4
SISTEMA COMPUTACIONAL PARA A PREVISÃO DE REFLUXO DE PROPANTE

A ferramenta computacional elaborada durante o desenvolvimento dessa dissertação de mestrado, denominada *PFP System* (*Proppant Flowback Prediction*), visa auxiliar na seleção do material de sustentação de fratura e no projeto de fraturamento hidráulico. Tal ferramenta é baseada nos modelos empíricos e teóricos existentes na literatura de acordo com as condições do poço e do material de sustentação selecionado.

O *PFP System* foi implementado em linguagem Java e em todas as etapas do projeto orientado a objetos (OOD - *Object Oriented Design*) foi utilizada a Unified Modeling Language, UML, que é uma linguagem gráfica universal para a modelagem de sistemas orientados a objeto (Deitel & Deitel, 2002).

Para o desenvolvimento do sistema PFP foi adotada a arquitetura MVC (*Model-View-Controller* – Modelo-Visão-Controlador). Essa arquitetura divide as responsabilidades do sistema em três subsistemas:

1. o modelo, que contém todos os dados e a lógica do programa, ou seja, analisa e calcula as respostas;
2. a visão, que gerencia a saída gráfica e textual de aplicação visível ao usuário, ou seja, fornece a apresentação visual para o modelo;
3. o controlador, que define o comportamento do sistema, enviando a entrada dos dados para o modelo. Ou seja, o controlador é o responsável pela modelagem do poço através da interface gráfica com o usuário (GUI – *Graphical User Interface*).

Portanto, a arquitetura MVC funciona, resumidamente, da seguinte forma de acordo com a Figura 4.1: através do controlador o usuário define os dados no modelo; o modelo, então, informa à visão sobre a alteração dos dados; a visão modifica a sua apresentação visual para refletir as alterações no modelo. Essa arquitetura tem como vantagens a gerência de múltiplos componentes de visão usando o mesmo modelo, a facilidade de incluir novos clientes apenas com a

entrada de novos componentes de visão e controlador e, ainda, a independência entre os componentes.

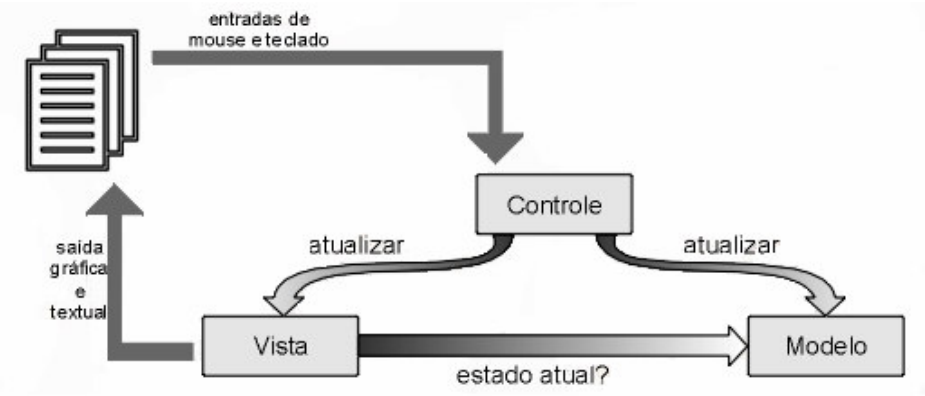

Figura 4.1 – Esquema de funcionamento da arquitetura MVC (http://hercules.nce.ufrj.br/arq-mvc.html).

De acordo com a UML, vários tipos de diagramas podem ser usados para modelar um sistema. Para o sistema PFP foram utilizados, basicamente, três tipos de diagramas:

1. Diagrama de casos de uso;
2. Diagrama de componentes;
3. Diagrama de classes.

Cada diagrama modela uma característica distinta da estrutura ou comportamento do sistema. O primeiro tipo de diagrama se refere ao comportamento do sistema, enquanto que, os dois últimos dizem respeito à estrutura do sistema. Assim, os diagramas de casos de uso representam a interação entre o usuário e o sistema, ou seja, todas as ações que o usuário pode executar no sistema. Os diagramas de componentes modelam os recursos e pacotes (que são grupos de classes) que constituem o sistema. E, por último, os diagramas de classes modelam as classes usadas para construir o sistema.

Na presente dissertação, foram elaborados os diagramas de casos de uso, de componentes e de classes para o sistema *PFP*, além dos diagramas de componentes para os pacotes *model* e *event* e do diagrama de classes para o pacote *model*.

4.1.
DIAGRAMA DO SISTEMA PFP

4.1.1.
Diagrama de casos de uso

Cada caso de uso representa um recurso diferente que o sistema oferece aos clientes. A UML oferece o diagrama de casos de uso para facilitar a reunião dos requisitos, modelando as interações entre os clientes e os casos de uso do sistema.

A Figura 4.2 apresenta o diagrama de casos de uso para o sistema PFP sob a perspectiva do usuário, onde cada caso de uso é representado por uma elipse.

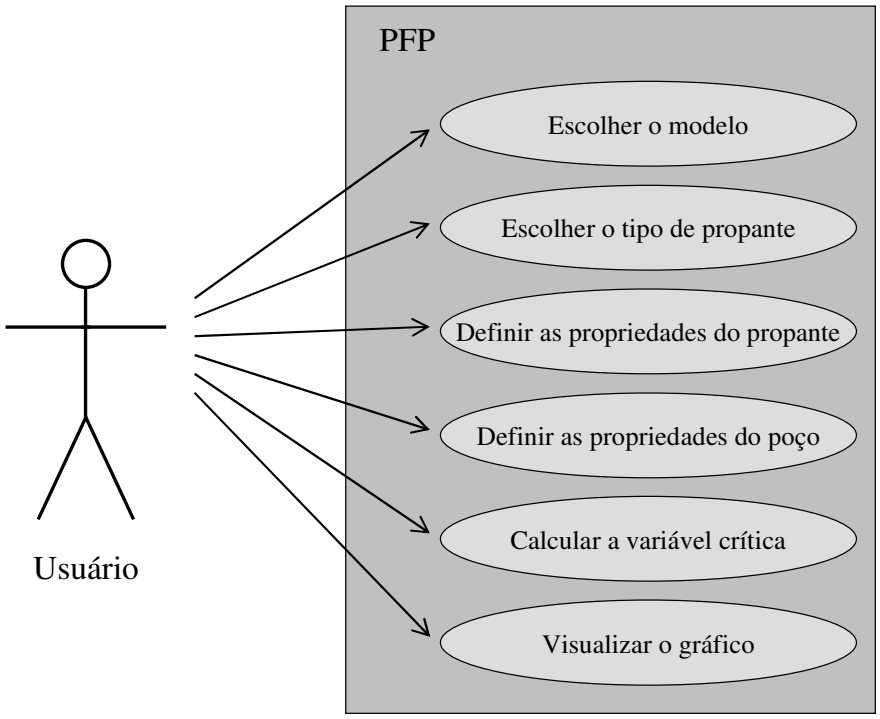

Figura 4.2 – Diagrama de caso de uso para o sistema PFP sob a perspectiva do usuário.

