Uma arquitetura para configuração de modeladores geométricos: aplicação a mecânica computacional

Marcelo Tilio M. de Carvalho, Luiz Fernando Martha[*],

Paul A. Wawrzynek[†]

RRsRRRRR

This paper proposes an extensible computational mechanics environment. The software architecture of this environment is independent of the type of mechanics problem under consideration. The application can be extended easily through the configuration of attributes, which constitute the semantics of the application. An embeded interpreted configuration language is adopted for this purpose. Attribute specification is performed independently of the geometric model description and its discretization. The system provides a high level interface to link the simulation attributes to the geometric and discretization entities. The model information and associated attributes are organized in object oriented classes. The classes are defined by way of the configuration language. New classes and methods can be created at run time, allowing the application to be reconfigured without recompilation. While its use for computational mechanics is described here, with the appropriate definition of classes and methods this architecture can be used for a large number of different applications that involve geometrical modeling.

Resumo

Este artigo propõe um ambiente extensível para aplicações de mecânica computacional. A arquitetura do *software* deste ambiente é independente do tipo de problema de mecânica considerado em específico. A aplicação pode ser estendida facilmente através da configuração de atributos, que

[*]ICAD – Laboratório de CAD Inteligente e TeCGraf–Grupo de Tecnologia em Computação Gráfica, ITS, PUC-Rio Prédio do ITS, Rua Marquês de São Vicente 225, 22453-900 Rio de Janeiro, RJ, Brasil {tilio,lfm}@icad.puc-rio.br

[†]C.F.G. – Cornell Fracture Group, Cornell University Engineering and Theory Center Building, room 435 Ithaca, N.Y., U.S.A, 14853. wash@stout.cfg.cornell.edu

constituem a semântica da aplicação. Uma linguagem de configuração interpretada e acoplada é adotada para esse fim. A especificação de atributos é feita independentemente da descrição geométrica do modelo e de sua discretização. O sistema oferece uma interface de alto nível para conectar os atributos de simulação com as entidades geométricas e de discretização. A informação sobre o modelo e os atributos são organizados em classes dentro do paradigma de programação orientada a objetos. As classes são definidas através da linguagem de configuração. Novas classes e métodos podem ser criados em tempo de execução, possibilitando que a aplicação seja configurada sem recompilação. Embora o sistema seja aqui descrito para a área de mecânica computacional, com uma definição apropriada de classes e métodos essa arquitetura pode ser usada em um grande número de aplicações que envolvem modelagem geométrica.

1 Introdução

Modeladores geométricos são ferramentas que permitem criar representações computacionais de objetos (modelo geométrico). Um modelo geométrico é formado por componentes com geometria bem definida (por exemplo através de suas coordenadas) e por relações de conectividade entre esses componentes (topologia). Dentre as aplicações que podem utilizar os serviços de um modelador geométrico, estão as simulações numéricas de problemas de engenharia. Exemplos de simulações incluem, entre outros, distribuição de tensões em meios elásticos, difusão térmica em um domínio arbitrário ou propagação de ondas sísmicas na crosta terrestre. Esses tipos de problemas são referidos neste trabalho como simulações de mecânica computacional.

Em simulações de problemas complexos são utilizados, usualmente, métodos numéricos para a análise matemática, principalmente o método dos elementos finitos [Bathe 1982]. Esses métodos requerem um modelo numérico, que consiste basicamente de informações específicas da aplicação associadas a uma discretização do objeto (descrição do objeto em termos de elementos de formas simples, como quadriláteros ou tetraedros). Essas informações específicas da aplicação são chamadas *atributos* e representam as informações adicionais à descrição geométrica necessárias para a definição completa de um problema.

Muitos sistemas existentes para simulações de mecânica computacional são específicos para uma aplicação em particular, sendo portanto difícil a sua utilização em novas aplicações, ou são muito grandes e genéricos, dificultando o uso e a manutenção. Esses sistemas são, geralmente, bastante complexos e utilizam estruturas de dados sofisticadas. Desse modo, é difícil adicionar novas funcionalidades para atender diferentes tipos de simulações, pois qualquer modificação requer, normalmente, um conhecimento detalhado do código da implementação. Por outro lado, existem vários aspectos da simulação, como a modelagem geométrica, a discretização e a visualização dos resultados, que são independentes do tipo de análise e poderiam ser utilizados em diversas aplicações.

