Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXIX, págs. 6029-6043 (artículo completo)
Eduardo Dvorkin, Marcela Goldschmit, Mario Storti (Eds.)
Buenos Aires, Argentina, 15-18 Noviembre 2010

A PLUGIN STRATEGY FOR EXTENSIBLE CONFIGURATION OF ATTRIBUTES IN GEOMETRIC MODELLING FOR FINITE ELEMENT SIMULATIONS

Ricardo Morrot[a], William Wagner Matos Lira[b] and Luiz Fernando Martha[a]

[a] *Tecgraf/PUC-Rio – Computer Graphics Technology Group, Pontifical Catholic University of Rio de Janeiro, http://www.tecgraf-puc.rio.br/*

[b] *Laboratory of Scientific Computing and Visualization (LCCV), Technology Center, Federal University of Alagoas*
http://www.lccv.ufal.br

Keywords: Finite elements, Geometric Modeling, Attributes of Simulation, Plugin.

Abstract. This work proposes a methodology, using plugins resources, for configuration of finite element simulation attributes in geometric modeling. The attributes are necessary additional information for the complete definition of a finite element model, as they constitute specific characterizations of the simulations. The configuration of these attributes allows a relatively simple extension of a given geometric modeler, so that it may be used in other types of finite element simulations. With the plugin methodology, it is possible incorporate to the modeler specific resources that were not initially considered. The strategy adopted in this work consists of using the plugin resource to link, through an interface, a geometric modeler to a library of classes of the module that manages simulation attributes. A simple example is presented to illustrate the proposed methodology.

1 INTRODUÇÃO

No contexto da mecânica computacional, um dos métodos numéricos mais utilizados tem sido o Método dos Elementos Finitos (MEF). Isto se deve à sua grande versatilidade, qualidade de resultados e, principalmente, sua relativa facilidade de implementação. O MEF se baseia em um modelo numérico obtido a partir da discretização do domínio do problema, juntamente com informações adicionais associadas a essa discretização, necessárias para uma completa definição do problema físico. Tais informações adicionais são denominadas atributos. A discretização, denominada malha de elementos finitos, consiste em um conjunto de nós ou vértices (pontos com coordenadas) e um conjunto de elementos finitos com uma topologia pré-definida (triângulos ou quadriláteros, por exemplo). Os elementos são definidos por uma lista de conectividade de seus vértices (sequência de vértices que pertencem a cada elemento).

Geralmente, a metodologia adotada na discretização do domínio é comum para os diversos tipos de problemas que podem ser analisados pelo MEF. Os atributos, entretanto, são específicos para cada tipo de simulação. Por exemplo, em uma análise de tensões de um problema estrutural, alguns atributos se referem às cargas distribuídas, às condições de apoio e aos deslocamentos prescritos. Por outro lado, em uma análise térmica os atributos que particularizam o problema são, entre outros, o fluxo de calor, o gradiente térmico e a temperatura prescrita no contorno do modelo.

Na maioria dos sistemas computacionais utilizados para simulações numéricas que utilizam o MEF, as informações referentes aos atributos estão intrinsecamente ligadas aos processos de modelagem geométrica, geração de malha, análise numérica e visualização de resultados, através da estrutura de dados implementada nos seus códigos. Nesses sistemas, a incorporação de novos tipos de atributos não é uma tarefa trivial, pois, geralmente, os códigos computacionais existentes são grandes e complexos.

É desejável, então, que os atributos possam ser configurados pelo usuário da aplicação, de acordo com o tipo de análise desejada, sem que este usuário precise acessar as informações do código computacional que implementa as outras etapas do processo de simulação. Isso permite que um sistema possa ser facilmente adaptado para diferentes tipos de simulações.

Nessa direção, Carvalho (Carvalho, 1995) apresentou uma arquitetura de software que permite a concepção de sistemas nos quais é possível configurar os atributos específicos a um determinado problema com um alto grau de abstração, ou seja, sem a necessidade de conhecer o código computacional da aplicação. Um sistema denominado ESAM (*Extensible System Attributes Management*) foi desenvolvido baseado nessa arquitetura, onde a configuração dos atributos é feita através de arquivos que contém instruções em uma linguagem de programação extensível (linguagens utilizadas para estender ou configurar uma aplicação para fins particulares) bastante simples. Essa linguagem é interpretada (Rossum, 1993;

Ousterhout, 1994; Ierusalimschy et al., 1994), permitindo que a configuração dos atributos seja feita sem a necessidade de recompilação da aplicação.

Para validação da arquitetura proposta por Carvalho, Lira (Lira, 1998; Lira 2002) incorporou o ESAM a dois pré-processadores de elementos finitos já existentes, apresentando sistemas configuráveis para simulações numéricas na mecânica computacional. Adaptações no ESAM foram necessárias para realizar essa validação e a sua utilização nesses sistemas permite a configuração dos atributos para o uso na modelagem geométrica e geração de malhas de elementos finitos de diversos problemas de engenharia, sem a necessidade de compilar o código para atender especificamente cada um desses problemas.

No entanto, o crescente aumento no volume de dados relacionados à modelagem geométrica de problemas complexos de engenharia tornou proibitivo o uso de linguagem interpretada. Isso se deve ao fato, principalmente, da utilização excessiva de memória que essas linguagens demandam, além do alto custo computacional proporcionado pela interpretação do código durante a execução do programa.