Os casos de uso foram implementados através de uma interface gráfica com o usuário (GUI). Implementou-se essa GUI no pacote *controller*, que possui como classe principal uma subclasse *Object*, denominada *PFPController*. A classe *PFPController* declara uma referência para o *PFPModel* para permitir a interação com o modelo. Além disso, possui os objetos *ControllerMenu*, *PanelNorth*, *PanelSouth*, *PanelWest*, *PanelEast*, *PanelCenter*, *DisplayField*,

ToolBar, *ToolBarStandard*, *ControllerMenu* e *ControllerMenuFile* que juntos formam a GUI.

4.1.2. Diagrama de componentes

O diagrama de componentes, apresentado na Figura 4.3, modela as "peças" – denominadas componentes – de que o sistema necessita para realizar suas tarefas. Essas peças incluem os executáveis binários, os arquivos tipo *.class* compilados, os arquivos tipo *.java*, imagens, pacotes, recurso etc.

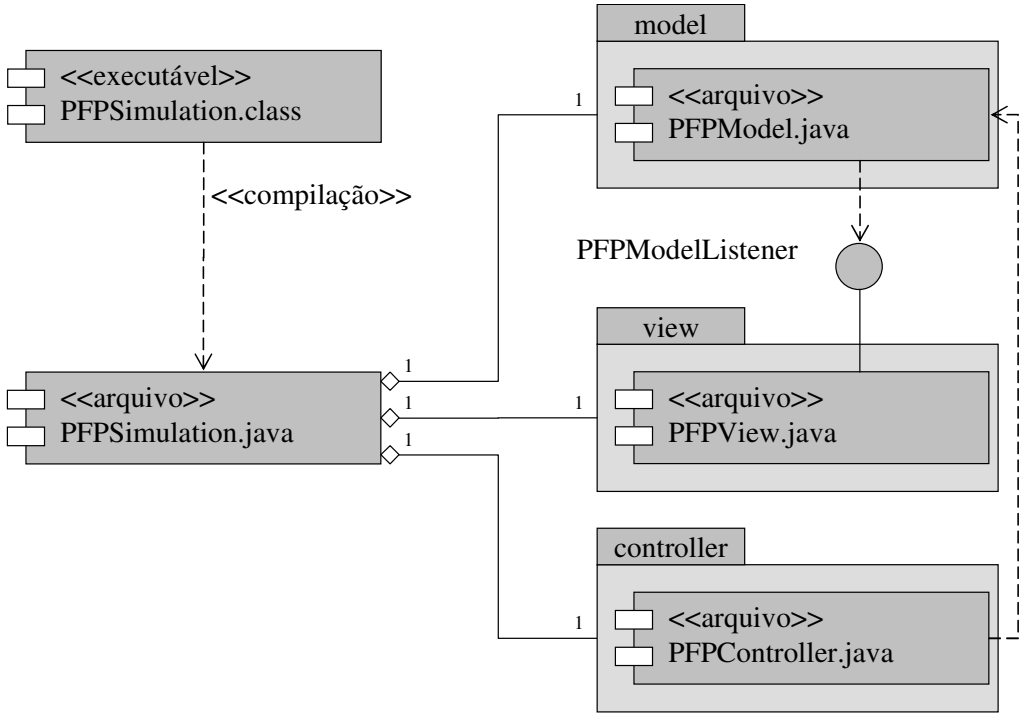

Figura 4.3 – Diagrama de componentes para o sistema PFP.

Na Figura 4.3, cada caixa que contém as duas pequenas caixas brancas sobrepostas em seu lado esquerdo é um componente. Desenhou-se o componente, conforme a UML, de forma similar a um *plug* (as duas caixas sobrepostas representam os pinos do *plug*) – permitindo a conexão de um componente a outros sistemas sem haver a obrigatoriedade de alteração do componente. O sistema PFP contém cinco componentes: *PFPSimulation.class*, *PFPSimulation.java*, *PFPModel.java*, *PFPView.java* e *PFPController.java*. Na Figura 4.3 os elementos com forma de pastas (caixas com pequenos retângulos no canto superior esquerdo) representam *pacotes* em UML. Classes, objetos, componentes, casos de uso, entre outros, podem ser agrupadas em pacotes. No

diagrama em questão, os pacotes UML se referem a pacotes Java. Os pacotes do sistema PFP são *model*, *view* e *controller*.

As setas tracejadas na Figura 4.3 indicam uma dependência entre dois componentes – a direção da seta indica o relacionamento "depende de". A dependência descreve o relacionamento entre os componentes no qual as alterações feitas em um componente afetam outro componente. Assim, o componente *PFPController.java* depende de *PFPModel.java* e o componente *PFPSimulation.class* depende do componente *PFPSimulation.java*. Portanto, quando uma alteração é feita em *PFPModel.java* ou em *PFPSimulation.java*, as classes *PFPController.java* e *PFPSimulation.class*, respectivamente, são afetadas.

Pode ser observado também na Figura 4.3 que *PFPModel.java* e *PFPView.java* não dependem um do outro – eles se comunicam através da interface *PFPModelListener*, que implementa todas as interfaces da simulação. Dessa forma, *PFPView.java* concretiza a interface *PFPModelListener* e *PFPModel.java* depende da interface *PFPModelListener*.

Portanto, o diagrama de dependência de componentes ajuda a agrupar componentes para a reutilização em sistema futuros. Por exemplo, *PFPModel.java* pode ser reutilizado em outros sistemas sem ser obrigatória a reutilização de *PFPView.java* (e vice-versa), porque esses componentes não são dependentes um do outro. Todavia, se a reutilização de *PFPController.java* for desejável, será necessário reutilizar *PFPModel.java*.

