns atributos de malha, como o tipo de elemento a ser utilizado e um parâmetro de controle da discretização do contorno utilizado para a geração da malha. Para este exemplo foram utilizados elementos triangulares lineares com três nós (T3).

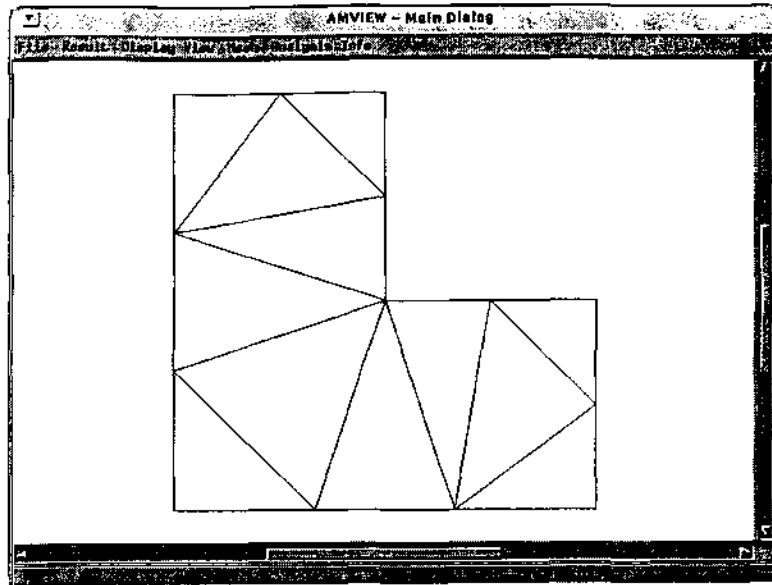


Figura 5.6 - Malha inicial.

Gerado o modelo numérico inicial pelo modelador, através do serviço *EsanCreateNumericalModel* é criada pelo ESAM uma abstração deste modelo no contexto do gerenciador de atributos. Esta abstração é composta por objetos das classes *Node* e *Element* que compõem o modelo numérico gerado. O modelador, então, chama o serviço do ESAM responsável pela geração do arquivo que contém o modelo numérico que será utilizado pelo programa de análise. É mostrado a seguir o método responsável pela gravação em arquivo do modelo numérico criado. Este arquivo é gerado em um formato específico para ser lido pelo módulo que procede a análise numérica. O arquivo correspondente ao modelo numérico inicial gerado pelo modelador é mostrado na Fig. 5.8.

```

function NUMERICAL_MODEL:writetofile(fname)

    local nodemap = self:getnodemap()

    write("\n%NODE\n")
    write(nodemap:getsize(),"i\n") -- imprime número total de nós do modelo.
    write("\n%NODE.COORD\n")
    local ind,node = next(nodemap,nil) -- para cada nó do modelo.
    write(nodemap:getsize(),"i\n")
    while ind do
        write(node:getid(),"i\t") -- imprime o identificador do nó.
        local coords = node:getcoords()
        write(coords.x,"f\t") -- imprime as coordenadas do nó.
        write(coords.y,"f\t")
        write(coords.z,"f\n")
        ind,node = next(nodemap,ind)
    end

    local fixmap = self:getnodewithattrs("Fixity") -- lista nós que possuem o
                                                    -- atributo "Fixity".
    write("\n%NODE.SUPPORT\n")
    write(fixmap:getsize(),"i\n") -- imprime número de nós com o atributo.
    local ind,node = next(fixmap,nil)
    while ind do
        local fix = node:getattrbtype("Fixity")
        write(node:getid(),"i\t") -- imprime identificador do nó.
        local index,support = next(fix,nil)
        while index do
            support:writetofile(fname) -- imprime os valores do atributo
            index,support = next(fix,index)
        end
        n,v = next(fixmap,ind)
    end

    local matmap = self::getelemattrbtype("Material") --lista todos os atributos
                                                    --"Material" utilizados pelos elementos.

    write("\n%MATERIAL\n")
    write(matmap:getsize(),"i\n") -- imprime o número de materiais utilizados
    local n,v = next(matmap,nil) -- para cada material da lista
    while n do
        write(v:getlabel(),"\n") -- imprime o identificador do material
        n,v = next(matmap,n)
    end

    local elemmap = self::getelemmap() -- lista de elementos do modelo
    write("\n%ELEMENT\n")
    write(elemmap:getsize(),"i\n") -- imprime número de elementos do modelo
    write("\n%ELEMENT.T3\n")
    local elemmap = model::getelemstyp("T3") -- lista de elementos do tipo "T3"
    write(elemmap:getsize(),"i\n") -- imprime número de elementos "T3"
    local ind,elem = next(elemmap,nil) -- para cada elemento "T3"da lista
    while ind do
        write(elem:getid(),"i\t") --imprime id do elemento corrente
        local mat = elem:getattrbtype("Material")-- material aplicado no elemento
        write(mat:getid(),"i\t") -- imprime id do material

        ... -- mesmo procedimento para espessura e ordem de integração

        write(elem:getconnect()) -- imprime conectividade do elemento corrente
        ind,elem = next(elemmap,ind)
    end
end

```

Figura 5.7 - Método responsável pela geração do arquivo que contém o modelo numérico.

```

%HEADER
'Exemplo 1 - Analise de tensoes'
%HEADER.ANALYSIS
'Plane_Strain'
%HEADER.FEMOOP.ERROR
0.100000
%NODE
16
%NODE.COORD
16
1      100.00  0.00  0.00
2      75.00  0.00  0.00
3      50.00  0.00  0.00
...
%NODE.SUPPORT
6
1      1      0      0      0      0      0
12     0      1      0      0      0      0
...
%THICKNESS
1
1      1.00
%INTEGRATION.ORDER
1
1      1      1      1      1      1      1
%MATERIAL
1
%MATERIAL.ISOTROPIC
1
1      100000.00      0.30
%ELEMENT
14
%ELEMENT.T3
14
1      1      1      1      1      7      2
2      1      1      1      7      9      2
...
%LOAD
1
1 'Caso Unico'
%LOAD.CASE
1
%LOAD.CASE.LINE.FORCE.UNIFORM
4
10     15     15     7
-1.00  0.00  0.00
13     12     16     7
-1.00  0.00  0.00
...
%END

```

Figura 5.8 - Arquivo contendo as informações sobre modelo numérico inicial.

Gerado o modelo numérico, o processo tem continuidade com a análise numérica do problema, análise de erro e refinamento da malha (Fig. 5.9). Para isto, um outro atributo utilizado (e configurado) é a percentagem de erro admissível para o processo adaptativo. No exemplo foi especificado 10%.

Todas as vezes que um novo modelo numérico for gerado pelo modelador, os serviços ESAM que fazem a criação do modelo numérico abstrato e geração do arquivo para o módulo de análise são chamados, gerando um novo arquivo contendo as informações

relativas ao novo modelo criado. Com o novo modelo numérico é feita uma outra análise e este ciclo continua até que o erro envolvido seja menor que o erro fornecido pelo usuário.