Nesse sentido, o presente trabalho dá continuidade aos trabalhos propostos por Carvalho e Lira e propõe um procedimento utilizando recursos de *plugins* para configuração de atributos de simulações por elementos finitos em modeladores geométricos. Este trabalho ainda permite que um mesmo modelador geométrico possa facilmente ser estendido para sua utilização em diferentes simulações por elementos finitos. O principal ganho é que a estratégia de *plugins*, em contraposição à de linguagens interpretadas, evita o uso desnecessário de memória pela aplicação, além de reduzir o tempo computacional necessário para realizar o gerenciamento dos atributos pelo ESAM.

Para alcançar esses resultados, além da utilização de *plugins*, o sistema ESAM é reescrito usando a linguagem compilada C++, permitindo a que a configuração dos atributos seja realizada das seguintes formas: (a) utilizando linguagens interpretadas, o que mantém as mesmas facilidades da versão original do ESAM; (b) utilizando *plugins*, os quais permitem o uso do ESAM para gerenciamento de atributos em modelagem de problemas complexos com baixo uso de memória e tempo computacional reduzido. No entanto, essa nova implementação do ESAM não é foco principal deste trabalho, não sendo detalhado neste artigo. O objetivo aqui é discutir como uma estratégia de *plugin* pode ser utilizada para configurar atributos de simulação em modelagem geométrica.

2 AMBIENTE CONFIGURÁVEL DE ATRIBUTOS

O ambiente de configuração proposto por Carvalho (Carvalho, 1995) considera apenas os aspectos relacionados à definição dos atributos da simulação. Os detalhes referentes à modelagem geométrica, à geração da malha de elementos finitos e à visualização dos resultados de uma análise numérica não são considerados. Assume-se que estes serviços são fornecidos por alguma aplicação. De modo a permitir que a especificação desses atributos seja feita de forma independente da aplicação, a

comunicação entre o gerenciador de atributos e a aplicação deve ser feita através de um mecanismo que mantenha a independência dos módulos envolvidos na tarefa. Ou seja, o módulo responsável pelo gerenciamento dos atributos não deve ter conhecimento sobre as características específicas da aplicação, assim como a aplicação não deve saber da parte referente à especificação dos atributos.

Nesse sentido, a arquitetura proposta por Carvalho divide a aplicação em três partes. Uma parte representa os serviços ou tecnologia da aplicação, composta pelos aspectos que são independentes do tipo de simulação, tais como modelagem geométrica, geração da malha e visualização dos resultados. Outra parte refere-se aos aspectos específicos de cada simulação, ou seja, a especificação dos atributos. A terceira parte refere-se a um módulo responsável pela comunicação entre as duas partes anteriores, que permite o acesso à base de dados da aplicação com um alto grau de abstração.

Essa arquitetura é implementada computacionalmente através do sistema ESAM (*Extensible System of Attributes Management*), o qual consiste de uma coleção de classes (*Core Classes*), no contexto da programação orientada a objetos, e de um conjunto de funções que são responsáveis pela interface entre as Core Classes e a aplicação.

O ESAM possui também um conjunto de funções que oferecem os serviços (funcionalidades do sistema) necessários para a configuração da aplicação. Esses serviços são oferecidos através de um conjunto de rotinas que devem ser incorporadas a aplicação. Os serviços, entretanto, não incorporam a semântica da aplicação. A Figura 1 mostra a arquitetura proposta por Carvalho (1995), 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 e o arquivo de configuração dos atributos. Nesse arquivo é realizada a especificação dos novos tipos de atributos para uma simulação especifica. Linguagens interpretadas são utilizadas para permitir que a configuração seja realizada sem a necessidade de recompilação do código computacional da aplicação.

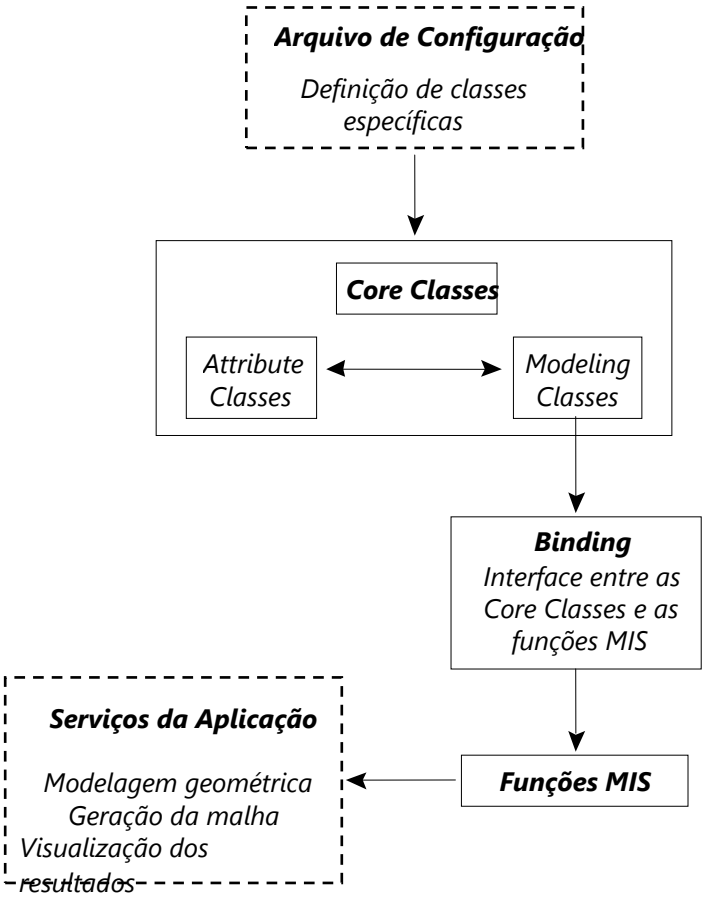

Figura 1: Arquitetura original do sistema ESAM (Lira, 1998).