É desejável, portanto, que exista um mecanismo que possibilite a especificação dos atributos de maneira independente dos serviços de modelagem da aplicação, aproveitando-se, assim, os módulos responsáveis por esses serviços em vários tipos de simulação.

Nesse sentido, este trabalho propõe um ambiente para simulação de mecânica computacional, no qual é possível a configuração de atributos específicos a um determinado problema, com um alto grau de abstração, sem a necessidade de recompilação e re-ligação do programa e, principalmente, sem precisar conhecer detalhes sobre o código. Como os sistemas de análise são tipicamente muito grandes, a recompilação de módulos e posterior re-ligação não é uma alternativa prática. Para isso, o mecanismo para a configuração dos atributos utiliza uma linguagem de extensão interpretada.

Até recentemente, a especificação dos atributos para a geração do modelo numérico, e em muitos casos a própria descrição da geometria do problema, era feita diretamente em relação às entidades da discretização deste modelo (nós e elementos). Este procedimento não é compatível com a intenção de automação do processo e também não se mostra adequado para problemas complexos, onde, por exemplo, a geometria se modifica durante a análise. Nesses casos, a definição do problema (descrição geométrica e atributos) deve ser feita independentemente da discretização.

Com essa abordagem, referida como modelagem baseada em geometria, o processo de criação do modelo numérico consiste em construir a geometria do modelo do objeto a ser analisado, associar os atributos a partes dessa geometria, discretizar o modelo e mapear os atributos das entidades geométricas para as entidades da discretização (nós e elementos). Este procedimento oferece diversas vantagens, entre as quais um melhor suporte para a automação do processo de modelagem e a simplificação da geração do modelo para a simulação, uma vez que a discretização herda, automaticamente, os atributos do modelo geométrico [Shephard-Finnigan 1988; Finnigan-Kela-Davis 1989].

Com a arquitetura proposta, a especificação dos atributos é feita de forma independente da descrição geométrica do modelo e de sua discretização. Utiliza-se uma estruturação de classes, dentro do paradigma de programação orientada a objetos e um conjunto de funções responsáveis pela interface entre o ambiente de configuração dos atributos e o modelador geométrico. Uma das metas importantes da arquitetura apresentada é a de poder ser implementada não só em novos programas, mas também em códigos já existentes.

Este trabalho faz inicialmente uma discussão, em termos gerais, de re-uso e configuração de aplicações. Em seguida os conceitos de configuração de atributos que caracterizam uma determinada aplicação e de re-uso de *software* são combinados para definir e criar o ambiente proposto. A seguir descreve-se a arquitetura de tal ambiente. Por fim apresenta-se uma breve descrição das classes implementadas e uma ilustração de configuração de um novo tipo de atributo.

2 Re-uso de software e configuração de aplicações

Devido ao alto custo de desenvolvimento e ao crescente aumento de complexidade das aplicações em computador, tem sido grande a preocupação com o re-uso de programas ou de partes de programas.

Em abrangente discussão a respeito de re-uso de programas, Krueger (1991) ressalta que a tarefa de re-usar código está intimamente ligada ao conceito de abstração. A eficácia na reutilização de técnicas pode ser medida, informalmente, como o esforço intelectual que deve ser gasto para reutilizar, parcial ou integralmente, um trabalho anteriormente desenvolvido.

Uma grande dificuldade encontrada para se re-usar código é o fato de, muitas vezes, módulos conterem informações que não são intrínsecas às suas funcionalidades, como, por exemplo, uma estrutura de dados ter informações sobre a interface da aplicação ou sobre como é feita a sua visualização, pois há diversas maneiras de se visualizar uma determinada estrutura de dados. Um módulo deve especificar seus requisitos, mas não a maneira como esses requisitos devem ser satisfeitos. Para facilitar o re-uso de código, deve existir uma interface que separe os serviços oferecidos pelos diversos módulos que compõem o programa. Um aspecto bastante vantajoso dessa abordagem é que uma interface bem especificada entre módulos define uma clara separação de tarefas [Krasner-Pope 1988, Cowan *et al* 1993].