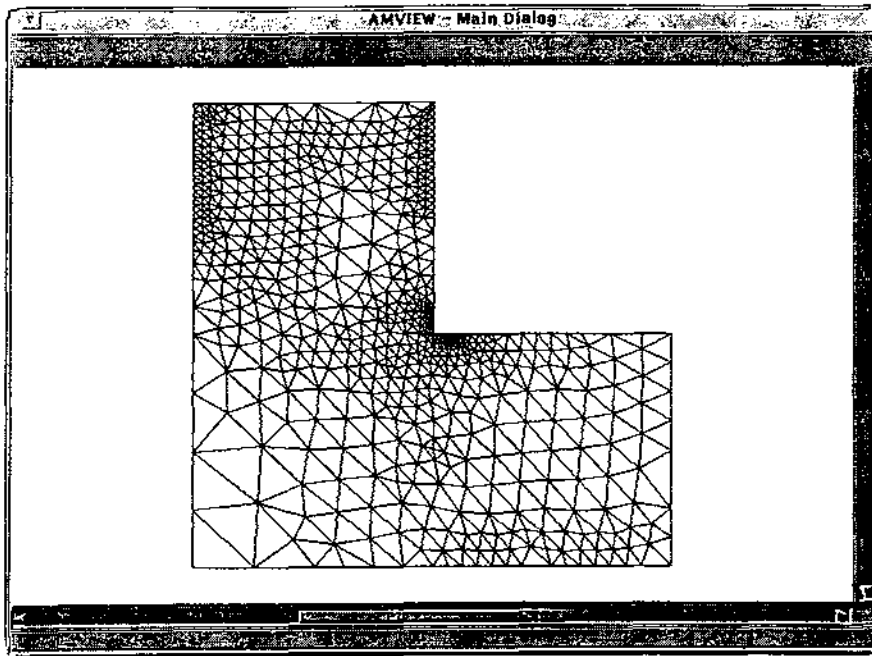


Figura 5.9 - Malha refinada.

Ao final ou a cada passo do processo, pode-se visualizar os resultados da análise dentro do mesmo ambiente onde foi realizada a simulação (Fig. 5.10).

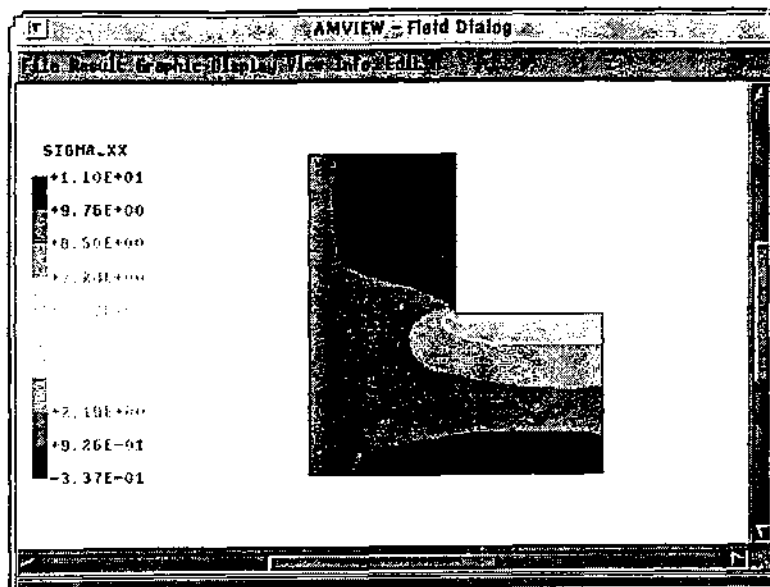


Figura 5.10 - Visualização dos resultados.

5.2 Análise Térmica

O segundo exemplo, mostrado na Fig. 5.11, é uma chapa retangular com material que apresenta condutividade térmica $k = 1.0$ submetida a temperaturas T_1 e T_2 e a um fluxo de calor q indicados.

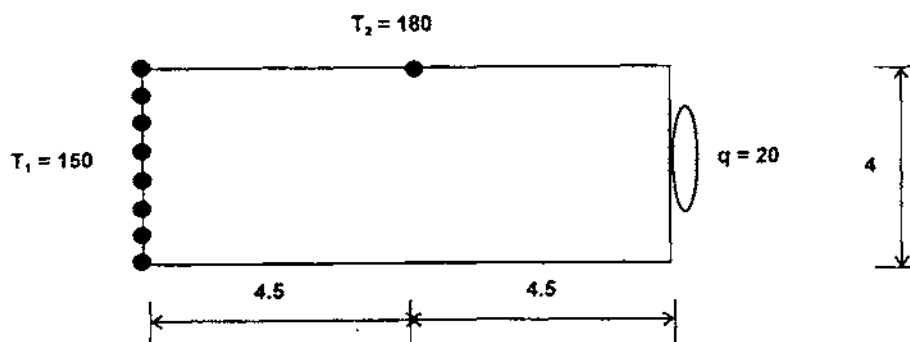


Figura 5.11 - Chapa aquecida.

Para este exemplo pode-se notar que os atributos do modelo são diferentes dos apresentados no exemplo anterior. Os atributos associados a este modelo são os seguintes:

- Material definido pela condutividade térmica.
- Temperaturas prescritas definidas por seus valores escalares.
- Fluxo de calor também definido por seu valor escalar.

Para a configuração destes atributos o procedimento é inteiramente análogo ao utilizado para a configuração dos atributos do exemplo anterior. É mostrado na Fig. 5.12 a configuração do atributo temperatura para sua utilização no processo de simulação. Os métodos implementados foram os mesmos descritos no exemplo anterior sendo que, agora, associados ao novo tipo de atributo.

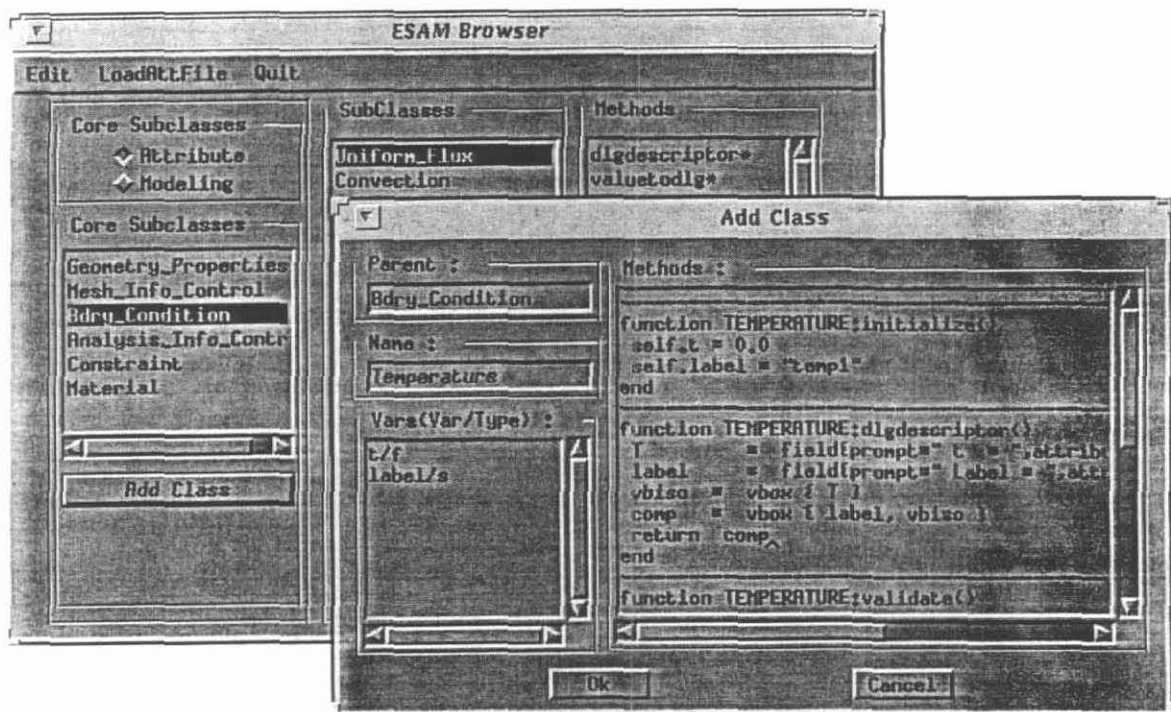


Figura 5.12 - Utilização do *Browser* para criação do atributo temperatura.