As *Core Classes* representam uma abstração das entidades envolvidas no processo de simulação, ou seja, representam as entidades geométricas e da malha (*Modeling Classes*) e os atributos de simulação (*Attribute Classes*).

A especificação de atributos pode requisitar informações referentes à aplicação que estão armazenadas na sua base de dados. O meio como as informações são obtidas é, portanto, dependente da implementação desta base de dados. Neste sentido, é desejável que as informações sobre a aplicação sejam obtidas através de chamadas procedurais e não através de acesso explícito à sua estrutura de dados. Ou seja, deve existir uma interface entre a base de dados da aplicação e a parte configurável da aplicação. Essa interface é realizada através de um conjunto de funções, denominadas funções MIS (*Modeling Interface Specification*), que representam uma camada de abstração sobre a base de dados da aplicação.

O módulo *Binding* é responsável pela ligação entre a parte configurável e a aplicação. Esse módulo representa a interface responsável pelo acesso da parte configurável à aplicação. Na aplicação, as funções MIS disponibilizam o acesso aos objetos da simulação aos quais os atributos estão relacionados. O módulo *Binding* permite que o usuário responsável pela implementação da aplicação escreva as funções MIS sem conhecer as características da linguagem de configuração utilizada e do sistema de configuração. Por outro lado, o configurador acessa os dados da

aplicação, sem conhecer o seu código, através de métodos das classes definidas no próprio ambiente de configuração.

Como dito anteriormente, a configuração dos atributos dependente da simulação desejada é, na proposta original (Carvalho, 1995), realizada em um arquivo de configuração escrito em uma linguagem interpretada. Como existe a necessidade de troca de informações entre a aplicação e o ESAM (e consequentemente com o arquivo de configuração), alguns outros módulos desse sistema também devem estar definidos nessa linguagem. Entretanto, o uso da linguagem interpretada torna as ações relacionadas ao gerenciamento de atributos das aplicações mais lento, além de utilizar excessivamente memória para armazenamento de informações. Linguagens interpretadas tendem a utilizar mais memória que linguagens compiladas, pois o seu gerenciamento é realizado de forma automática, sem intervenção do programador.

Como as aplicações que o sistema ESAM foi incorporado focam na solução de problemas complexos, normalmente uma grande quantidade de informações relacionadas ao gerenciamento de atributos é manipulada. Pelos motivos expostos, o uso do ESAM na simulação desses problemas complexos tornou-se proibitivo.

Na solução desses problemas, a idéia é manter as características originais do ESAM e permitir que ele também possa ser utilizado em aplicações que visam a solução de problemas complexos de engenharia através do MEF. Nesse sentido, o código computacional do ESAM é reestruturado e reescrito em uma linguagem compilada e, para manter as mesmas funcionalidades do sistema, conceitos de *plugin* são utilizados, permitindo a configuração de atributos da aplicação de acordo com a simulação desejada. Apesar de necessitar da compilação desses *plugins* para serem incorporados a aplicação, essa compilação é realizada de forma independente da aplicação. Isso permite que a mesma aplicação possa ser utilizada em diferentes simulações, mantendo, assim, uma das principais características do ambiente de configuração usado neste trabalho. A Figura 2 mostra a nova arquitetura do ESAM que suporta o uso de *plugins*.

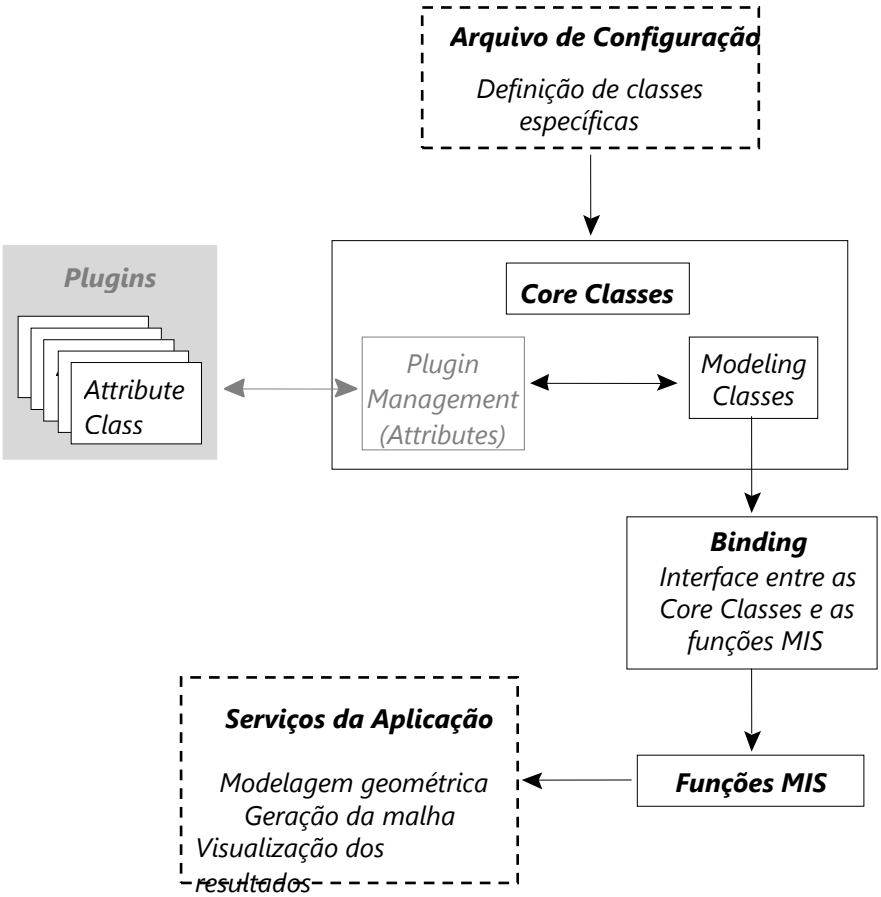

Figura 2: Nova arquitetura proposta para o sistema ESAM.