Ultimamente, tem sido crescente a demanda por programas configuráveis. Uma arquitetura composta por um único módulo, formado por diversos subcomponentes, não se mostra mais adequada às necessidades de diversas áreas. É comum encontrarmos programas que, apesar de implementarem tecnologias bastantes sofisticadas, deixam de poder ser aplicados em áreas afins por simples inadequação de sua interface. Em outros casos, requerem que o reaproveitamento seja feito com acesso direto à implementação da tecnologia da aplicação, o que torna, em geral, a tarefa bastante árdua.

Aplicações complexas são melhores estruturadas se compostas de dois ou mais componentes: um componente principal, que implementa os serviços oferecidos pela aplicação, e componentes auxiliares que oferecem acesso programável aos serviços existentes [Valdés 1991]. Existem diversos exemplos desta arquitetura em diferentes áreas, tais como banco de dados (Paradox), planilhas (Excel) e CAD (Autocad).

A configuração de aplicações pode ser vista também sob o enfoque de re-uso, já que reaproveitamos a implementação de uma mesma tecnologia para diversos fins mais específicos.

3 Arquitetura para aplicações configuráveis em mecânica computacional

Este trabalho não se destina a tratar de detalhes referentes aos serviços de modelagem, mas sim dos aspectos referentes à especificação dos atributos de simulação. Assume-se que os serviços de modelagem são fornecidos por algum modelador. Os atributos, no entanto, devem ser associados às entidades geométricas definidas pelo modelador. Além disso, a especificação de um atributo pode requisitar informações referentes ao modelador como, por exemplo, o valor da normal de uma superfície em um determinado ponto ou que entidades foram selecionadas pelo usuário em um determinado momento (para aplicar ou consultar um atributo, por exemplo).

É necessário, portanto, uma comunicação entre o mecanismo de configuração dos atributos e os serviços de modelagem. De modo a permitir que a especificação dos atributos seja feita de forma independente do modelador geométrico, essa comunicação 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 deve ser independente da especificação dos atributos.

Nesse sentido, a arquitetura proposta divide a aplicação em três partes. Uma parte representa a tecnologia da aplicação (serviços de modelagem). 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 aos serviços de modelagem, feito através de uma linguagem acoplada, estendida com comandos específicos para esse fim.

Uma linguagem acoplada é uma linguagem que depende de um programa hospedeiro. Essa linguagem será referida ao longo deste trabalho como linguagem de configuração. A linguagem utilizada para implementação dos serviços da aplicação (programa hospedeiro) será referida como linguagem de programação. Uma aplicação construída com a arquitetura proposta consiste de:

- Um módulo para os serviços de modelagem. Conforme mencionado anteriormente, deve-se entender por modelagem, a capacidade de descrição da geometria, geração de malha e visualização dos resultados.

- Uma linguagem para configuração, como Tcl [Ousterhout 1994] ou Lua [Ierusalimschy *et al.* 1994], estendidas com mecanismos para suportar a definição de classes, no contexto de programação orientada a objetos (Lua já possui esse mecanismo) e uma interface gráfica para acessar e manipular essas classes, como Tk [Ousterhout 1994] ou EDG [Celes 1994].