O trecho de código implementado com a linguagem de configuração Lua mostrado na Fig. 5.13 foi gerado automaticamente, após a definição (via *Browser*) do atributo

temperatura. A variável T corresponde ao objeto de interface associado ao valor da temperatura do diálogo utilizado para captura dos dados do objeto.

```
TEMPERATURE = crtclass { name="Temperature",
                        parent = BDRY_CONDITION,
                        vars = {"t","label"},
                        ntype = {"f","s"}
                        }

-- Definição dos métodos:
-----
function TEMPERATURE:initialize()
  self.t = 0.00000
  self.label = ""
end
-----
function TEMPERATURE:dlgdescriptor()
  T      = field{prompt=" t = ",attributes='SIZE=75x12'}
  label  = field{prompt=" Label = ",attributes='SIZE=50x12'}
  vbiso  = vbox { T }
  comp   = vbox { label, vbiso }
  return comp
end
-----
function TEMPERATURE:validate()
  if T.nvalue == nil or T.nvalue <= 0 or type(T.nvalue) == string then
    return nil
  end
  return 1
end
-----
function TEMPERATURE:valuetodlg()
  T:set(self.t)
  label:set( self.label)
end
-----
function TEMPERATURE:valuefromdlg()
  self.t = T.nvalue
  self.label = label.value
  self.name = label.value
end
-----
function TEMPERATURE:writetprops( fname )
  write(self.t,"f>10.2\n")
end
-----
```

Figura 5.13 - Código gerado através do *Browser* para a configuração do atributo temperatura.

O mesmo procedimento foi utilizado para configuração dos outros atributos associados a este tipo de análise (fluxo de calor e material).

Configurados os atributos para este tipo de simulação, é iniciado o processo de simulação a partir da geração do modelo geométrico e a associação dos atributos às entidades geométrica. Isto é ilustrado na Fig. 5.14 que mostra a criação do atributo temperatura que será associado ao vértice central da borda superior do modelo.

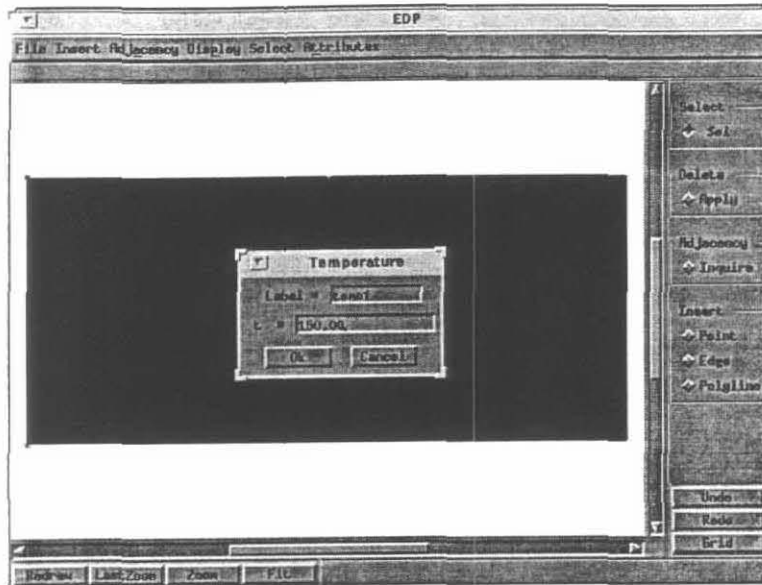


Figura 5.14 - Modelo geométrico e especificação dos atributos.

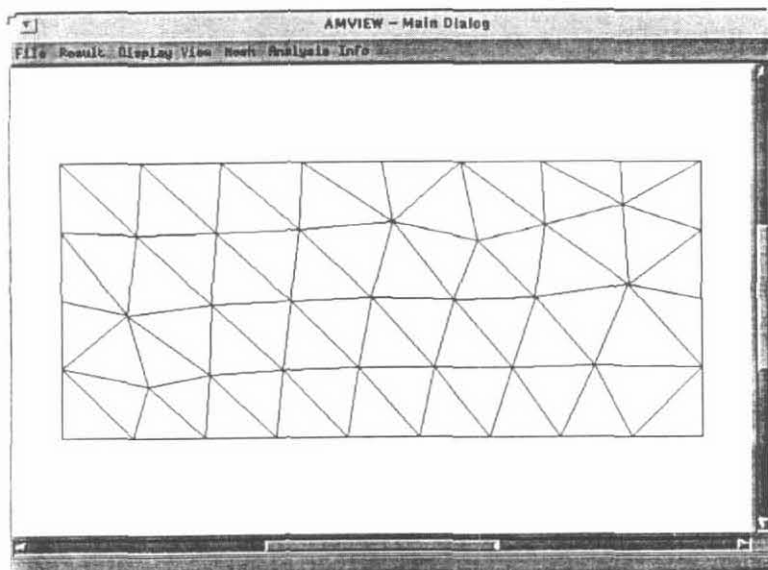


Figura 5.15 - Malha inicial.

Definido o modelo geométrico, o sistema realiza a geração automática da malha inicial (Fig. 5.15). Gerada a malha o modelador informa ao ESAM que um novo modelo foi criado, e este por sua vez procede com a criação de uma representação deste modelo gerando o arquivo, mostrado na Fig.5.16 , que será lido pelo módulo de análise.

```

%HEADER
'Exemplo 2 - Analise de termica'
%HEADER.TITLE
'Chapa submetida a fluxo de calor e temperatura'
%HEADER.ANALYSIS
'Temperature2d'
%HEADER.FEMOOP.ERROR
0.100000
%NODE
12
%NODE.COORD
12
1      4.50   4.00   0.00
2      6.75   4.00   0.00
3      9.00   4.00   0.00
.      .      .
.      .      .
.      .      .
12     0.00   2.00   0.00
%THICKNESS
1
1      1.00
%INTEGRATION.ORDER
1
1      1      1      1      1      1      1
%MATERIAL
1
%MATERIAL.PROPERTY.THERMAL.CONDUCTIVITY
1
1      1
%ELEMENT
10
%ELEMENT.T3
10
1      1      1      1      1      7      2
2      1      1      1      1      11     7
.      .      .      .      .      .      .
.      .      .      .      .      .      .
10     1      1      1      4      3      2
%LOAD.CASE.NODAL.TEMPERATURE
4
9      150.00
12     150.00
10     150.00
1      180.00
%LOAD.CASE.LINE.HEAT.FLUX.UNIFORM
2
10     4      3      20.00
9      5      4      20.00
%END

```

Figura 5.16 - Arquivo com as informações sobre o modelo numérico inicial.

O arquivo mostrado na Fig. 5.16 foi gerado através do mesmo método mostrado no exemplo anterior (Fig. 5.7), adicionando as informações referentes aos novos atributos.

A partir deste arquivo o módulo de análise (FEMOOP) realiza a análise numérica do modelo dado. De posse dos resultados obtidos o modelador verifica se o erro obtido foi menor que o erro admitido pelo usuário. Caso isto não tenha acontecido, é gerado um novo modelo numérico com a malha devidamente refinada (Fig. 5.17) procedendo-se com uma nova análise. Este ciclo continua até que o erro obtido seja menor que o admitido pelo usuário.

