UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Arthur Hortmann Erpen

DESENVOLVIMENTO DE UM MÓDULO DE REGRAS NO SO-BR, SISTEMA DE ACOMPANHAMENTO E APOIO À TOMADA DE DECISÃO EM PERFURAÇÕES DE POÇOS DO PRÉ-SAL

Florianópolis
2016

Arthur Hortmann Erpen

DESENVOLVIMENTO DE UM MÓDULO DE REGRAS NO SO-BR, SISTEMA DE ACOMPANHAMENTO E APOIO À TOMADA DE DECISÃO EM PERFURAÇÕES DE POÇOS DO PRÉ-SAL

Trabalho de conclusão de curso submetido ao curso de Bacharelado em Ciência da Computação para a obtenção do Grau de Bacharel em Ciência da Computação.
Orientador: Prof. Dr. Paulo José de Freitas Filho

Florianópolis
2016

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Erpen, Arthur
DESENVOLVIMENTO DE UM MÓDULO DE REGRAS NO SO-BR, SISTEMA DE ACOMPANHAMENTO E APOIO À TOMADA DE DECISÃO EM PERFURAÇÕES DE POÇOS DO PRÉ-SAL / Arthur Erpen ; orientador, Paulo Freitas, 2017.
31 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2017.

Inclui referências.

1. Ciências da Computação. 2. Inteligência Artificial. 3. Sistemas Especialistas. 4. Regras. 5. CLIPS. I. Freitas, Paulo. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

Arthur Hortmann Erpen

DESENVOLVIMENTO DE UM MÓDULO DE REGRAS NO SO-BR, SISTEMA DE ACOMPANHAMENTO E APOIO À TOMADA DE DECISÃO EM PERFURAÇÕES DE POÇOS DO PRÉ-SAL

Este Trabalho de conclusão de curso foi julgado aprovado para a obtenção do Título de “Bacharel em Ciência da Computação”, e aprovado em sua forma final pelo curso de Bacharelado em Ciência da Computação.

Florianópolis, 30 de outubro de 2017.

_____

Prof. Dr. Renato Cislaghi
Coordenador de Projetos

Banca Examinadora:

_____

Prof. Dr. Paulo José de Freitas Filho
Orientador

_____

Prof. Dr. Mauro Roisenberg

_____

Prof. Dr. Sílvia Modesto Nassar

AGRADECIMENTOS

Agradeço à minha mãe Estela, à minha irmã Thariane e à toda minha família por todo apoio e incentivo necessários para que eu tenha chegado até aqui.

Agradeço ao meu namorado Leonardo e à minha amiga Aline por toda parceria e ajuda em momentos de tensão.

Agradeço ao professor Dr. Paulo José de Freitas pela oportunidade e orientação para a realização deste trabalho.

Agradeço aos professores membros da banca Mauro Roisenberg e Sílvia Modesto Nassar por aceitarem o convite e pelas contribuições com a avaliação deste trabalho.

Agradeço à toda equipe do PerformanceLab pela oportunidade de aprendizado, confiança, e pela compreensão em momentos difíceis que cada integrante depositou em mim.

RESUMO

O software SO-BR é um sistema de acompanhamento e apoio à tomada de decisão em perfurações de poços do pré-sal. Em alguns cenários de perfuração, a abordagem utilizada pelo software, que combina as técnicas de IA Neuro-Fuzzy e Redes Bayesianas, demonstra não ser adequada. Este trabalho apresenta a programação de um módulo de regras que busca possibilitar que engenheiros insiram no SO-BR parte de seu conhecimento através de regras, a fim de aperfeiçoar a capacidade do software. A ferramenta CLIPS foi utilizada para a programação da engine do módulo, provendo o mecanismo de verificação das regras. Um editor gráfico de regras foi programado para que os engenheiros não precisem dominar a linguagem não-natural das regras utilizada pela engine. A ênfase deste trabalho é na área de petróleo porém a abordagem utilizada é aplicável a diversas outras áreas.

Palavras-chave: Inteligência Artificial, Sistemas Especialistas, Sistemas Baseados em Regras, CLIPS, Cox-Stuart

ABSTRACT

SO-BR software is a monitoring and decision support system for pre-salt well drilling. In some drilling scenarios, the software approach, which combines artificial intelligence techniques Neuro-Fuzzy and Bayesian Networks, proves to be inadequate. This work presents the programming of a rules module that seeks to enable engineers to insert in SO-BR part of their knowledge through rules in order to improve the software's performance. The CLIPS tool was used to program the module engine, providing the mechanism for matching rules. A graphical rule editor has been programmed so that engineers do not need to master the unnatural language of the rules used by the engine. The emphasis of this work is in the area of oil but the approach used is applicable in other areas.