3 CONCEITOS DE *PLUGIN*

Os *plugins* são "pequenos aplicativos" que têm a finalidade de incorporar determinadas características ou funcionalidades a um "aplicativo maior", ou "aplicativo principal". Neste trabalho, algumas vezes o termo "modelador geométrico" aparece em substituição ao termo "aplicativo principal".

Plugins, normalmente, são encontrados em programas gráficos, browsers, programas matemáticos e diversos outros programas que permitem a integração de recursos específicos e que não foram implementados no aplicativo principal. Um tipo de *plugin* bem explorado por desenvolvedores é o *Add-on* de navegadores de internet, também conhecido como *Addon*, *Add-In* e *Addin*.

Empresas como a Netscape, com o *NPAPI (Netscape Plugin Application Programming Interface*), Adobe (responsável pela popularização do *plugin*), Mozilla, MathWorks e National Instruments disponibilizam aos desenvolvedores de *software* ferramentas (as chamadas *frameworks* ou *SDK's - Software Development Kit*) que permitem a criação de *plugins* para os seus aplicativos.

O *plugin* é conceitualmente uma biblioteca que é vinculada ao aplicativo principal através de uma interface, podendo ser vínculo estático ou dinâmico, as já conhecidas

"DLLs" (*Dynamic-Link Library*). Um *plugin*, por características próprias, não "estende" novas funcionalidades ou modifica funcionalidades já existentes do aplicativo principal. Para fazer isto existem as *Extensions* que permitem a um aplicativo ter novas funcionalidades de forma independente e/ou conjunta. O principal conceito relacionado aos *plugins* é a minimização do esforço envolvido na configuração e administração de um sistema.

3.1 Funcionamento de um *plugin*

Em cada *framework* ou SDK, específicos para cada aplicativo, encontram-se uma série de bibliotecas que permitem a comunicação do *plugin* com o aplicativo principal. Basicamente, toda biblioteca disponibilizada também está vinculada ao aplicativo principal como se pode visualizar na Figura 3.

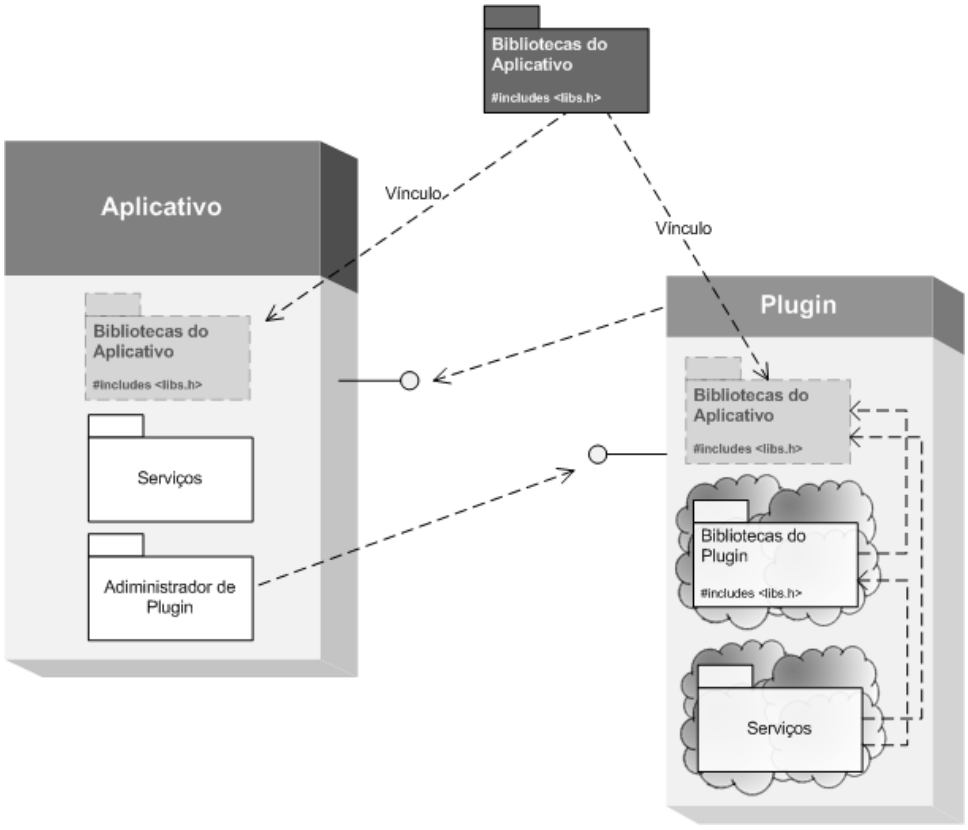

Figura 3: Modelo de *framework* para *plugin*.