Finalmente, os itens (palavras) colocados entre os símbolos de aspas francesas, << >>, são estereótipos que indicam o papel de um elemento. Assim, o estereótipo <<compilação>> descreve a dependência entre *PFPSimulation.class* e *PFPSimulation.java*. O estereótipo <<executável>> especifica que o componente é um aplicativo, ou seja, é um programa que é executado com o interpretador Java. E, por último, o estereótipo <<arquivo>> especifica que o componente é um arquivo que contém o código-fonte para o executável.

4.1.3. Diagramas de classes

Para se mostrar a aplicação da arquitetura MVC no programa PFP, foi desenvolvido um diagrama de classes de "alto nível" apresentado na Figura 4.4. A classe *PFPSimulation* – uma subclasse de *JFrame* – agrega uma instância de

cada uma das classes *PFPModel*, *PFPView* e *PFPController* para criar o apliativo *PFPSimulation*. O retângulo com o canto superior “dobrado” representa uma notação em UML. Nesse caso, cada notação aponta uma classe específica (por meio da linha tracejada) para descrever o papel daquela classe no sistema. Portanto, as notações *model*, *view*, *controller* demonstram que as classes *PFPModel*, *PFPView* e *PFPController* encapsulam todos os objetos que compreendem o modelo, a visão e o controlador de simulação, respectivamente. E a notação *aplicativo* mostra que a classe PFPSimulation é responsável pela execução do programa PFP.

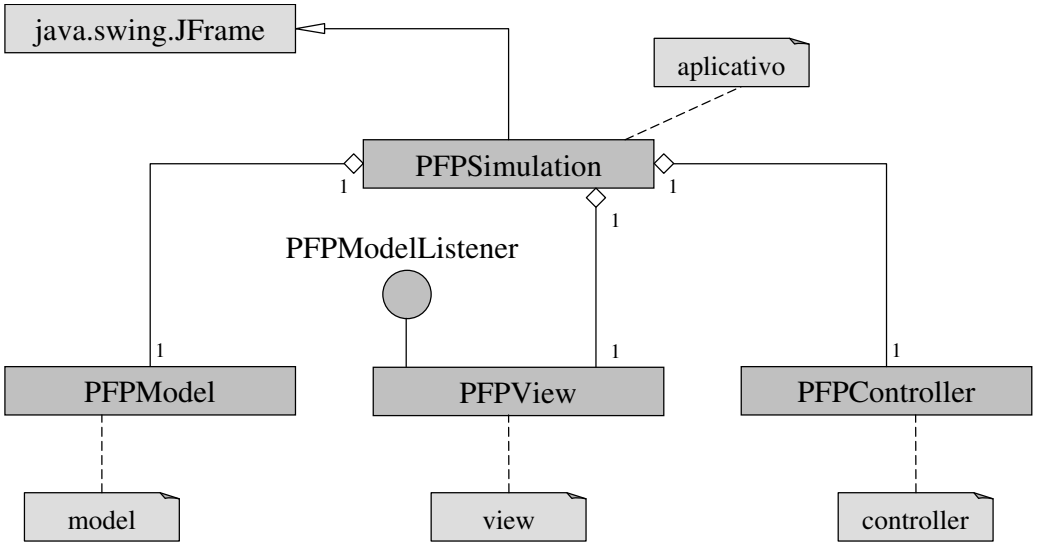

Figura 4.4 – Diagrama de classes do sistema PFP.

A classe *PFPView*, que é uma agregação de classes, implementa a interface *PFPModelListener*, que por sua vez, implementa todas as interfaces usadas na simulação, de modo que *PFPView* possa receber todos os eventos do modelo. A classe *PFPController* representa o controlador da simulação.

A classe *PFPSimulation* não contém outros atributos além de suas referências para um objeto *PFPModel*, um objeto *PFPView* e um objeto *PFPController*. O único comportamento para a classe *PFPSimulation* é iniciar o programa. Portanto, em linguagem Java, a classe *PFPSimulation* contém um método *static main* que chama o construtor, o qual instancia os objetos *PFPModel*, *PFPView* e *PFPController*.

4.2.
DIAGRAMA DE CLASSE PARA O SUBSISTEMA *MODEL*

As classes do modelo são apresentadas em um diagrama de classes, no qual cada uma foi modelada como um retângulo, conforme apresenta a Figura 4.5. Esse diagrama usa a supressão dos atributos e das operações das classes para tornar o diagrama mais legível, sendo denominados de diagramas elididos. Desta forma, as operações das classes do subsistema model e os atributos serão descritos em outro item adiante.

O sistema *PFP* requer uma única classe para representar o modelo. Esta classe, chamada de *PFPModel*, se refere implicitamente a todas as classes que compõem o modelo. Tal classe age como a "representante" do modelo, porque agrega todas as outras que o compõem.

Nesse diagrama, as classes que se relacionam umas com as outras através de associações são conectadas por uma linha cheia. Os números próximos às linhas expressam valores de multiplicidade. Os valores de multiplicidade indicam quantos objetos de uma classe participam da associação.

O losango colocado no final de uma linha de associação, ao lado do retângulo da classe, indica que a classe tem um relacionamento de agregação com a classe a qual esta conectada, ou seja, ela agrega um ou mais objetos da outra classe.

As associações são direcionais, sendo a direção indicada por uma seta junto ao nome da associação, que nesse caso, é a classe *PredictModel*.

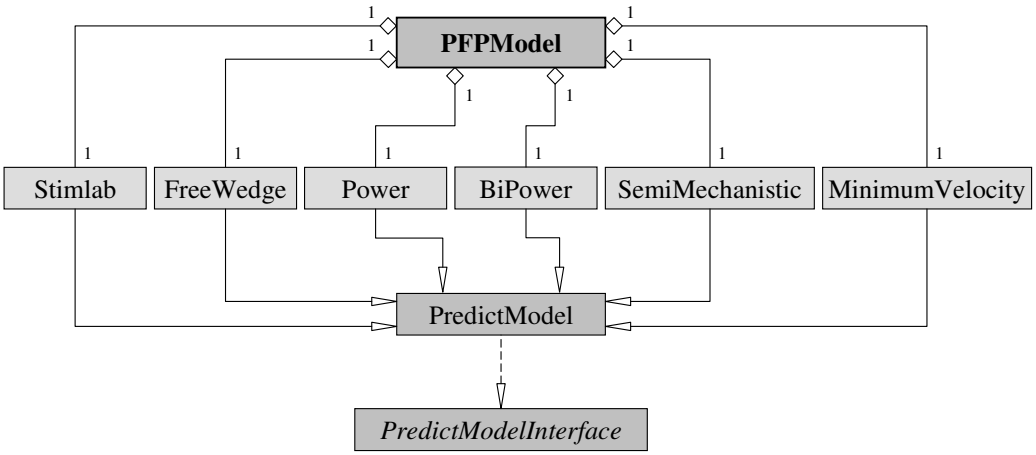

Figura 4.5 – Diagrama de classe elidido para o subsistema modelo.