Keywords: Artificial Intelligence, Expert Systems, Rule Based Systems, CLIPS, Cox-Stuart

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

IA Inteligência Artificial
SBC Sistemas Baseados em Conhecimento
SE Sistemas Especialistas
SP Sistemas de Produção
CLIPS *C Language Integrated Production System*
ROP *Rate Of Penetration*
PSB Peso Sobre Broca
RPM Rotações Por Minuto

SUMÁRIO

1 INTRODUÇÃO

A humanidade sempre esteve em busca da evolução. A civilização surgiu a partir do advento da agricultura, quando os humanos que antes viviam permanentemente viajando em busca de alimentos puderam se fixar e formar as primeiras cidades. Com o desenvolvimento do conhecimento e da tecnologia, as máquinas surgiram, poupando o trabalho humano e modificando completamente o modo de produção da sociedade, dando início a era industrial. Quando o conhecimento científico se uniu à produção industrial, profundas evoluções no campo tecnológico foram desencadeadas, possibilitando o avanço da eletrônica e da computação, que fez com que a humanidade emergisse para a era digital em que vivemos hoje. Tal progresso facilitou a resolução de problemas que anteriormente dependiam de muito esforço para serem solucionados.

Enquanto que alguns dos problemas computacionais tratam apenas da manipulação de dados, como a ordenação de uma sequência de números, outros problemas envolvem o uso do raciocínio e pensamento humano, como jogar xadrez ou entender uma linguagem natural. A Inteligência Artificial é o ramo da Ciência da Computação que busca a automação do comportamento inteligente (LUGER, 2013).

Este trabalho tem como objetivo o desenvolvimento de um módulo de regras integrado ao software SO-BR, um sistema de acompanhamento e apoio à tomada de decisão em perfurações de poços do pré-sal, o qual é resultado de um projeto de pesquisa entre a UFSC (PerformanceLab/INE) e a Petrobras (Cenpes) que teve início há cerca de cinco anos. Em seu desenvolvimento verificou-se que seria de grande valia se os engenheiros que utilizam o programa pudessem configurar alertas a serem disparados em determinadas condições. Por exemplo, sendo registrada uma tendência constante para a taxa de perfuração, simultânea a uma tendência crescente para o peso sobre a broca, um alerta de desgaste de broca poderia ser emitido, incluindo opcionalmente sugestões de alterações em variáveis operacionais da perfuração.

Para a programação do módulo, uma pesquisa foi realizada em busca da ferramenta que melhor se adequasse ao SO-BR. Por prover um ambiente completo para a construção de sistemas baseados em regras, possuir código fonte aberto e ser escrita em linguagem compatível ao ambiente de desenvolvimento do programa, a ferramenta CLIPS foi escolhida.

Como as regras interpretadas pela ferramenta são escritas em sua própria linguagem não natural, também é objetivo do trabalho a programação de um editor gráfico, para facilitar a criação e edição das regras, de modo que os engenheiros não precisem estudar a ferramenta CLIPS para utilizar o sistema.

Embora a ênfase do trabalho seja na área de petróleo, é importante salientar que as técnicas empregadas podem ser utilizadas em diversas outras instâncias.

1.1 JUSTIFICATIVA

O SO-BR faz uso de sistemas Neuro-Fuzzy e Redes Bayesianas para apoiar a tomada de decisão durante a perfuração de poços do pré-sal. Quando a taxa de perfuração se encontra distante do esperado, o programa indica alterações a serem realizadas em parâmetros operacionais, como o peso sobre a broca.

Em algumas situações, essa abordagem demonstra não ser adequada. Em um caso de desgaste na broca, por exemplo, pode ser mais interessante iniciar o processo de sua substituição ao invés de continuar a perfuração com alterações nos parâmetros. Para cobrir essas situações, é necessário inserir o conhecimento técnico do engenheiro no programa, de modo que alertas possam ser exibidos quando determinadas condições, envolvendo a tendência de variáveis da perfuração, são verdadeiras. O uso de regras demonstra ser uma ótima alternativa para a solução desse problema.