O que o aplicativo principal não conhece, e também não precisa conhecer, é a estrutura interna que cada desenvolvedor utilizou na construção de seus *plugins*. Toda essa estrutura "nebulosa" é particular de cada desenvolvedor e não sendo, portanto, necessário o seu conhecimento pelo aplicativo principal. A implantação de *plugins* em um aplicativo exige regras que não só determinam como facilitam a comunicação entre os *plugins* e aplicativo principal. Estas regras são as interfaces.

O aplicativo principal, antes de adicionar um *plugin* tem de reconhecê-lo, uma das exigências das regras. Esse reconhecimento deve ser feito pela própria leitura da

biblioteca (arquivo *plugin*), que deve ter uma marca, mnemônico ou extensão. Em seguida, ocorre um registro para que, ao ser acionado no futuro, ocorra a *troca de dados*. Então, essa *troca de dados* é apenas um recurso de interface que permite ao aplicativo principal reconhecer o *plugin* e consequentemente "oferecer" ferramentas através de métodos e funções. Além disso, nada mais é possível de ser realizado. Um aplicativo que aceita *plugins* deve executar normalmente sem que algum *plugin* tenha sido adicionado.

Outro fator importante na criação de um *plugin* é o controle de versão que determina o tempo de vida útil. Cada *framework* ou SDK liberado pelo fornecedor do *software* possui um número de versão que é automaticamente recuperado no momento do registro. É esse número que permite o registro de um *plugin* no aplicativo, uma forma de manter a compatibilidade da interface.

3.2 Tempo de vida útil de um plugin

Os aplicativos lançados com uma nova versão podem exigir ou não a recompilação dos *plugins* caso tenham ocorrido mudanças importantes em suas bibliotecas. Esse fator depende simplesmente da exigência do fornecedor do *software*. Por isso, os *plugins* quase sempre têm um tempo de vida útil bem reduzido. Na Figura 4 pode-se observar um aplicativo suportando diferentes versões da mesma biblioteca. Esse conceito é muito utilizado, entretanto, quando muitas versões são lançadas sucessivamente. Em geral, a administração de versões se torna um fator complicador, e deve ser evitado a todo custo.

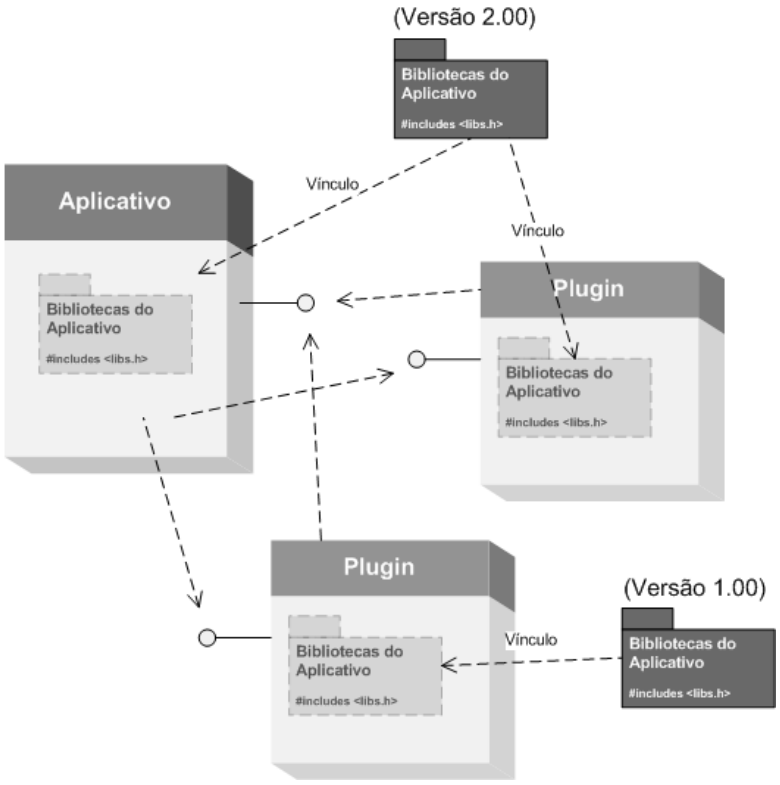

Figura 4: Controle de versão do plugin.

A complexidade envolvida na manutenção, criação de novas versões do aplicativo principal, muitas vezes, gera incompatibilidade de versões e, consequentemente, e leva a modificações do *framework* e SDK disponibilizados para construção dos *plugins*. Entretanto, a mudança mais significativa para uma recompilação de todos os *plugins* se deve exclusivamente à atualização da interface com a inclusão e/ou remoção de funções e métodos.

3.3 Tipos de plugins

As variantes existentes para *plugins* só dizem respeito à tecnologia adotada na modelagem de um aplicativo. Existem várias tecnologias como o ATL/COM, DLL's pura (em C), VBX (16bit), OCX (32bit) e ActiveX (OCX renomeado devido as estratégias da Microsoft na utilização para internet). Entretanto, todas as variantes utilizam a tecnologia de criação de bibliotecas de vínculo dinâmico ou estático.

4 MODELO DE *PLUGIN* PROPOSTO

O modelo de interface que está sendo proposto neste trabalho é uma superclasse abstrata que define uma interface geral para criação de *plugins*, como ilustra a Figura 5. Tudo que a aplicação deve fazer é adicionar os *plugins*, se houverem, e interagir com um composto de funções e métodos membros visíveis, aumentados pelas classes derivadas para atributos (aqui genericamente referidas como classes "Atributos"), porém estas invisíveis para a aplicação.