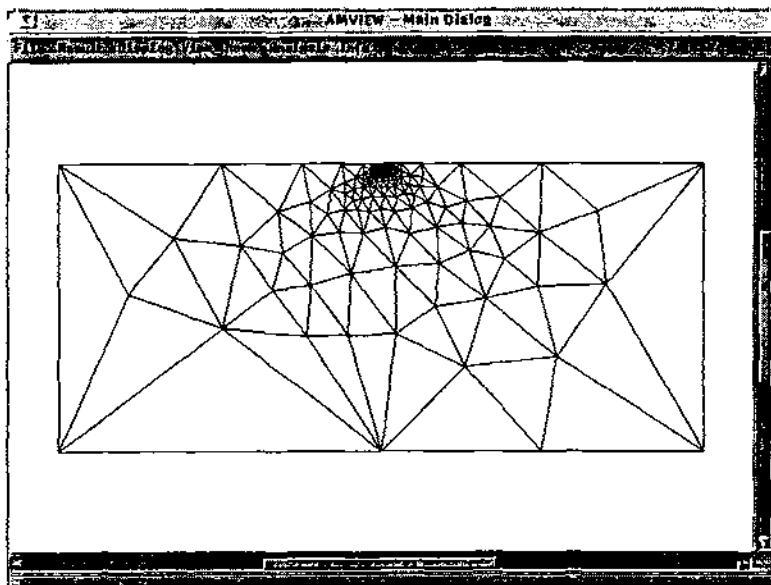


Figura 5.17 - Refinamento da malha.

Novamente, a cada passo de análise ou ao final do processo pode-se visualizar graficamente o modelo e os resultados obtidos pelo módulo de análise (Fig. 5.18).

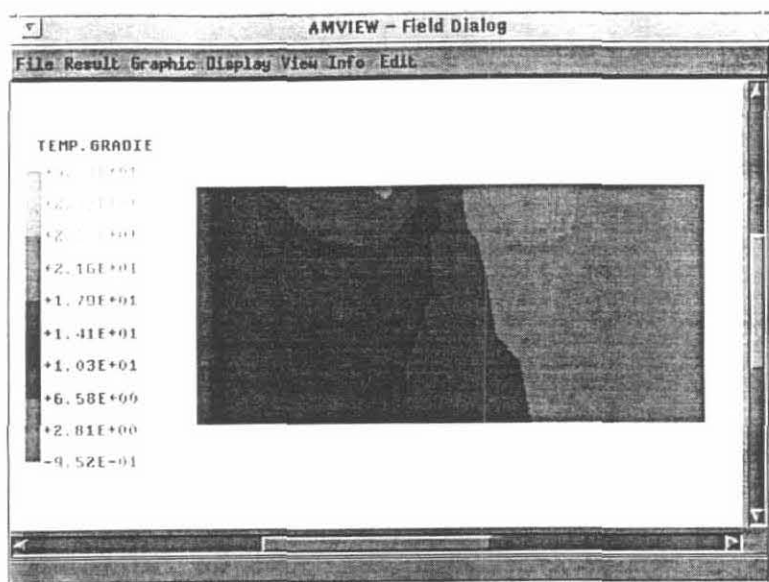


Figura 5.18 - Visualização dos resultados.

Este exemplo ilustra bem o poder de modelagem do sistema apresentado, visto que foi possível a realização de um outro tipo de simulação através da simples configuração dos atributos associados ao tipo de análise desejada.

Este trabalho apresenta um sistema extensível e configurável para simulação adaptativa de problemas bidimensionais de mecânica computacional pelo método dos elementos finitos. O sistema possibilita a configuração dos atributos a serem utilizados de acordo com o tipo de simulação desejada. Esta configuração é feita pelo usuário através de uma linguagem interpretada relativamente simples, permitindo a um usuário com algum conhecimento definir, em tempo de utilização, os tipos de atributos específicos ao problema a ser analisado.

O sistema possibilita a realização de todas as etapas envolvidas no processo de simulação adaptativa bidimensional. Ele contém um modelador geométrico que permite ao usuário a criação da geometria do modelo através de uma interface gráfica amigável. A partir da geometria criada o sistema procede com a geração automática da malha inicial que será analisada pelo módulo de análise numérica oferecido pelo sistema. Como se trata de uma simulação adaptativa, a partir da geometria, da malha inicial e dos resultados obtidos na análise, o sistema tem capacidade de decidir onde refinar a malha para que uma nova análise seja feita com a obtenção de melhores resultados. Este ciclo continua até que um critério de convergência pré-estipulado seja atingido.

Dois exemplos de aplicação do sistema são mostrados. O primeiro para simulação adaptativa de um problema de elasticidade plana e o segundo de um problema de difusão térmica.

A comunicação com o módulo de análise de elementos finitos é genérica, permitindo que outros programas possam ser utilizados para este fim. No presente trabalho, foi utilizado um módulo baseado em programação orientada a objetos.

6.1 Principais contribuições e avaliação dos resultados

Foram dois os principais objetivos deste trabalho. O primeiro foi a implementação da arquitetura proposta por Carvalho (1995) para um sistema extensível de configuração de atributos em mecânica computacional. O segundo foi a utilização deste sistema para o desenvolvimento de uma aplicação configurável para simulação adaptativa bidimensional em mecânica computacional.

São apresentadas a seguir as principais contribuições deste trabalho:

- A implementação de um sistema extensível para configuração de atributos em mecânica computacional, proposto por Carvalho (1995), que pode ser integrado com qualquer modelador sem que haja alteração no restante da aplicação.
- A utilização do sistema citado no item anterior em uma aplicação, permitindo a aferição do sistema implementado.
- A integração de um sistema para simulação adaptativa bidimensional, resultado de diversos trabalhos desta linha de pesquisa, com o sistema extensível para configuração de atributos resultando num sistema configurável para simulação adaptativa de mecânica computacional.

O sistema configurável para simulação adaptativa obtido com esta integração permite a configuração dos atributos a serem utilizados de acordo com o tipo de análise desejada. Esta configuração é feita de uma forma relativamente fácil e rápida, permitindo ao

usuário configurador o acesso à tecnologia utilizada pela aplicação sem que este conheça os detalhes desta tecnologia.

A integração do sistema ESAM com o sistema para simulação adaptativa permitiu a simplificação do código que implementa a aplicação, facilitando o seu entendimento, visto que o gerenciamento de atributos que era feito pela própria passou a ser inteiramente feito pelo sistema ESAM.

Os resultados obtidos com a integração do ESAM com o sistema para simulação adaptativa bidimensional de mecânica computacional foram extremamente satisfatórios.

6.2 Sugestões para trabalhos futuros

De uma forma geral, existem duas linhas principais a serem seguidas com a finalização deste trabalho. A primeira diz respeito ao sistema extensível implementado e a segunda é relativa ao sistema para simulação adaptativa bidimensional de mecânica computacional. Como este trabalho apenas utilizou a estratégia de adaptatividade proposta por Cavalcante (1994), as sugestões para trabalhos futuros só se referem ao sistema de configuração de atributos. Neste contexto, existem alguns pontos a serem discutidos e implementados:

- Resolver o problema do *feedback* (exibição gráfica) do atributo, uma vez que todas as informações a respeito de atributos do modelo são feitas pelo ESAM e este não possui nenhum conhecimento sobre o sistema gráfico do modelador ou sobre os objetos da interface onde o atributo deve ser desenhado, enquanto que a aplicação que gerencia toda a parte gráfica não tem conhecimento sobre os atributos que devem ser desenhados.