A ferramenta CLIPS possui diversos fatores positivos que motivaram sua escolha. Sendo de domínio público, não há necessidade de preocupar-se com aspectos relacionados a sua licença. Por ser desenvolvida desde 1985 e por sua ampla popularidade, há uma grande documentação disponível na internet, facilitando o estudo de sua utilização. Outro motivo importante para seu uso é o fato de ser escrita na linguagem C, compatível à linguagem C++, utilizada no desenvolvimento do SO-BR, tornando desnecessária a preocupação com integração de diferentes linguagens, tarefa que costuma requerer muito esforço.

Um editor gráfico para a criação das regras demonstra ser essencial. Sem o mesmo, os engenheiros que utilizam o programa precisariam estudar a fundo a sintaxe da linguagem da ferramenta, não natural e de difícil compreensão.

1.2 PROBLEMA A SER TRATADO

O problema que será tratado neste trabalho é a necessidade de aperfeiçoar o apoio à tomada de decisão do SO-BR, tendo em vista que no programa são utilizadas técnicas de aprendizagem de máquina, que demonstram ser ineficientes em alguns cenários de perfuração. Para isso, a inserção de conhecimento especialista no programa se faz necessária.

1.3 OBJETIVOS

Nesta seção, são apresentados os objetivos gerais e específicos deste trabalho.

1.3.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um módulo de regras para o programa SO-BR, composto por uma engine e um editor gráfico de regras. A engine é responsável pelo mecanismo de verificação e ativação das regras. Ela deve receber as sequências de valores para cada variável da perfuração, determinar suas tendências, e realizar um cruzamento entre as regras, informando quais foram ativadas.

O editor gráfico de regras é uma ferramenta de interface gráfica, que possibilita que engenheiros criem regras sem conhecerem a linguagem da ferramenta CLIPS, utilizada pela engine. Ele deve traduzir as regras definidas a partir de informações inseridas em sua interface para o formato adequado à engine.

1.3.2 Objetivos Específicos

1. Estudar o algoritmo do teste de Cox-Stuart para análise de tendência em sequências de dados.
2. Investigar o uso da ferramenta CLIPS
3. Definir as possibilidades de condições e ações para as regras
4. Analisar a manutenção da consistência entre regras
5. Projetar a aparência e usabilidade do editor gráfico de regras
6. Examinar diferentes formas de persistência para as regras

1.4 METODOLOGIA

O desenvolvimento do módulo de regras é dividido em duas etapas. Inicialmente, a engine do módulo de regras é programada e testada. Para isso, a documentação da ferramenta CLIPS é amplamente estudada, através de sua documentação disponível na internet. Nessa etapa, a engine é testada individualmente, com regras e dados programados em seu próprio código.

Em seguida, a etapa de desenvolvimento do editor gráfico de regras é iniciada, com um estudo buscando comparar editores gráficos de regras de diferentes aplicações disponíveis no mercado, a fim de escolher qual modelo de interface gráfica melhor se adequa a este trabalho. A partir daí, o editor é programado e a tradução de regras para o formato da engine é testada e validada.

Por fim, o módulo de regras será integrado ao SO-BR e testado em conjunto com o módulo de controle do programa, simulando uma perfuração real que ative regras estabelecidas pelo editor.

1.5 ESTRUTURA DO TRABALHO

Este trabalho está dividido em 4 capítulos. No capítulo 1, é feita uma introdução ao tema, justificando o trabalho, apresentando seus objetivos, metodologia e estrutura. No capítulo 2, é realizado uma análise do estado da arte, onde cada um dos temas envolvidos no trabalho é apresentado detalhadamente. No capítulo 3, a proposta do trabalho é abordada, detalhando o processo de desenvolvimento do módulo de regras. O capítulo 4 trata da conclusão do trabalho.

2 ESTADO DA ARTE

Neste capítulo, são apresentados os principais temas envolvidos no trabalho. Inicialmente, temas chaves tratados no trabalho são explicados. Em seguida, o teste de Cox-Stuart, utilizado para a análise de tendência em sequências de dados, é abordado. Por fim, a ferramenta CLIPS é apresentada.