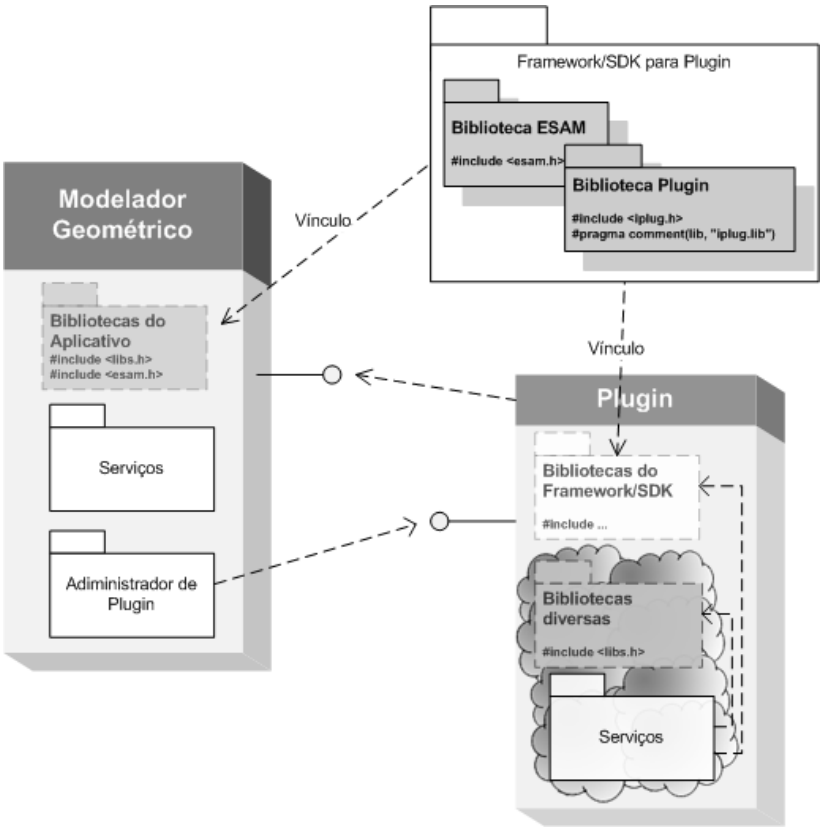

Figura 5: Diagrama ilustrando o vínculo da biblioteca *plugin* no modelador geométrico.

Uma possibilidade de implementação é fazer com que os atributos passem a herdar uma superclasse abstrata IPlug, conhecida também pelo modelador geométrico. Dessa forma, os atributos, a partir de então chamados de plugins, farão parte de uma lista no modelador geométrico (Figura 6).

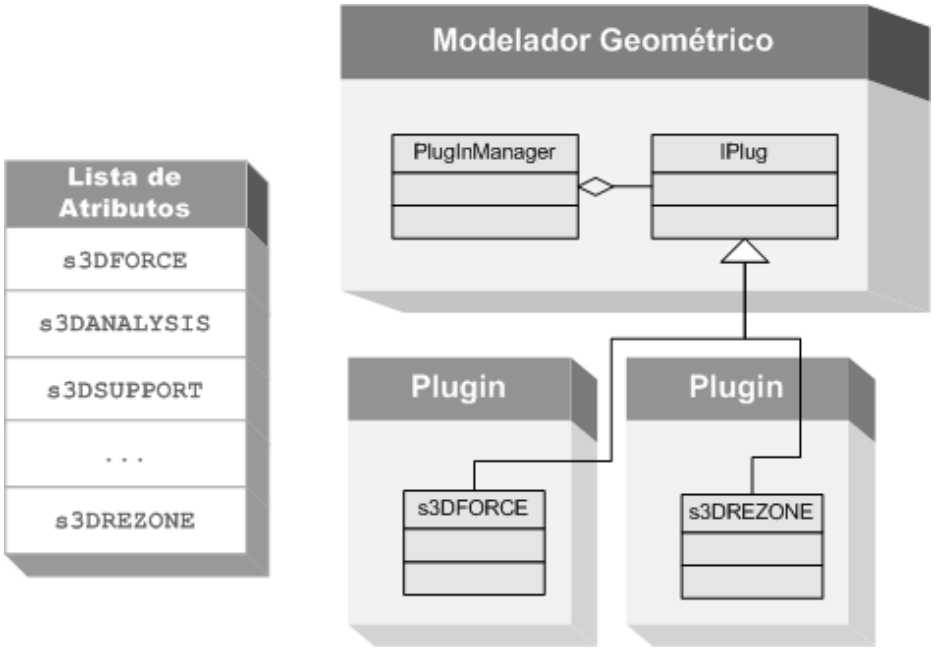

Figura 6: Exemplo de estrutura para *plugin* de única interface.

O inconveniente dessa estrutura é que não representa toda a flexibilidade de um sistema como o ESAM. O *plugin*, assim como o modelador geométrico, só depende em conhecer as classes do ESAM e a superclasse abstrata IPlug que compõe a interface. Devido a restrições de projeto, as classes "Atributos" não possuem um padrão em suas funções e métodos, o que as tornam, de certa forma singulares, e, neste momento, o mais importante numa uniformização é observar com bastante cautela todas as assinaturas e parâmetros, inclusive o retorno das funções onde acontece as principais restrições.