- Oferecer ao configurador um conjunto mais abrangente de classes de atributos que englobem a maior parte dos atributos utilizados em mecânica computacional, com o objetivo de oferecer ao configurador um conjunto de classes em que todos os atributos que ele precise configurar sejam derivados de algumas destas classes.
- Verificar a eficiência dos módulos que estão implementados. Alguns módulos do ESAM, por exemplo, que estão implementados em Lua e que não serão acessíveis ao usuário configurador podem ser implementados em C com o objetivo de melhorar a eficiência do sistema.
- Buscar a especificação dos atributos através de expressões escritas em termos de qualquer parâmetro. Os parâmetros podem ser simples variáveis, outros atributos, funções escritas pelo usuário, funções matemáticas, etc.
- Melhorar o *Browser* com algoritmos mais eficientes de busca e a implementação de mecanismo para *undo* são outros pontos que podem ser tratados.

Referências Bibliográficas

- [Carvalho 1995] Carvalho, M.T.M., “Uma Estratégia para o Desenvolvimento de Aplicações Configuráveis em Mecânica Computacional”, Tese de Doutorado, PUC-Rio, Departamento de Engenharia Civil, 1995.
- [Cavalcante 1994] Cavalcante Neto, J.B., “Simulação Auto-Adaptativa Baseada em Enumeração Espacial Recursiva de Modelos Bidimensionais de Elementos Finitos”. Dissertação de Mestrado, PUC-Rio, Departamento de Engenharia Civil, 1994.
- [Cavalcanti 1992] Cavalcanti, P.R., “Criação e Manutenção de Subdivisões do Espaço”, Tese de Doutorado, PUC-Rio, Departamento de Informática, 1992.
- [Celes et al. 1995a] Celes Filho, W., de Figueiredo, L.H., Gattass, M., “Sistema EDG - Manual de Programação”, TeCGraf, PUC-Rio, 1995.
- [Celes et al. 1995b] Celes Filho, W., de Figueiredo, L.H., Ierusalimschy R., “Programando em Lua - Teoria e Prática”, TeCGraf, PUC-Rio, 1985.

- [Celes *et al.* 1995c] Celes Filho, W., de Figueiredo, L.H., Gattass, M., “EDG: uma ferramenta para criação de interfaces gráficas interativas”, artigo submetido ao VIII SIBGRAPI, 1995.
- [Cook 1989] Cook, R.D., Malkus, D.S e Plesha, M.E. - Concepts and Applications of Finite Element Analysis, John-Wiley, New York - NY, 1989.
- [Cox-Novobilsky 1991] Cox, B.J. e Novobilsky, Object-Oriented Programming: An Evolutionary Approach, Addison-Wesley, Publishing Company, 2nd Edition, 1991.
- [Ferraz 1993] Ferraz, M.F.R., “Reconstituição de Seções Geológicas Utilizando Subdivisões Planares, Transformações Geométricas e Computação Gráfica Interativa”, Dissertação de Mestrado, PUC-Rio, Departamento de Informática, 1993.
- [Figueiredo *et al.* 1994] Figueiredo, L.H. de, Ierusalimschy, R. e W. Celes Filho, “The design and implementation of a language for extending applications”, Anais do XXI Semish, 273-283, 1994.
- [Finnigam *et al.* 1989] Finnigam, P.M., A. Kela e J.E. Davis, “Geometry as a Basis for Finite Element Automation”, Engineering with Computers 5 (1989) 147-160.
- [Goldberg 1983] Goldberg, A., D. Robson, SMALLTALK-80 *The Language and its Implementation*, Addison, Wesley, Reading, Massachusetts, 1983.

- [Guimarães 1992] Guimarães, L.G.S. - “Disciplina de Orientação a Objetos para Análise e Pós-processamento Bidimensional de Elementos Finitos”, Dissertação de Mestrado, PUC-Rio, Departamento de Engenharia Civil, 1992.
- [Ierusalimschy *et al.* 1994] Ierusalimschy R., L.H de Figueiredo, W. Celes Filho, “Reference manual of the programming language Lua”, Monografias em Ciência da Computação **4/94**, PUC-Rio, Departamento de Informática, 1994.
- [Levy 1993] Levy, C.H., “IUP/LED: uma ferramenta portátil de interface com o usuário”, Dissertação de Mestrado, PUC-Rio, Departamento de Informática, 1993.
- [Mäntylä 1988] Mäntylä, M. *An Introduction to Solid Modeling Computer*, Science Press, 1988.
- [Martha 1989] Martha, L.F., “Topological and Geometrical Modeling Approach to Numerical Discretization and Arbitrary Fracture Simulation in Three-Dimensions”, Ph.D. Dissertation, Cornell University, Ithaca, NY, 1989.
- [Martha 1993] Martha, L.F., Carvalho, P.C.P., Celes F. e Ferraz, M., “Gerenciamento de Subdivisões do Plano Usando HED”, Caderno de Ferramentas do VII Simpósio Brasileiro de Engenharia de Software, PUC-Rio/SBC, 51-52 (1993).
- [Ousterhout 1994] Ousterhout, J.K., *Tcl and Tk toolkit*, Addison-Wesley, 1994.

- [Preparata 1985] Preparata, F.P. e M. Ian Shamos, Computational Geometry - an introduction, Springer-Verlag, New York, 1990.
- [Samet 1984] Samet, H. - "The Quadtree and Related Hierarchical Data Structures", ACM Computer Surveys, 16 (02) ,1984.
- [Shepard e Finnigam 1988] Shepard, M.S. e P.M. Finnigam, "Integration of Geometric Modeling and Advanced Finite Element Preprocessing", Finite Elements in Analysis and Design 4 (1988) 147-162.
- [TeCGraf 1989] Manual de Referência do GKS/PUC, TeCGraf, PUC-Rio, 1995.
- [Vianna 1992] Vianna, A.C. - "Modelagem Geométrica Estendida para Modelos Bidimensionais de Elementos Finitos", Dissertação de Mestrado, PUC-Rio, Departamento de Engenharia Civil, 1992.

Apêndice A

Descrição das “Core Classes”

A.1 Classe *Attribute*

Nome da classe: *Attribute*

Superclasse: *Core*

Variáveis:

- *name* - Contém o nome da classe.
 - *parent* - Contém a superclasse desta classe.
 - *label* - Identificador, cadeia de caracteres, que referencia o objeto.
 - *id* - Identificador, número inteiro, que referencia o objeto.
 - *entities* - Lista de entidades geométricas que contém o objeto atributo.
 - *dialog* - Objeto referente ao diálogo para a captura de dados do objeto.
 - *dim* - Dimensão do atributo.
-

Métodos:

- *New ()* - Este método cria um objeto da classe, inicializa as variáveis e retorna o valor *nil* se a operação não for bem sucedida.
 - *Delete ()* - Este método elimina o objeto corrente.
 - *ShowDialog ()* - Este método cria o diálogo para a captura dos dados do objeto.
 - *Validate ()* - Este método é associado ao diálogo do objeto e procede a validação ou não dos dados do objeto. Como *default* este método valida todos os diálogos retornando o valor 1.
 - *AppendEntity (entity)* - Este método coloca o objeto *entity* da classe *Geometry* (passado como parâmetro) na lista de entidades do objeto.
 - *RemoveEntity (entity)* - Este método elimina o objeto *entity* da classe *Geometry* da lista de entidades do objeto.
 - *ShowEntities ()* - Método disparado por um serviço oferecido pelo sistema que informa a aplicação quais as entidades às quais foi associado o objeto atributo corrente.
 - *SetId (id)* - Este método preenche o campo *id* do objeto corrente com o valor passado como parâmetro.
 - *GetId ()* - Este método retorna o identificador *id* do objeto corrente.
 - *ValueToDlg ()* - Este método é responsável pelo transporte das variáveis do objeto para os objetos de interface associados
 - *ValueFromDlg (id)* - Este método é responsável pelo transporte das variáveis dos objetos de interface para as variáveis do objeto.
 - *GetLabel ()* - Este método retorna o *label* do objeto corrente.
-