2.1 SISTEMAS BASEADOS EM CONHECIMENTO

Um grande campo de estudo da Inteligência Artificial é o desenvolvimento de Sistemas Baseados em Conhecimento (SBCs), softwares que solucionam problemas através de conhecimento representado explicitamente. Esse tipo de sistema vem sendo utilizado amplamente, tanto na área acadêmica quanto na área comercial. Seu desenvolvimento representa um grande avanço tecnológico, pois possibilitou a resolução de problemas que antes só eram resolvidos por seres humanos (REZENDE, 2003).

Segundo (SCHREIBER; WIELINGA; BREUKER, 1993), o maior desafio na construção de um SBC é encontrar a resposta adequada para a pergunta de como modelar o conhecimento. É esse aspecto que diferencia o desenvolvimento de SBCs do desenvolvimento de sistemas tradicionais.

No campo dos SBCs, três níveis podem ser reconhecidos: o nível do usuário final, o nível do engenheiro de conhecimento e o nível das ferramentas utilizadas para construção dos sistemas.

Para o usuário final, um SBC é composto essencialmente de três componentes: uma engine, a interface de usuário e uma base de conhecimento.

Uma interface de usuário com boa usabilidade é quase sempre essencial em SBCs, pois o usuário final do sistema normalmente não possui domínio em computação.

A base de conhecimento é uma das principais características de um SBC, na qual o conhecimento humano é representado de maneira simbólica. A partir daí, a engine faz uso de um mecanismo de raciocínio para realizar inferências e obter conclusões.

O engenheiro de conhecimento é responsável por definir o tipo de conhecimento, sua organização, sua representação e a maneira de utilizá-lo para processar os dados que são providos pelo usuário e fornecer a solução. Isso é alcançado pela interação entre o engenheiro e o especialista do domínio.

2.2 SISTEMAS ESPECIALISTAS

No campo dos SBCs, os Sistemas Especialistas (SEs) têm sido os mais eficazes na resolução de problemas. São programas que resolvem problemas específicos a um determinado domínio ou auxiliam a tomada de decisão, da mesma forma e com nível similar a especialistas do domínio. Diagnóstico médico e aconselhamento financeiro são exemplos de problemas abordados por SEs (LUCAS; GAAG, 1991).

Com a evolução tecnológica no campo dos SEs, diversas técnicas foram incorporadas a seus motores de inferência. Uma delas é o uso da lógica fuzzy, que estende o uso de regras para representar conhecimento ao associar uma probabilidade a cada regra.

Ontologias, modelos de dados que representam conceitos de um domínio e seus relacionamentos, também podem ser utilizadas em SEs.

Outro componente que também pode ser utilizado em SEs são sistemas de manutenção da verdade, que registram dependências na base de conhecimento do SE de tal forma que quando fatos são alterados, o conhecimento dependente também é alterado. Esse tipo de sistema aumenta a confiança de conclusões obtidas no processo de inferência, pois mantém consistência na base de conhecimento (JANKOWSKA, 2005).

Em sistemas tradicionais, a lógica de funcionamento de um programa é embutida em seu código, que tipicamente apenas quem possui conhecimento em programação consegue analisar. Em SEs, as regras que ditam o funcionamento do sistema costumam ser escritas em um formato intuitivo e de fácil compreensão, possibilitando que especialistas de um domínio possam analisá-las e até editá-las, dispensando a necessidade de um especialista em computação.

Essa abordagem traz inúmeros benefícios, principalmente agilidade no desenvolvimento de sistemas e facilidade na manutenção. Muitas vezes, o fluxo lógico de um programa que faz uso de SEs consiste apenas na ativação do motor de inferência. Com isso, protótipos de sistemas complexos podem ser desenvolvidos em pouquíssimo tempo.

Uma das principais desvantagens de SEs se encontra na necessidade de aquisição do conhecimento. O tempo de especialistas em domínios é de grande valor e estes costumam estar em constante demanda pelas organizações em que trabalham.

Para contornar esse problema, houve um grande esforço empregado em pesquisa no campo de SEs para o desenvolvimento de ferramentas que auxiliam esse processo.

2.3 SISTEMAS DE PRODUÇÃO

Uma maneira de implementar SEs é utilizar Sistemas de Produção (SPs), nos quais o conhecimento do especialista é representado através de regras, chamadas de produções, na forma "se uma determinada condição for verdadeira, então uma ação correspondente deve ser executada". Um SP é composto por uma base de regras, uma memória de trabalho e um interpretador (BARR; FEIGENBAUM, 1981).