Na Figura 7 é apresentado um modelo simplificado de diagrama de classes com a alternativa adotada, que leva em conta a excessiva quantidade de atributos que não compõem um formato único de classe no modelador geométrico. Através da definição das características singulares de cada atributo pode-se reuni-los em grupos bem específicos de forma a implementar *plugins* com tipos de interfaces diferenciadas e consequentemente acessos diferenciados, como ilustrado na Figura 8, porém, mantendo as características básicas de todos os atributos. Essa forma adotada proporciona mais flexibilidade para um sistema complexo, como o ESAM.

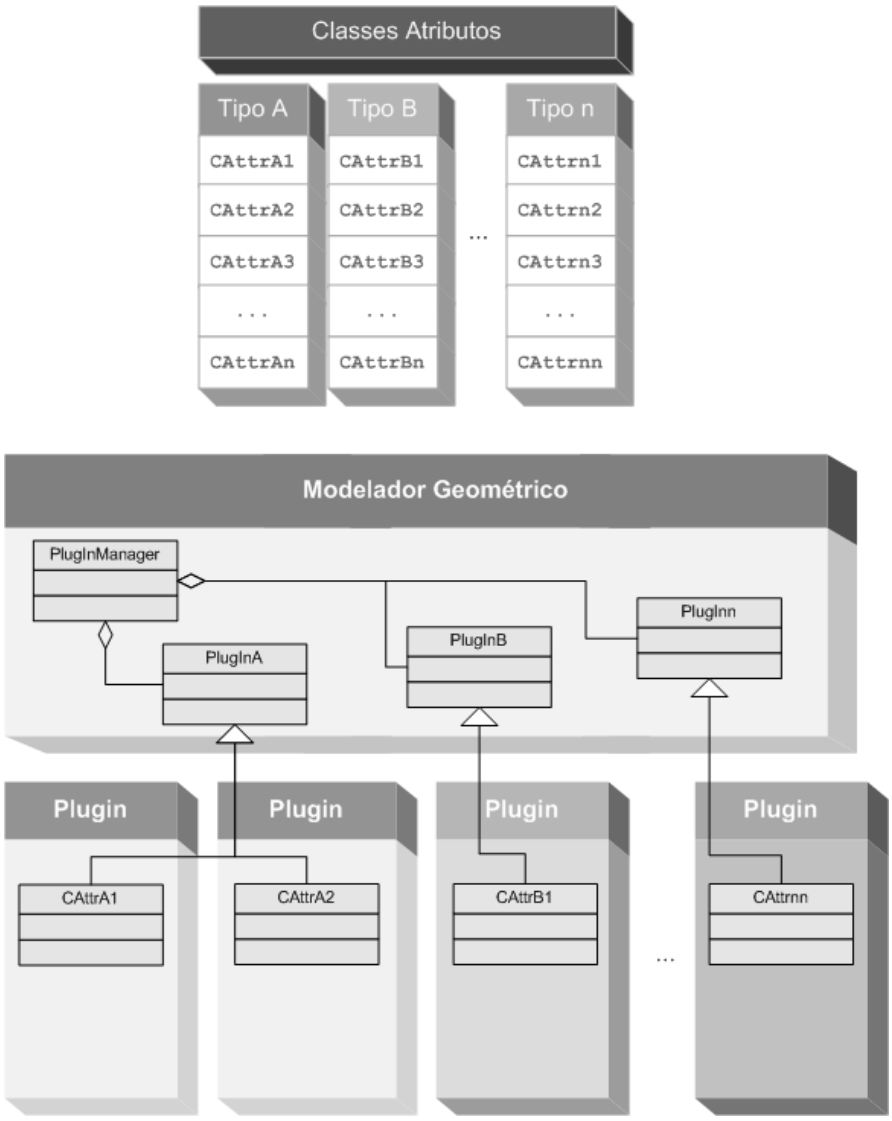

Figura 7: Diagrama do plugin no modelador geométrico.

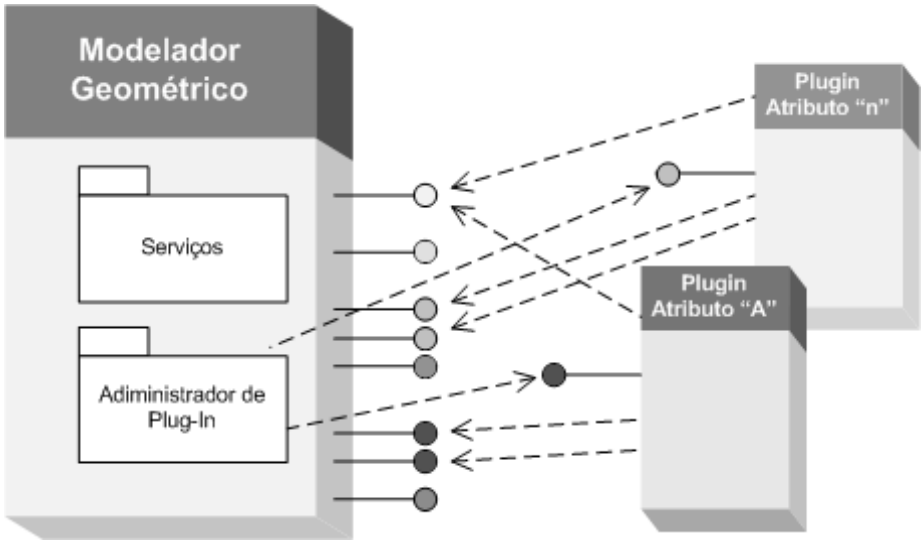

Figura 8: Ilustração dos plugins no modelador geométrico.