A.2 Classe *Model*

Nome da classe: *Model*

Superclasse: *Modeling*

Variáveis:

- **name** - Nome da classe.
 - **parent** - Campo contendo a superclasse desta classe.
 - **attrlist** - Lista contendo todos os atributos do modelo. Esta lista é indexada pelo *label* do atributo e contém o objeto da classe *Attribute* relativo ao atributo criado.
 - **toplist** - Lista contendo todos os objetos geométricos do modelo. Esta lista é indexada pelo identificador da entidade geométrica e contém o objeto da classe *Geometry* relativo a entidade geométrica criada.
 - **spcdlg** - Este campo contém o objeto referente ao diálogo principal para a especificação dos atributos oferecidos pelo sistema.
 - **attdlg** - Este campo contém o objeto referente ao diálogo principal para o serviço de especificação do atributo corrente oferecido pelo sistema.
 - **inqdlg** - Este campo contém o diálogo principal para o serviço de inquire oferecido pelo sistema.
 - **curatt** - Este campo contém o atributo corrente do modelo.
-

Métodos:

- **New ()** - Cria um objeto da classe, inicializa as variáveis retornando o objeto criado.
- **Delete ()** - Elimina o objeto corrente.
- **GetAttribute (attribute)** - Dado o *label* ou o *id* (identificador) do atributo, percorre-se a lista de atributos do modelo e retorna o atributo caso o mesmo exista. Caso contrário retorna *nil*.
- **GetNAttribs ()** - Retorna o número de atributos do modelo.
- **GetTopObj (identity)** - Dado o identificador da entidade geométrica, retorna a entidade, caso ela exista. Caso contrário retorna *nil*.
- **HasTopEntity (identity)** - Dado o identificador da entidade geométrica., verifica se já existe na lista. Caso exista, retorna 1, caso contrário retorna *nil*.
- **CreateGeomEntity (identity)** - Dado o identificador da entidade geométrica, verifica se existe atributo corrente, caso exista, cria o objeto geométrico associado a entidade, coloca-o na lista de entidades geométricas do modelo. E aplica o atributo corrente na entidade criada.
- **DeleteGeomEntity (identity)** - Dado o identificador da entidade geométrica, verifica se a mesma já existe. Caso exista, remove-a da lista. Atualiza-se a lista de entidades geométricas do modelo e atualiza as listas de entidades dos atributos do modelo.
- **CheckSplitGeomEntity (entobj)** - Este método verifica se é possível ou não a divisão da entidade geométrica dada. Caso não seja possível, retorna o valor *nil*.
- **CheckJoinGeomEntity (entobj1, entobj2)** - Este método verifica se é possível ou não a união das duas entidades geométricas dadas. Caso não seja possível, retorna o valor *nil*.
- **SplitGeomEntity (identold, identnew1, identnew2)** - Este método recebe como parâmetros os identificadores da entidade a ser dividida e os identificadores das duas novas entidades a serem criadas e executa a divisão da entidade, removendo-a da lista e adicionando as duas novas entidades.

- **JoinGeomEntity (identoid1, identoid2, identnew)** - Este método recebe como parâmetros os identificadores das entidades a serem unidas e o identificador da nova entidade a ser criada e executa a junção das entidades, removendo as duas entidades antigas da lista e adicionando a nova entidade criada.
 - **SetCurAttribute (attobj)** - Dado um objeto da classe *Atributo*, coloca-o como o atributo corrente do modelo.
 - **CreateAttribute ()** - Este método dispara um diálogo para a criação de um atributo. É disparado por um serviço da aplicação.
 - **ShowAttribsDlg ()** - Mostra o diálogo *attdlg* do modelo, para seleção do atributo corrente. Este método é disparado por um dos serviços oferecidos pelo ESAM.
 - **AttachAttribs ()** - Método associado ao serviço responsável pela aplicação do atributo corrente à uma entidade geométrica.
 - **SpecifyAttribs ()** - Mostra o diálogo *spcdlg* do modelo. Este método é disparado por um dos serviços do ESAM.
 - **ReadTopFromFile (fname)** - Dado o nome do arquivo, lê a parte relativa aos objetos topológicos do arquivo de persistência.
 - **ReadAttrFromFile (fname)** - Dado o nome do arquivo, lê a parte relativa aos atributos do arquivo de persistência.
 - **WriteTopToFile ()** - Dado o nome do arquivo, este método escreve a parte referente aos objetos geométricos no arquivo de persistência.
 - **WriteAttrToFile ()** - Dado o nome do arquivo, este método escreve a parte referente aos atributos no arquivo de persistência.
 - **Attach (entlist)** - Dada uma lista de entidades geométricas selecionadas, verifica se existe atributo corrente no modelo. Caso não exista exibe uma mensagem informando ao usuário. Caso exista, percorre-se toda a *entlist*, cria-se as entidades geométrica associadas adicionando-as à lista de entidades do modelo. Caso a dimensão da entidade geométrica seja compatível com a dimensão do atributo corrente e aplica o atributo corrente na entidade criada e adiciona a entidade na lista de entidades do atributo.
 - **Detach (entlist)** - Dada uma lista de entidades geométricas selecionadas, verifica se existe atributo corrente no modelo. Caso não exista exibe uma mensagem informando ao usuário. Caso exista, percorre-se toda a *entlist*, pega a lista de atributos da entidade, verifica se o atributo corrente está contido nessa lista, e caso esteja, retira o atributo da mesma. Atualiza as listas de atributos da entidade e a lista de entidades do atributo.
 - **Modify ()** - Mostra o diálogo relativo ao atributo corrente com os valores dos campos associados ao atributo para modificação do(s) mesmo(s). Sua validação indica que o atributo foi modificado.
 - **Delete ()** - Retira o atributo corrente da lista de atributos do modelo, caso o mesmo não esteja aplicado em uma entidade geométrica. Se estiver aplicado, pergunta ao usuário se ele quer apagar o atributo, o que implica na remoção do mesmo de todas as lista de atributos das entidades geométricas.
 - **ShowInquireDlg ()** - Mostra o diálogo de inquire do modelo (*inqdlg*). Método disparado por um serviço do ESAM.
 - **ShowAttributesInquire (entlist)** - Recebe a lista de entidades para o inquire e manda mensagem de inquire para cada entidade geométrica da lista.
 - **ShowEntityAttr ()** - Com o valor do *label* do atributo da lista de atributos do diálogo de inquire e manda uma mensagem de inquire para o objeto da classe atributo exibir seus valores.
 - **ShowAttrTypes ()** - Captura do diálogo de inquire o tipo de atributo desejado. E a partir do tipo, constrói e mostra um diálogo com todos os atributos, daquele tipo, que foram criados.
-

A.3 Classe *Geometry*

Nome da classe: *Geometry*

Superclasse: *Modeling*

Variáveis:

- *name* - Nome da classe.
 - *parent* - Campo contendo a superclasse desta classe.
 - *attribs* - Lista de atributos (objetos da classe *Attribute*) associados a entidade geométrica ;
 - *id* - Identificador, número inteiro, que referencia o objeto.
-

Métodos:

- *New ()* - Este método cria um objeto da classe e inicializa suas variáveis retornando o objeto.
 - *Delete ()* - Este método elimina o objeto corrente.
 - *WriteAttribs ()* - Escreve sequencialmente os identificadores dos atributos aplicados à entidade geométrica no arquivo corrente.
 - *GetAttribs ()* - Retorna a lista de atributos aplicados ao objeto corrente.
 - *GetDimension ()* - Retorna a dimensão do objeto corrente.
 - *Attach (attribute)* - Recebe o objeto atributo, verifica se ele já existe na lista de atributos da entidade. Caso ainda não exista coloca-o na lista.
 - *RemoveAttribute (attribute)* - Recebe o objeto atributo, verifica se o atributo existe na lista de atributos do objeto corrente. Se existir remove-o da lista.
 - *GetAttrbsType (attrtype)* - Dado o tipo do atributo retorna uma lista com todos os atributos daquele tipo que estão aplicados ao objeto corrente. Caso não existam atributos do referido tipo, retorna nil.
 - *ShowAttribs ()* - Verifica se existe atributos aplicados ao objeto corrente, se existir monta um diálogo com os *labels* dos atributos existentes e mostra-o para escolha do atributo a ser consultado.
-

A.4 Classe *Numerical Model*

Nome da classe: *Numerical_Model*

Superclasse: *Modeling*

Variáveis:

- *name* - Nome da classe.
 - *parent* - Campo que contém a superclasse desta classe.
 - *nodemap* - Lista contendo os objetos da classe *Node* que compõem o modelo numérico.
 - *elemmap* - Lista contendo os objetos da classe *Element* que compõem o modelo numérico.
 - *datdlg* - Objeto referente ao diálogo para a captura dos dados necessários à criação de um modelo numérico.
-

Métodos:

- **New ()** - Cria objeto da classe, coloca o objeto criado como objeto numérico corrente do modelo e retorna o objeto criado.
 - **Delete ()** - Elimina o objeto corrente.
 - **Initialize ()** - Inicializa o objeto corrente.
 - **ShowDatDlg ()** - Mostra o diálogo para captura dos dados relativos ao objeto corrente do modelo.
 - **GetNodeMap ()** - Retorna a lista de nós do objeto corrente.
 - **GetElemMap ()** - Retorna a lista de elementos do objeto corrente.
 - **GetElemWithAttributes (attrtype)** - Retorna uma lista com todos os elementos do objeto corrente que possuem o tipo *attrtype*.
 - **GetNodeWithAttributes (attrtype)** - Retorna uma lista com todos os nós do objeto corrente que possuem o tipo *attrtype*.
 - **GetElemAttribsType (attrtype)** - Retorna uma lista com todos os atributos do tipo *attrtype* que estão aplicados nos elementos do objeto corrente.
 - **GetNodeAttribsType (attrtype)** - Dado o tipo do atributo, retorna uma lista com todos os atributos do tipo *attrtype* que estão aplicados nos nós do objeto corrente.
-

A.5 Classe *Node*

Nome da classe: *Node*

Superclasse: *Modeling*

Variáveis :

- *id* - Identificador do objeto corrente.
 - *top* - Objeto geométrico associado ao objeto corrente.
-

Métodos :

- **New ()** - Este método cria objeto da classe e retorna o objeto criado.
 - **Delete ()** - Este método elimina o objeto corrente.
 - **SetId (id)** - Este método recebe o identificador *id* associado ao objeto corrente e coloca-o no campo *id* do objeto corrente.
 - **GetId ()** - Retorna o identificador do objeto corrente.
 - **SetTopEntity (topentity)** - Este método recebe o objeto geométrico associado ao nó e preenche o campo do nó que armazena esta informação.
 - **GetCoords ()** - Este método retorna as coordenadas do objeto corrente..
 - **GetAttribsType (type)** - Este método recebe o tipo do atributo e retorna uma lista com todos os atributos do tipo dado associado ao objeto.
-

A.6 Classe *Element*

Nome da classe: *Element*

Superclasse: *Modeling*

Variáveis:

- *name* - Campo contendo o nome da classe.
 - *parent* - Campo que contém a classe-base desta classe.
 - *id* - Identificador do objeto corrente.
 - *connect* - Conectividade do objeto corrente.
-

Métodos da classe:

- *New ()* - Este método cria objeto da classe e retorna o objeto criado.
 - *Delete ()* - Este método elimina o objeto corrente.
 - *SetId (id)* - Este método recebe o identificador *id* e coloca-o no campo do objeto corrente que armazena esta informação.
 - *GetId ()* - Retorna o identificador do objeto corrente.
 - *GetConnect ()* - Este método recebe o objeto geométrico associado ao nó e preenche o campo *top* do objeto corrente.
 - *GetAttribsType (atttype)* - Este método retorna as coordenadas do objeto corrente.
 - *GetFeaturesWithAttribs (type)* - Este método recebe o tipo do atributo e retorna uma lista com todos os objetos da classe *Element Feature* associados ao objeto corrente.
-

A.7 Classe *Element Feature*

Nome da classe: *Element_Feature*

Superclasse: *Modeling*

Variáveis:

- *elem* - Objeto da classe *Element* associado ao objeto corrente.
 - *top* - Objeto da classe *Geometry* associado ao objeto corrente.
 - *nodes* - Tabela contendo os nós (objetos da classe *Node*) associados ao objeto corrente.
-

Métodos da classe:

- *New ()* - Este método cria um objeto da classe e retorna o objeto criado.
- *Delete ()* - Este método elimina o objeto corrente.
- *SetElement (elemobj)* - Este método coloca o objeto, *elemobj* (classe *Element*), no campo *elem* do objeto corrente.

- **GetElement ()** - Retorna o campo *elem* do objeto corrente.
 - **SetTopEntity (topobj)** - Este método recebe um objeto topológico (objeto da classe *Geometry*) associado ao objeto corrente.
 - **GetTopEntity ()** - Este método retorna o objeto topológico associado ao objeto corrente.
 - **SetNodes (nodes)** - Este método recebe uma tabela de nós (objetos da classe *Node*) e adiciona cada um desses nós a tabela de nós do objeto corrente.
 - **GetNodes ()** - Retorna a tabela de nós associada ao objeto corrente.
 - **GetConnect ()** - Este método retorna a conectividade do objeto corrente.
 - **GetAttribsType (attype)** - Este método recebe uma cadeia de caracteres contendo um tipo de atributo e retorna uma lista contendo todos os atributos deste tipo associados ao objeto corrente.
-

Apêndice B

Descrição das Funções MIS

- **MisRegGeometryEntities (void (*register_entities)(char *name, int type, int dim))**

Descrição: esta função recebe uma outra função como parâmetro que deve ser chamada pela aplicação para registrar todos os tipos de entidades geométricas utilizadas pela aplicação. A função passada como parâmetro deve ser chamada passando os parâmetros mostrados abaixo.

Parâmetros:

name - cadeia de caracteres contendo o nome da classe que será criada pelo ESAM associada a um determinado tipo de entidade geométrica. (Entra)

type - este parâmetro é o identificador (representado por um inteiro) no contexto do modelador. (Entra)

dim - indica a dimensão da entidade geométrica que será utilizada pelo ESAM para determinar a superclasse da classe que vai ser criada. (Entra)

- **MisGetNSelect (int n, int *ent, int *type)**

Descrição: esta função deve informar ao ESAM quais as entidades do modelador que estão selecionadas em um determinado instante. Estas informações são passadas através dos parâmetros mostrados abaixo.