De acordo com o diagrama, a classe *PFPModel* agrega um objeto de cada uma das classes específicas *Stimlab*, *FreeWedge*, *Power*, *BiPower*, *SemiMechanistic* e *MinimumVelocity*. Essas classes específicas herdam a classe básica *PredictModel*, que por sua vez pré-implementa a interface *PredictModelInterface*.

Essa forma de escrever "Interface (*PredictModelInterface*) – Classe Básica (*PredictModel*) – Classe Específica (*Stimlab*, *FreeWedge*, *Power*, *BiPower*, *SemiMechanistic* e *MinimumVelocity*)" possibilita escrever toda a GUI do programa em função da classe básica. Desse modo, se evita alterar as classes que implementam a GUI a cada nova classe específica adicionada ao programa.

A UML exige que nomes de classes abstratas (e métodos abstratos) sejam escritos em itálico, como é o caso da classe *PredictModelInterface*. Além disso, a UML oferece um relacionamento chamado generalização para modelar herança. Assim, as setas com pontas vazadas especificam a relação de herança entre as classes. Portanto, a Figura 4.5 mostra que as classes *Stimlab*, *FreeWedge*, *Power*, *BiPower*, *SemiMechanistic* e *MinimumVelocity* herdam a classe *PredictModel*, como já descrito anteriormente.

O conceito de herança foi aplicado, nesse caso, da seguinte forma: primeiramente, foram identificados os parâmetros comuns entre as classes; a seguir, se extraíram tais parâmetros, colocando-os em uma superclasse; e, finalmente, as subclasses foram derivadas a partir da superclasse.

O relacionamento entre uma classe e uma interface é expresso através de uma realização. A classe realiza, ou implementa, os comportamentos de uma interface. Esse relacionamento é expresso por uma seta tracejada, como pode ser observado na Figura 4.5, onde a classe *PredictModel* pré-implementa a interface *PredictModelInterface*.

4.2.1. Atributos e operações das classes

Os atributos e operações das classes foram reunidos em diagramas de classes completos apresentados na Figura 4.6. Como pode ser observado, cada diagrama representa uma classe e é constituído por três partes: a parte superior contém o nome da classe, a parte do meio, os atributos e a parte inferior, as operações.

Os atributos de cada classe descritos a seguir não incluem as referências a outros objetos e representam apenas os atributos principais de cada classe.

Três componentes de informação descrevem os atributos: o nome, o tipo e o valor inicial. Se o atributo não tiver um valor inicial especificado, somente seu nome e tipo são mostrados.

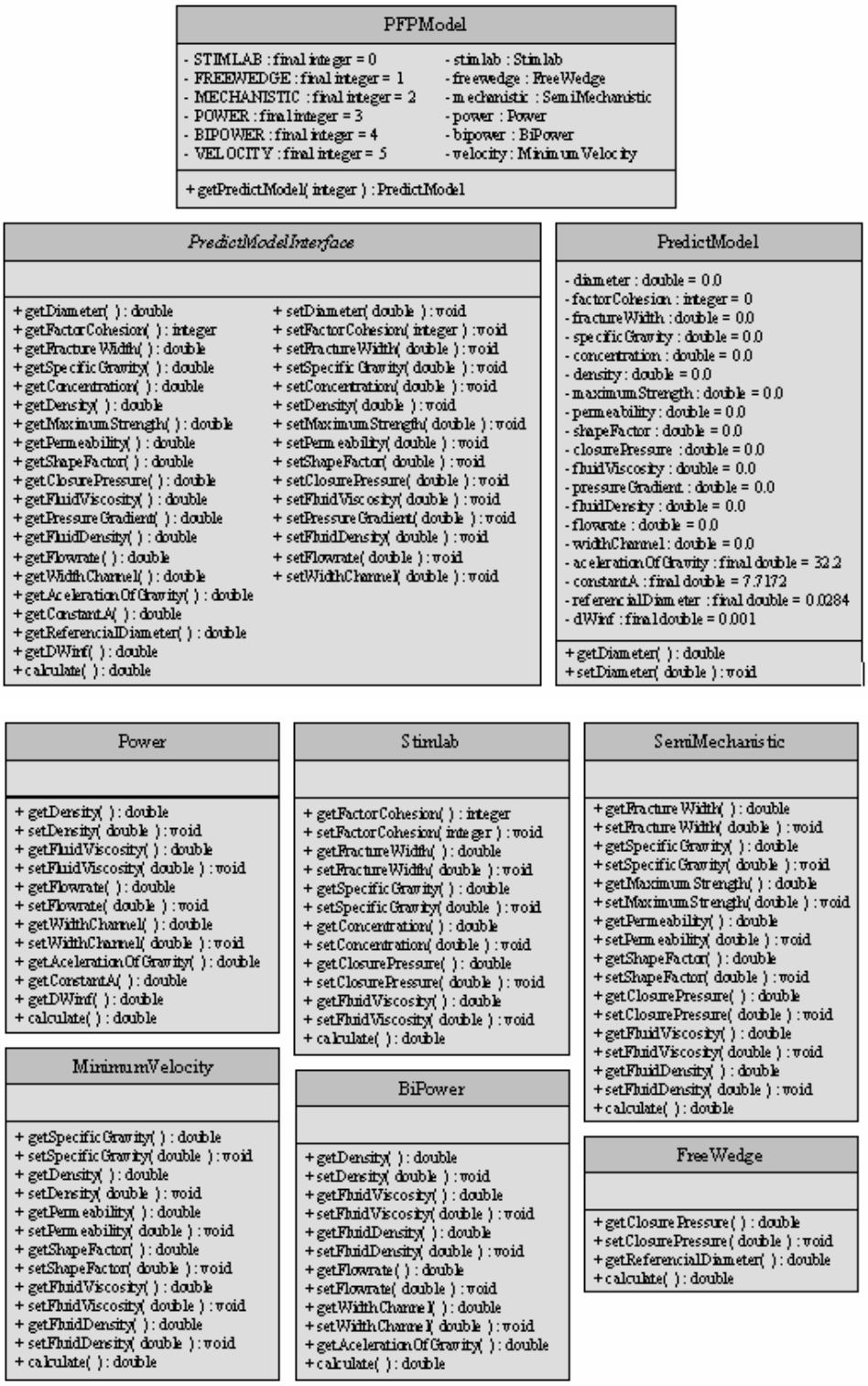

Figura 4.6 – Classes com atributos, operações e notações de visibilidade.