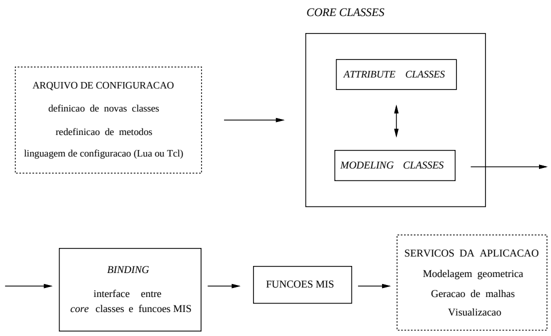

Figura 1: Arquitetura do sistema

- Uma coleção de classes básicas (*core classes*), que definem, com alto grau de abstração, as entidades normalmente envolvidas no processo de geração dos modelos geométrico e numérico (*modeling classes*) e a base a partir da qual classes específicas de atributos são construídas (*attribute classes*).
- Um conjunto de funções de interface (MIS – *Modeling Interface Specification*) que são responsáveis pelo acesso à base de dados do modelador. Qualquer consulta a essa base de dados é feita através dessas funções.
- Um conjunto de funções que são responsáveis pela ligação (*binding*) entre as *core classes* e as funções de interface mencionadas acima.

A Fig. 1 ilustra uma aplicação que utiliza a arquitetura proposta. Os retângulos com linha tracejada correspondem aos módulos que não fazem parte do sistema proposto, ou seja, o módulo referente aos serviços da aplicação (modelador) e o arquivo de configuração, onde é feita a configuração dos atributos (especificação de novos tipos de atributos para uma determinada simulação). Os retângulos com linhas cheias representam o sistema responsável pela comunicação entre o ambiente de configuração e o modelador, denominado ESAM (*Extensible System of Attribute Management*), composto dos módulos que permitem o acesso programável ao modelador, feito em diversos níveis de abstração.

4 Sistema para o gerenciamento dos atributos

Além dos módulos que compõem o sistema ESAM, descritos acima, existem um conjunto de rotinas, implementadas em C, que representam os serviços oferecidos pelo sistema. Essas funções incorporam o ambiente de configuração de atributos à aplicação e têm como objetivo permitir ao usuário final acessar às funcionalidades do ambiente de configuração. Esses serviços incluem funções para criar um atributo novo, aplicar um atributo a uma entidades geométrica, modificar um atributo, especificar um atributo como o atributo corrente, consultar um atributo

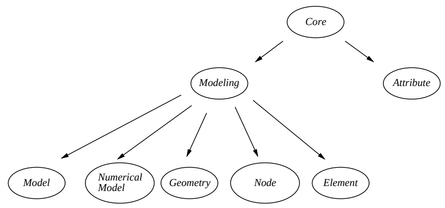

Figura 2: Organização das *core classes*

de uma entidade, carregar o arquivo de configuração desejado, salvar o modelo numérico em um formato específico, entre outros. A aplicação programa sua interface para que os usuários tenham acesso a esses serviços (através de *callbacks* de eventos, por exemplo). O sistema foi implementado com a linguagem Lua e o *toolkit* EDG.

4.1 Core classes

As *core classes* formam uma biblioteca de classes que representam as grandezas comuns envolvidas em uma simulação de mecânica computacional baseada em geometria. Essas classes representam a abstração dos atributos, dos modelos geométrico e numérico e das entidades referentes a esses modelos. A hierarquia dessas classes 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*) (Fig. 2). 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.

4.2 Classe Attribute

A classe *Attribute* representa a superclasse dos atributos necessários à simulação. Subclasses derivadas dessa classe devem ser criadas para atender aos requisitos específicos de cada simulação, como por exemplo, condições de contorno e materiais. As instâncias dessas 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 da classe *Geometry*). Esses objetos são referenciados como objetos *atributo*.

Os objetos atributo são referenciados por 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.

Qualquer método específico a um determinado atributo pode ser criado pelo usuário configurador. Existem métodos, no entanto, que devem ser obrigatoriamente escritos para cada novo tipo criado, como um método para a captura dos dados que definem o atributo (especifica o diálogo para a definição do atributo), um método para validação dos dados especificados e métodos para transportar os valores do ambiente de configuração para a interface gráfica e vice-versa. Esses métodos serão ilustrados posteriormente.

4.3 Modeling classes

A classe *Modeling* é a superclasse das classes que representam as entidades envolvidas no processo de geração dos modelos geométrico e numérico de uma simulação. Desse modo, a hierarquia dessa classe inclui as subclasses: *Model, Numerical Model, Geometry, Element* e *Node.*

A classe *Model e Numerical Model* implementam a abstração dos próprios modelos geométrico e numérico, e contêm as informações que devem existir nesses modelos. A classe *Model* contém uma coleção dos objetos que representam as entidades geométricas do modelo, juntamente com uma coleção dos objetos que representam os atributos especificados (objetos das subclasses de *Geometry e Attribute*). Cada instância da classe *Model* representa um modelo geométrico específico.