5 USO DO ESAM COM RECURSOS DE *PLUGIN* EM UMA APLICAÇÃO

Esta seção ilustra a utilização de plugins em uma aplicação que utiliza o ESAM para o gerenciamento dos atributos da simulação, o qual é feito tomando-se como base o modelador geométrico apresentado por Lira (Lira, 2002). Para iniciar os estudos, uma análise das classes "Atributos" que herdam ESAMAttribute é realizada. Nessa análise, descartam-se todos os membros privados incluindo os atributos, funções e métodos privados, construtores e destrutores públicos ou privados, deixando apenas os membros públicos restantes. Aqui, o importante é identificar o ponto comum a todas essas classes de forma a destacar apenas funções e métodos comuns (Figura 9). Como resultado dessa análise, tem-se uma classe abstrata chamada de IPlug, para o modelo de interface base (Figura 10). Essa superclasse é unida a classe ESAMAtribute tornando-a única. É importante observar que nessa análise não considera-se os parâmetros das mesmas. Entretanto, para o caso de métodos e funções com diferentes tipos de parâmetros utiliza-se o conceito de sobrecarga.

Figura 9: Métodos e funções comuns a todas as classes "Atributos".

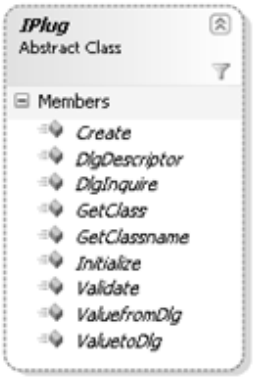

Figura 10: Classe abstrata para os "Atributos".

Após a conclusão dessa análise e integração de classes, tem-se uma superclasse abstrata comum a todos os atributos. Essa superclasse é utilizada pelo modelador geométrico, permitindo-o acessar os métodos e funções de interface dos *plugins*, e suas variantes.

6 CONCLUSÕES

Este trabalho apresentou um procedimento utilizando recursos de *plugins* para configuração de atributos de simulações por elementos finitos em modelagem geométrica. Esse procedimento é incorporado a um ambiente de configuração de atributos que permite que uma aplicação seja configurada para a sua utilização em diversos tipos de simulação através do Método dos Elementos Finitos (MEF). Além disso, o uso da estratégia apresentada permite que a aplicação seja configurada sem a necessidade da sua recompilação.

A idéia principal do trabalho consistiu na utilização do recurso de *plugin* para ligar, através de uma interface, uma aplicação (modelador geométrico) a uma biblioteca de classes usada no gerenciamento de atributos de simulação.

Os principais ganhos obtidos com o desenvolvimento deste trabalho relacionados à configuração de atributos de aplicações para o seu uso na solução de problemas complexos de engenharia através do MEF são: (a) possibilidade de utilização da mesma aplicação em diferentes tipos de simulação sem a necessidade de recompilação do código computacional; (b) redução no uso da memória na utilização dessas aplicações; (c) redução no custo de processamento das informações relacionadas aos atributos nessas aplicações.

O uso de *plugins* em uma implementação de projeto desse nível permite uma flexibilidade, aderente a uma interação com múltiplos recursos e conseqüentemente mais fácil de posterior manutenção de novos *plugins* à medida que eles, invariavelmente, se tornem necessários.

AGRADECIMENTOS

Este trabalho está sendo desenvolvido através de uma colaboração do Tecgraf/PUC-Rio (Grupo de Tecnologia em Computação Gráfica da PUC-Rio) e do LCCV/UFAL (Laboratório de Computação Científica e Visualização da UFAL). Os autores agradecem o suporte de infraestrutura e financeiro fornecidos pelo Tecgraf e pelo LCCV. Em particular, o projeto em que este trabalho é desenvolvido é financiado pela Petrobras.

REFERÊNCIAS

Carvalho, M. T., M., *Uma Estratégia para o Desenvolvimento de Aplicações Configuráveis em Mecânica Computacional*, Tese de Doutorado, Departamento de Engenharia Civil – Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio, Rio de Janeiro, RJ, 1995.

Ierusalimschy, R., Figueiredo, L. H., *Reference Manual of the Programming Language Lua*, Monografias em Ciências da Computação 4/94, Departamento de Informática, PUC-Rio, 1994.

Lira, William Wagner Matos, *Um Sistema Integrado Configurável para Simulações em Mecânica Computacional*. Dissertação de Mestrado, Departamento de Engenharia Civil – Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio, Rio de Janeiro, RJ, 1998.

Lira, William Wagner Matos, *Modelagem Geométrica Para Elementos Finitos Usando Multi-Regiões e Superfícies Paramétricas*. Tese de Doutorado, Departamento de Engenharia Civil – Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio, Rio de Janeiro, RJ, 2002.

Object Technology International, Inc., Eclipse Platform Technical Overview, Disponível em: <http://www.eclipse.org/whitepapers/eclipse-overview.pdf>.

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

P. Oriezy, N. Medvidovic, and R. Taylor. *Architecture-based runtime software evolution*. In ICSE '98, 1998.

Plugins, Mozilla Developer Center.
Disponível em: <https://developer.mozilla.org/en/Plugins>. Acesso em: 08/set/2010.

Rossum, G. V., *An introduction to Python for UNIX/C programmers*, Dutch UNIX users group, 1993.

Szyperski, C., *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Pub Co, 1997.

TEXTPATTERN. *Plugin Development Guidelines*. Team TEXTPATTERN, 2010. Disponível em: <http://textpattern.net/wiki/index.php?title=Plugin_Development_Guidelines>. Acesso em: 08/set/2010.