Nos diagramas de classes da Figura 4.6, os nomes das operações são representados como métodos e incluem as informações sobre o tipo de valor devolvido depois do símbolo dois-pontos.

Ainda, os diagramas de classes da Figura 4.6 listam atributos e operações para algumas das principais classes com notações de visibilidade. Na UML, a visibilidade *public* é indicada através de um sinal (+) antes do elemento particular, ou seja, o método ou atributo de um objeto dessa classe pode ser acessado por qualquer outro objeto (visibilidade externa total). O sinal (-) indica a visibilidade *private*, ou seja, o método ou atributo de um objeto dessa classe não pode ser acessado por nenhum outro objeto (nenhuma visibilidade externa).

O compartimento do meio da interface PredictModelInterface está vazio, porque as interfaces podem conter constantes; no entanto, essa interface, em particular, não contém constante alguma.

A classe *PFPModel* age como o "representante" para o modelo, visando a interação com outras partes do sistema, de modo que os atributos de configuração de todo o aplicativo sejam concedidos a essa classe.

As classes *Stimlab*, *FreeWedge*, *Power*, *BiPower*, *SemiMechanistic* e *MinimumVelocity* não possuem atributos, visto que os mesmos foram declarados na classe básica *PredictModel*. Já a interface *PredictModelInterface* define os métodos básicos para todos os modelos de previsão de refluxo de propante, enquanto que a classe *PredictModel* fornece uma pré-implementação para esses métodos. Elimina-se, desta forma, a exigência de que todas as classes que herdarem *PredictModel* implementem todos os métodos, inclusive aqueles que não são do interesse das mesmas.

4.3. DIAGRAMA DE COMPONENTES DO PACOTE *MODEL*

A Figura 4.7 apresenta o diagrama de componentes para o pacote *model*. O pacote *model* agrega o pacote *event*, ou seja, cada componente *model* contém uma agregação com todos os componentes do pacote *event*.

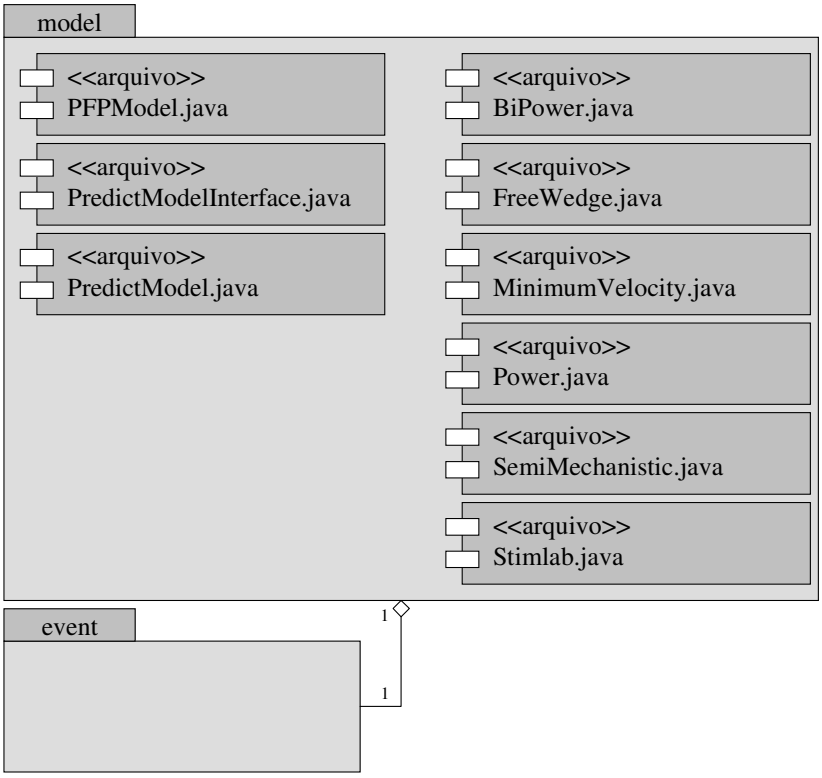

Figura 4.7 – Diagrama de componentes do pacote model.

4.4. DIAGRAMA DE COMPONENTES DO PACOTE *EVENT*

Evento é uma mensagem que notifica um objeto de uma ação que já aconteceu. Normalmente, os eventos são gerados quando o usuário do programa interage com a GUI.

Nesse aplicativo, *PFPView* e *PFPModel* importam o pacote *event*, cujo diagrama de componentes é apresentado na Figura 4.8.

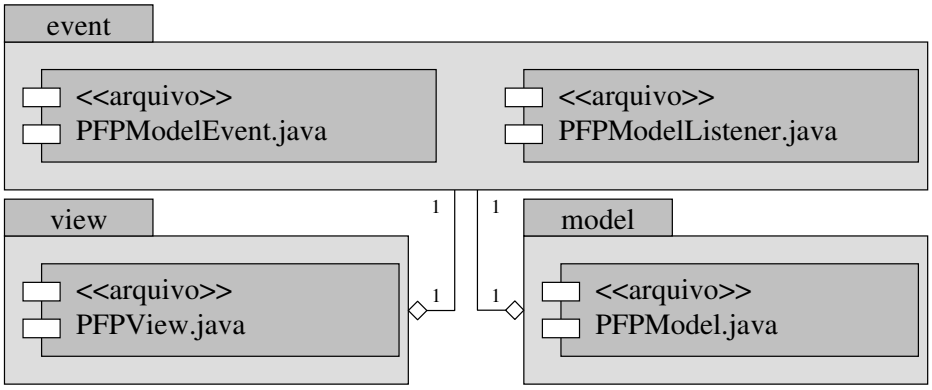

Figura 4.8 – Diagrama de componentes para o pacote event.

Foi criada uma superclasse chamada *PFPModelEvent* para representar um evento no sistema. *PFPModelEvent* contém uma referência *Object* para a origem do evento. Os objetos do sistema usam instâncias de *PFPModelEvent* para enviar eventos para outros objetos. Quando um objeto recebe um evento, aquele objeto pode usar o método *getSource* para determinar a origem do evento.

Implementada a superclasse *PFPModelEvent* foi necessário implementar a interface que "espera" os eventos, ou seja, *PFPModelListener*. Essa interface oferece os métodos para o ouvinte de evento e é chamada de interface de ouvinte de evento.