A classe *Numerical Model* contém uma coleção de objetos que representam as entidades da discretização do modelo (objetos das subclasses de *Node e Element*). Essa classe representa uma abstração, no contexto do ambiente de configuração, do modelo numérico utilizado na simulação. Uma instância dessa classe é criada para cada análise a ser efetuada e contém as informações necessárias para complementar a especificação do modelo numérico, ou seja, contém referências às entidades da discretização e um meio de acessar os atributos associados a essas entidades.

4.4 Classe Geometry

Esta classe representa a superclasse das classes que implementam a abstração das entidades geométricas existentes no modelador. Os objetos dessa classe, referidos como objetos *geométricos*, contêm referências para as entidades geométricas do modelador e referências para os objetos atributo associados a essas entidades.

Um objeto geométrico 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 (*id*), que pode ser qualquer valor que identifica a entidade no contexto do modelador.

A classe *Geometry* implementa métodos para a manipulação da lista de atributos da entidade geométrica, como, por exemplo, adicionar e retirar um atributo dessa lista, métodos para aplicação de atributos na entidade geométrica ou para

Figura 3: Relação entre objetos das classes *Node e Element, Geometry e Attribute*

consulta dos atributos da entidade geométrica. 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 esse método para que, antes de adicionar o atributo na lista, verifique se já existe algum atributo do mesmo tipo associado ao vértice selecionado.

4.5 Classes Element e Node

Estas classes representam a abstração dos nós e elementos da discretização do modelo. É criado um objeto para cada nó e elemento da malha. Um objeto dessas classes referencia o objeto geométrico correspondente à entidade geométrica à qual o nó ou elemento representado está associado. Por exemplo, se um nó da malha no modelo geométrico está sobre uma aresta, o objeto correspondente a esse nó referencia o objeto geométrico correspondente à aresta.

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 a análise, é necessário mapear os atributos dos objetos geométricos para os objetos da classe *Node*. Esse mapeamento, no entanto, é feito automaticamente pelo sistema, através da referência ao objeto geométrico.

Essas classes implementam 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 capturar as coordenadas de um nó, acessar os atributos aplicados no nó ou elemento, ou obter a conectividade do elemento (lista de nós que compõem o elemento).

A Fig. 3 ilustra a relação dos objetos das classes *Node e Element* com os objetos atributo e geométricos, que permite o mapeamento dos atributos das entidades geométricas para as entidades da discretização.

4.6 Funções de interface com o modelador

Conforme mencionado anteriormente, a especificação de um atributo pode requisitar informações referentes ao modelador. Essas informações são armazenadas na base de dados do modelador, embora possam não estar disponíveis diretamente. O meio como as informações requisitadas são obtidas é, portanto, dependente da implementação da base de dados. Nesse sentido deve existir uma interface entre a base de dados do modelador e o restante da aplicação.

Nesse contexto, o sistema proposto especifica um conjunto de funções, referidas como MIS (*Modeling Interface Specification*), necessárias para a comunicação entre o ambiente de configuração e o modelador. A comunicação é feita somente através da API *(Application Program Interface)*, ou seja, das variáveis dos argumentos dessas funções, que não contêm nenhum tipo de estrutura de dados. Esse 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, é necessário somente reescrever as funções MIS para esse novo modelador, sem nenhuma alteração no restante da aplicação.

4.7 Binding

O módulo *binding* é o responsável pela comunicação entre a parte configurável da aplicação e o modelador. O acesso às funções MIS pela parte configurável da aplicação representa uma extensão da linguagem de configuração para esse domínio. A forma como as funções do programa hospedeiro (escritas em C) são acessadas pela parte configurável depende da linguagem de configuração utilizada, e deve ser feita por um módulo independente tanto do modelador como da parte configurável. Caso contrário, o implementador das funções MIS deveria ter acesso à linguagem de configuração, ou o configurador à linguagem do programa hospedeiro. O módulo *biding* é o responsável por estender a linguagem de configuração. Com isso, na parte configurável da aplicação, acessa-se os dados do modelador através dos métodos das *modeling classes*. No modelador, escreve-se as funções MIS que são posteriormente acessadas pelas *modeling classes* através do módulo *binding*.