Parâmetros:

n - número de entidades selecionadas. (Sai)

ent - vetor de inteiros contendo os identificadores de todas as entidades que estão selecionadas. (Sai)

type - vetor de inteiros contendo os tipos das respectivas entidades selecionadas armazenada no vetor de entidades. (Sai)

- **MisGetSelect (int *ent, int *type)**

Descrição: esta função difere da função anterior apenas no número de entidades selecionadas, visto que esta função passa informações apenas de uma única entidade que está selecionada em um dado instante.

Parâmetros:

ent - inteiro contendo o identificador da entidade selecionada. (Sai)

type - inteiro que representa o tipo da entidade selecionada. (Sai)

- **MisGetNInquire (int n, int *ent, int *type)**

Descrição: esta função deve informar ao ESAM quais as entidades do modelador que estão selecionadas para consulta em um determinado instante. Estas informações são passadas através dos parâmetros mostrados abaixo.

Parâmetros:

n - número de entidades selecionadas. (Sai)

ent - vetor de inteiros contendo os identificadores de todas as entidades que estão selecionadas. (Sai)

type - vetor de inteiros contendo os tipos das respectivas entidades selecionadas armazenada no vetor de entidades. (Sai)

- **MisGetInquire (int *ent, int *type)**

Descrição: esta função difere da função anterior apenas no número de entidades selecionadas para consulta, visto que esta função passa informações apenas de uma única entidade que está selecionada em um dado instante.

Parâmetros:

ent - inteiro contendo o identificador da entidade selecionada para consulta. (Sai)

type - inteiro que representa o tipo da entidade selecionada para consulta. (Sai)

-
- **MisRegElementTypes (void (*register_element)(char *n, int nnos, int type, int interp))**

Descrição: esta função recebe uma outra função como parâmetro que deve ser chamada pela aplicação para registrar todos os tipos de elementos finitos utilizados pela aplicação. A função passada como parâmetro deve ser chamada passando os parâmetros mostrados abaixo.

Parâmetros:

n - cadeia de caracteres contendo o nome da classe que será criada pelo ESAM associada a o determinado tipo de elemento. **(Entra)**

nnos - número de nós do elemento finito. **(Entra)**

type - este parâmetro é o identificador do tipo de elemento no contexto do modelador. **(Entra)**

interp - ordem do polinômio de interpolação associado ao elemento finito em questão. **(Entra)**

- **MisGetNodeList (int *nnode, int **nodelist)**

Descrição: nesta função é preenchido um vetor contendo os identificadores dos nós existentes no modelo numérico.

Parâmetros:

nnode - número de nós existentes. **(Sai)**

nodelist - vetor contendo os identificadores dos nós. **(Sai)**

- **MisGetElementList (int *nelem, int **elemlist, int **type, int **nnos)**

Descrição: nesta função é preenchido um vetor contendo os identificadores dos elementos existentes no modelo numérico.

Parâmetros:

nelem - número de elementos existentes no modelo numérico. **(Sai)**

elemlist - vetor contendo os identificadores dos elementos. **(Sai)**

type - vetor contendo os tipos dos respectivos elementos do vetor *elemlist*. **(Sai)**

nnos - vetor contendo o número de nós dos respectivos elementos do vetor *elemlist*. **(Sai)**

- **MisGetNodeCoords (int *node*, int **n*, float **coords*)**

Descrição: esta função retorna através do vetor *coords* as coordenadas do nó cujo identificador é passado como parâmetro.

Parâmetros:

node - identificador do nó. (Entra)
n - número de coordenadas. (Sai)
coords - vetor contendo as coordenadas do nó em questão. (Sai)

- **MisGetElemConnect (int *elem*, int **n*, int ***connect*)**

Descrição: esta função informa a conectividade de um elemento cujo o identificador é passado como parâmetro.

Parâmetros:

elem - inteiro identificador do elemento. (Entra)
n - variável que deverá conter o número de nós da elemento finito em questão. (Entra)
connect - vetor contendo a conectividade do elemento dado. (Sai)

- **MisGetNumElems (int **nelems*)**

Descrição: esta função retorna através da variável *nelems* o número de elementos existentes no modelo numérico.

Parâmetros:

nnodes - variável através da qual é passado o número de elementos do modelo numérico. (Sai)

- **MisGetNumNodes (int **nnodes*)**

Descrição: esta função retorna através da variável *nnodes* o número de nós existentes no modelo numérico.

Parâmetros:

nnodes - variável através da qual é passado o número de nós do modelo numérico. (Sai)

- **MisGetNodeTopology (int *node*, int **type*, int **nodetop*)**

Descrição: dado o identificador do elemento, esta função responde qual o tipo da entidade geométrica associada ao elemento e o identificador desta entidade.

Parâmetros:

elem - inteiro identificador do elemento. **(Entra)**

type - variável que deverá conter o tipo da entidade geométrica associada ao elemento em questão. **(Sai)**

elemedgetop - variável que deve conter o identificador da entidade geométrica associada ao elemento. **(Sai)**

- **MisGetElemTopology (int **elem*, int **type*, int *elemtop*)**

Descrição: dado o identificador do elemento, esta função responde qual o tipo da entidade geométrica associada ao elemento e o identificador desta entidade.

Parâmetros:

elem - inteiro identificador do elemento. **(Entra)**

type - variável que deverá conter o tipo da entidade geométrica associada ao elemento em questão. **(Sai)**

elemedgetop - variável que deve conter o identificador da entidade geométrica associada ao elemento. **(Sai)**

- **MisGetElemEdgeTopology (int *node0*, int *node1*, int **type*, int **elemedgetop*)**

Descrição: Esta função responde qual a entidade geométrica e seu tipo associados a uma aresta cujo os identificadores dos extremos são passados.

Parâmetros:

node0 - inteiro identificador de um dos extremos da aresta. **(Entra)**

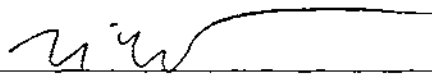
node1 - inteiro identificador do outro extremo da aresta. **(Entra)**

type - variável que deverá conter o tipo da entidade geométrica associada a aresta em questão. **(Sai)**

elemedgetop - variável que deve conter o identificador da entidade geométrica associada a aresta em questão. **(Sai)**

Um Sistema de Modelagem Bidimensional Configurável para Simulação Adaptativa em Mecânica Computacional

Dissertação de mestrado apresentada por Eduardo Setton Sampaio da Silveira em 30 de agosto de 1995 ao Departamento de Engenharia Civil da PUC/RJ e aprovada pela Banca Examinadora, formada pelos seguintes professores:



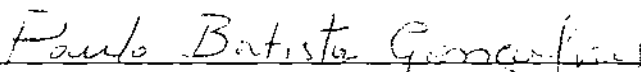
Luiz Fernando Ramos Martha (Orientador)
Departamento de Engenharia Civil - PUC-RJ



Marcelo Gattass
Departamento de Informática - PUC-RJ



Bruno Feijó
Departamento de Informática - PUC-RJ




Paulo Batista Gonçalves
Departamento de Engenharia Civil - PUC-RJ



Túlio Nogueira Bittencourt
EPUSP

Visto e permitida a impressão.
Rio de Janeiro, 07 de maio de 1996.



Coordenador de Programas de Pós-Graduação do
Centro Técnico e Científico