4.4.1. Tratamento de eventos

O conceito de tratamento de eventos em linguagem Java é similar ao conceito de colaboração: consiste em um objeto de uma classe enviar uma mensagem particular (o que Java denomina de um *evento*) para os objetos de outras classes que esperam aquele tipo de mensagem. Entretanto, os objetos que recebem a mensagem precisam ser registrados, ou seja, transformados em ouvintes de eventos (*event listeners*). Para enviar um evento, o objeto remetente invoca um método particular do objeto destinatário, passando o evento desejado como um parâmetro. No sistema PFP, tal objeto pertence à classe que estende *PFPModelEvent*.

O mecanismo de tratamento de eventos possui três partes: a origem do evento, o objeto evento e o ouvinte (listener) do evento. A origem do evento é o componente GUI particular com o qual o usuário interage. O objeto evento encapsula as informações sobre o evento que ocorreu. Essas informações incluem uma referência para a origem do evento e quaisquer outros dados específicos que possam ser necessários para que o ouvinte de evento trate desse evento.

4.4.2. Ouvintes de eventos

O ouvinte do evento é um objeto que é notificado do evento e, então, usa o objeto para responder ao evento. A origem do evento também precisa manter uma lista de seus ouvintes registrados e ser capaz de notificar seus ouvintes quando ocorre um evento.

Para processar um evento da interface gráfica com o usuário, além de ser necessário o registro de um ouvinte de evento para o componente GUI, também é necessária a implementação de um método de tratamento de eventos (ou conjuntos de métodos de tratamento de eventos). O ouvinte de eventos para um evento GUI é um objeto de uma classe que implementa uma ou mais classes das interfaces *listeners* de eventos dos pacotes *java.awt.event* e *javax.swing.event*.

4.5. INTERFACE GRÁFICA DO SISTEMA PFP

A interface gráfica do sistema PFP foi dividida em três partes principais. A primeira, localizada na parte superior da interface, foi destinada à identificação do cenário que está sendo avaliado, contendo, portanto, campos para o preenchimento das informações sobre a identificação, localização e profundidade do poço, fluido de fraturamento, técnica de fraturamento, material e porosidade da rocha-reservatório, vazão de produção e zona de contribuição. Além disso, essa parte contém os campos de seleção para a escolha do modelo de previsão de refluxo de propante e do tipo de propante.

A segunda parte foi dividida em três colunas. A da esquerda foi reservada para as propriedades do propante, como densidade, concentração, massa específica, resistência ao esmagamento e esfericidade do propante. Contém, ainda, um campo de seleção para a escolha do diâmetro médio das partículas e um campo para a determinação da largura da fratura, que apesar de não ser uma propriedade do propante, está relacionada à granulometria do mesmo. Outro parâmetro que está presente nessa parte é o fator de coesão (Stimlab, 2002), que depende do tipo de tratamento aplicado ao propante e é exclusivo ao modelo Stimlab. A coluna central contém uma coluna com os dados relacionados ao poço e à rocha-reservatório, como a tensão de fechamento, viscosidade e massa específica do fluido, gradiente hidráulico, vazão de produção e velocidade do fluido. A coluna da direita da interface mostra as constantes utilizadas pelos modelos e seus respectivos valores, que não podem ser alterados pelo usuário.

A última parte da interface está localizada na parte inferior e apresenta o botão que calcula a variável crítica para o modelo de previsão de refluxo de propante selecionado e o campo para a visualização do valor da mesma. Como a variável crítica depende do modelo selecionado, nenhuma unidade é apresentada.

Todos os parâmetros devem ser determinados na unidade indicada ao lado do campo específico.

Como todo *software*, existe ainda uma barra de ferramentas e o menu *File* com as seguintes opções: novo arquivo, abrir, salvar, gerar relatório e imprimir.

A Figura 4.9 apresenta a interface gráfica do sistema PFP.

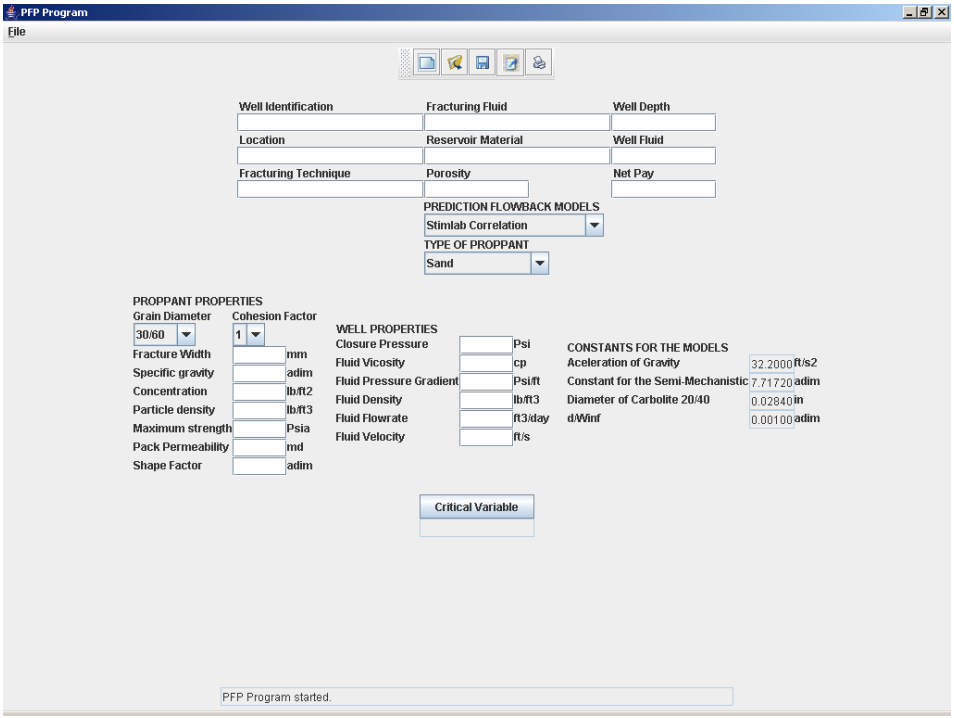

Figura 4.9 – Interface gráfica do PFP System.