Com esta arquitetura, cria-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. Por outro, o configurador acessa os dados do modelador através de métodos das classes definidas no próprio ambiente de configuração. Assim, uma eventual troca de modelador não invalida o trabalho de configuração, assim como uma eventual troca de linguagem de configuração não invalida as implementações das funções de acesso ao modelador.

5 Níveis de abstração

Pretende-se, com a arquitetura proposta, tornar a tarefa de especificação de atributos para simulação de mecânica computacional o mais simples possível. A tecnologia da aplicação se refere a modeladores geométricos e geradores da discretização de elementos finitos.

Nesse contexto, o desenvolvimento de uma aplicação configurável que utiliza a arquitetura proposta envolve a participação de quatro grupos diferentes, com crescente nível de abstração para a realização das tarefas e que diferem entre si de acordo com seus conhecimentos de programação e tarefas que devem desempenhar ao longo do desenvolvimento de uma aplicação:

- Programadores dos serviços da aplicação são os responsáveis pela implementação da tecnologia da aplicação. Possuem um alto nível de conhecimento sobre estrutura de dados. São os responsáveis por escreverem as funções MIS.

- Programadores da configuração são os responsáveis pela implementação das funcionalidades referentes à configuração da aplicação. Devem estender a linguagem de configuração, adicionando comandos que possibilitem o acesso à tecnologia da aplicação (funções do módulo *binding*). Para isso devem conhecer a API da camada de abstração implementada pelos programadores de aplicação (funções MIS). Não têm conhecimento sobre detalhes de implementação dessa tecnologia. Podem adicionar uma nova *core class*, aumentando, deste modo, a capacidade de configuração da aplicação. Esse nível representa o sistema proposto para comunicação entre a parte configurável e os serviços da aplicação.

- Usuários avançados, ou usuários mestres, são especialistas na área relacionada com seu trabalho e são os responsáveis pela configuração da aplicação para suas necessidades específicas, feita através da especificação de subclasses das *core classes*. Devem saber como criar uma nova classe. Para isto devem conhecer e entender as classes existentes. Possuem conhecimento da linguagem de configuração.

- Usuários finais somente utilizam as funcionalidades oferecidas pelo sistema através de uma interface gráfica (botões, listas, etc.), tais como criar um atributo novo ou selecionar um atributo de uma lista existente para aplicar a uma determinada entidade geométrica. Não é requerido nenhum conhecimento sobre a linguagem de programação ou de configuração. Devem saber escolher o arquivo de configuração adequado às suas necessidades.

6 Ilustração

Suponha, por exemplo, que uma simulação necessite de um novo tipo de material, chamado Isotrópico, que contém duas constantes (e e nu) como parâmetros. O configurador da aplicação deve criar essa classe no arquivo de configuração. A partir desse instante, o serviço para criação de um novo atributo coloca esse novo de material disponível para ser utilizado pela aplicação (lista de materiais mostrada na Fig. 4). Uma vez selecionada pelo usuário, a opção Isotrópico

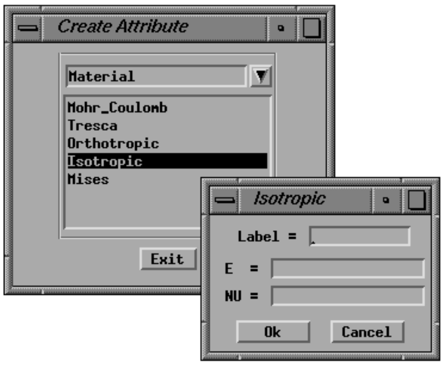

Figura 4: Diálogo para captura dos atributos do material

dispara o método *CrtDlg* escrito pelo configurador, que mostra o diálogo menor da Fig. 4. A seguir ilustra-se um trecho do arquivo de configuração para a criação desse novo material. O método corresponde ao diálogo menor da Fig. 4.

```
MATISO  = crtclass { name = "Isotropic", parent = MATERIAL,
                     vars = { "e", "nu" } }

function MATISO:CrtDlg()

  E  =  field { prompt=" E  = " }  \\
  NU =  field { prompt=" NU = " }  \\
end
```