A Figura 4.10 apresenta o campo para a seleção do modelo de previsão de refluxo de propante. Observa-se que a caixa de diálogo inferior é modificada de acordo com o modelo selecionado. A Figura 4.11 apresenta o campo para a seleção do tipo de propante, que são: areia, areia tratada com resina, bauxita, cerâmica de resistência intermediária, cerâmica de alta resistência e carbolite (propante fornecido pela Carboceramics). A Figura 4.12 apresenta o campo para a seleção da granulometria do propante; o valor do diâmetro médio está embutido no sistema. A Figura 4.13 apresenta o campo para a seleção do fator de coesão. E, finalmente, a Figura 4.14 apresenta um exemplo de aplicação para a visualização do valor da variável crítica.

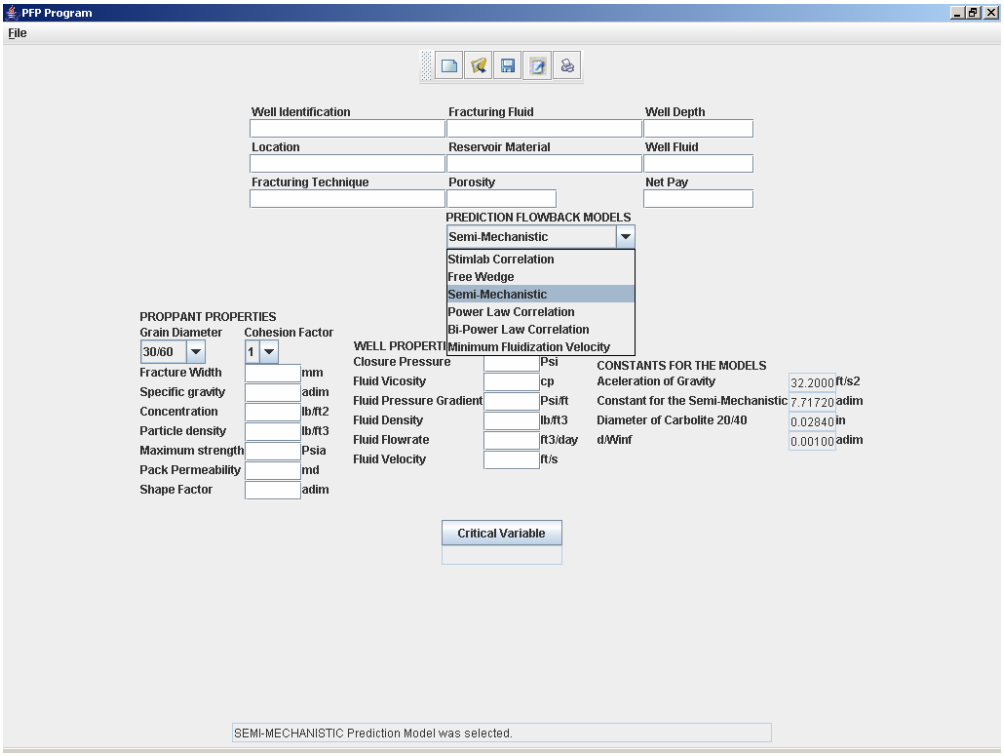

Figura 4.10 – Visualização do campo de seleção do modelo de previsão desejado.

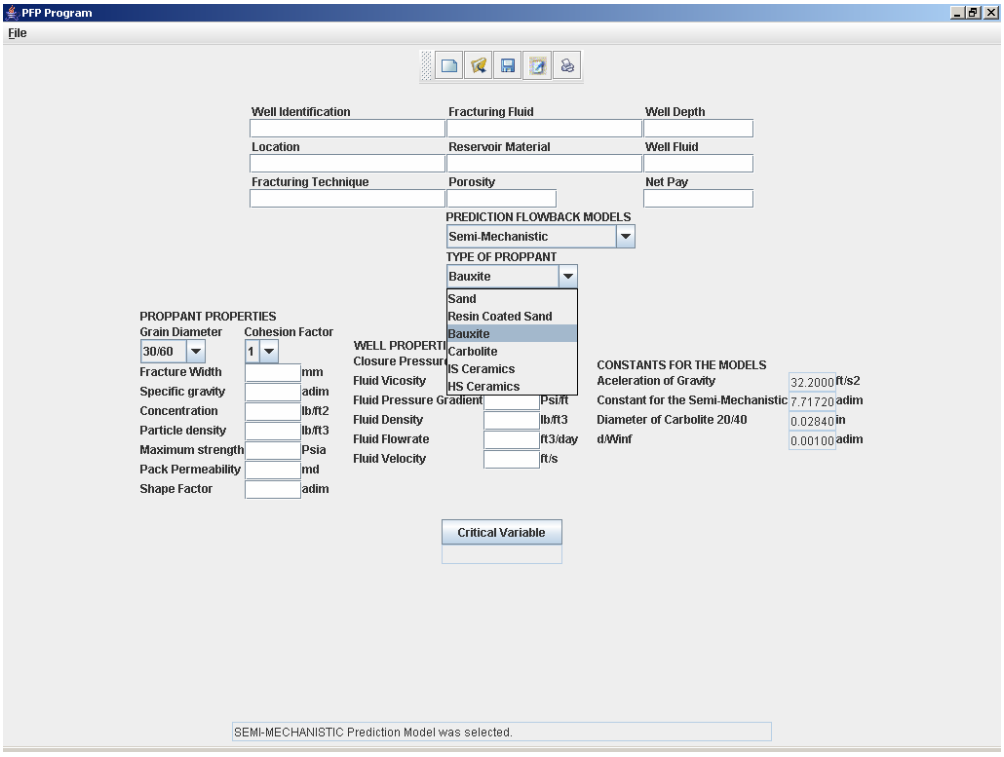

Figura 4.11 – Visualização do campo de seleção do tipo de propante.

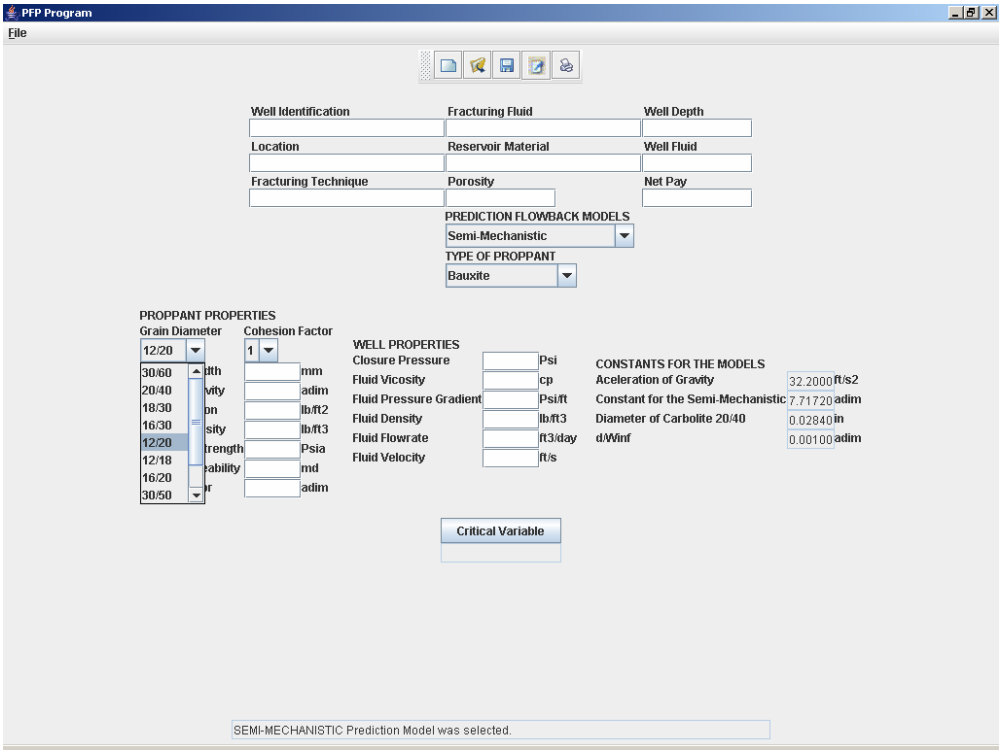

Figura 4.12 – Visualização do campo de seleção do diâmetro médio do grão.

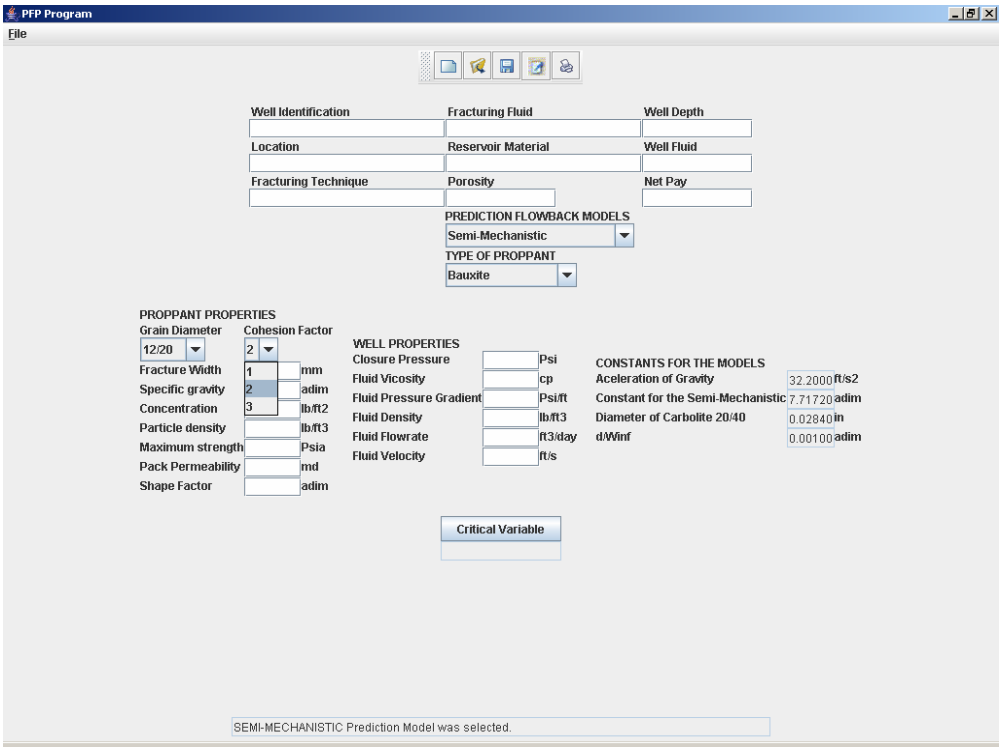

Figura 4.13 – Visualização do campo de seleção do fator de coesão.

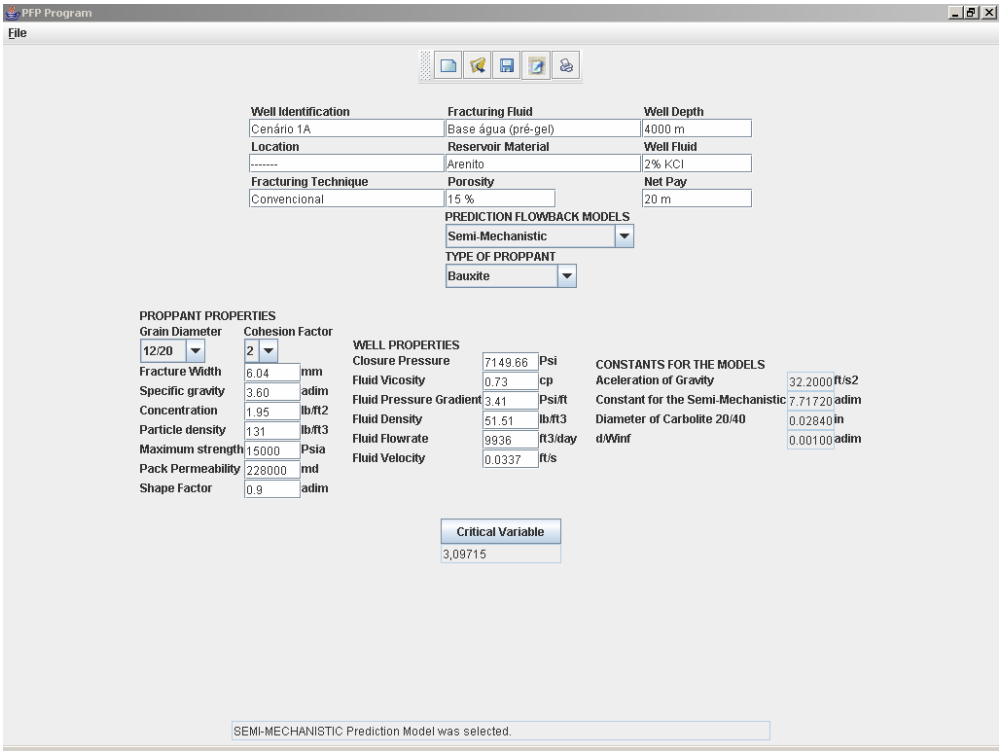

Figura 4.14 – Exemplo de aplicação do PFP System.