7 Conclusões

A arquitetura proposta define uma clara independência entre os serviços da aplicação e o ambiente de configuração. A especificação de funções para a interface com os serviços da aplicação (MIS) possibilita a utilização do ambiente de configuração com qualquer modelador geométrico, sem que haja nenhuma alteração no restante do sistema. Para isso, é necessário somente escrever essas funções para o novo modelador. Isso representa uma grande vantagem em um ambiente onde existem diversos modeladores similares. A arquitetura proposta permite, devido a essas funções de interface, que aplicações existentes tornem-se configuráveis, mesmo não tendo sido concebidas originalmente para isso.

A arquitetura apresentada, devido ao fato dos objetos atributo serem especificados independentemente dos objetos geométricos, permite que o ambiente de configuração proposto seja utilizado em outras aplicações além de simulação em mecânica computacional. Para isso, é necessário somente a criação de subclasses da classe *Attribute*, referentes ao domínio de interesse da aplicação. Desse modo, a estratégia proposta pode ser vista como uma estratégia para configuração de modeladores geométricos em geral. Nesse caso, a aplicação do sistema proposto à mecânica computacional pode ser vista como um caso particular.

Esta proposta representa um significativo avanço em simulações de mecânica computacional pois proporciona um ambiente onde é possível o re-uso da tecnologia da aplicação e novas implementações podem ser feitas com extrema rapidez e relativa simplicidade.

7.1 Agradecimentos

Os autores gostariam de agradecer a Luiz Henrique de Figueiredo e Waldemar Celes Filho pelas valiosas discussões e sugestões ao longo deste trabalho. Este trabalho teve suporte parcial do projeto temático do CNPq GEOTEC – Geoprocessamento: Sistemas e Técnicas.

8 Referências Bibliográficas

K. J. Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice Hall, 1982.

J. Bentley, "Programming Pearls: Little Languages", *Communications of the ACM* (Aug 1986)

W. Celes Filho, "Sistema EDG – Manual de Programação", TeCGraf – Grupo de Tecnologia em Computação Gráfica, PUC-Rio, PETROBRÁS/CENPES/SEPROC, 1994.

D. D. Cowan, R. Ierusalimschy, C. J. P. Lucena, T. M. Stepien, 'Àbstract Data Views", *Structered Programming*, **14** (1) (Jan 1993) 1–13.

P. M. Finnigan, A. Kela, J. E. Davis, "Geometry as a Basis for Finite Element Automation", *Engineering with Computers* **5** (1989) 147–160.

R. Ierusalimschy, L. H. de Figueiredo, W. Celes Filho, "Reference manual of the programming language ßLua", *Monografias em Ciência da Computação* **4/94**, Departamento de Informática, PUC-Rio, 1994.

G. E. Krasner, S. T. Pope, 'À Cookbook for Using The Model-View-Controller User Interface Paradigm in Smalltalk-80", *JOOP* (Aug 1988) 26–49.

C. W. Krueger, "Software Reuse", *ACM Computing Surveys* **24** (2) (Jun 1992)

J. K. Ousterhout, *Tcl and the Tk ToolKit*, Addison-Wesley, 1994.

M. S. Shephard, P. Finnigan, "Integration of Geometric Modeling and Advanced Finite Element Preprocessing", *Finite Elements in Analysis and Design* **4** (1988) 147–162.

M. S. Shephard, "The Specification of Physical Attribute Information for Engineering Analysis", *Engineering with Computers*, **4** (1988) 145–155.

R. Valdés, "Little languages, big questions", *Dr. Dobb's Journal* **16** (9) (Sep 1991) 16–25.