A base de regras armazena as produções do sistema, que são constituídas de duas partes. A parte “se” das produções é chamada de condição, ou lado esquerdo, enquanto que a parte “então” é chamada de ação, ou lado direito da produção. Para uma produção ser acionada, sua condição deve estar presente na memória de trabalho.

O interpretador é o componente que realiza a correspondência entre as produções e a memória de trabalho, definindo quais produções devem ser acionadas. Para isso, um algoritmo de encadeamento progressivo costuma ser utilizado.

Uma maneira simples de realizar o casamento de padrões entre os fatos da base de conhecimento e as condições das produções consiste em percorrer a lista de produções ativando aquelas cuja condição se encontra presente na base de conhecimento.

Essa abordagem é simples porém demonstra ser muito lenta para uma quantidade razoável de fatos e regras. Os principais SPs implementam o casamento de padrões entre fatos e regras através do algoritmo Rete, que constrói uma rede de nodos representando condições relacionadas, sacrificando memória para aumentar a velocidade de processamento.

Outra questão acerca de SPs consiste na seleção final de quais produções retornadas pelo algoritmo de casamento de padrões entre fatos e regras devem ser acionadas.

Esse processo é chamado de estratégia de resolução de conflitos, e pode utilizar diferentes critérios, como a ordem em que as regras foram escritas ou prioridades previamente estabelecidas.

Ainda segundo (BARR; FEIGENBAUM, 1981), uma das principais qualidades de SPs é a modularidade, tendo em vista que as produções da base de regras podem ser adicionadas, removidas ou alteradas sem afetarem outras produções diretamente, comportando-se como parcelas independentes de conhecimento. Outra vantagem é a naturalidade na expressão de certos tipos de conhecimento, pois afirmações do tipo “tal ação deve ser realizada quando uma condição for verdadeira” são naturalmente traduzidas para produções.

2.4 TESTE DE COX-STUART

O método proposto por (Cox & Stuart, 1995) é utilizado para testar tendência monótonas, isto é, tendências crescentes ou decrescentes, em sequências de observações. Esse teste requer que o número de observações seja par, por isso, remove-se a observação central quando o número é ímpar.

O método consiste num teste de hipóteses, onde a hipótese nula é de que não há tendência e a hipótese alternativa é de que há tendência. Dada uma sequencia X1, X2, …, Xm, Xm+1, … Xn, sendo m = n/2, deve-se calcular as diferenças Xm+1 - X1, Xm+2 - X2, …, Xn - Xm. Conta-se então a quantidade de diferenças positivas e a quantidade de diferenças negativas. Espera-se uma tendência positiva quando a quantidade de diferenças positivas for superior e negativa caso contrário.

O p-valor do teste é obtido calculando a probabilidade de ocorrência da quantidade de diferenças positivas, caso a tendência esperada seja positiva, ou negativas, caso a tendência esperada seja negativa, encontrados nas observações, através da distribuição Binomial com parâmetros p = 0.5 e n = m.

2.5 CLIPS

CLIPS, acrônimo em inglês para “C Language Integrated Production System”, é uma ferramenta de domínio público para desenvolvimento de sistemas especialistas baseados em regras. Foi lançada em 1986 e desde então vem sendo aprimorada.

A ferramenta opera mantendo uma lista de fatos e um conjunto de regras que trabalha sobre os fatos. As regras consistem em condições, que normalmente envolvem fatos, e ações, que podem inserir ou remover fatos da memória de trabalho. A figura 1 mostra a definição de dois fatos e uma regra através de sua linguagem. O motor de inferência da ferramenta trabalha buscando a correspondência entre fatos da memória de trabalho e condições das regras do sistema.

```
(deftemplate car_problem
    (slot name)
    (slot status)
)
(deffacts trouble_shooting
    (car_problem (name ignition_key) (status on))
    (car_problem (name engine) (status wont_start))
    (car_problem (name headlights) (status work))
 )
(defrule rule1
    (car_problem (name ignition_key) (status on))
    (car_problem (name engine) (status wont_start))
     =>
    (assert (car_problem (name starter) (status faulty)))
 )
```

Figura 1 - Definições na linguagem da ferramenta CLIPS

Embora seja voltada para a construção de sistemas baseados em regras, também há possibilidade de programação procedural na ferramenta através de funções e uso de programação orientada à objetos. Além disso, regras no sistema podem ter objetos em suas condições, possibilitando inúmeras formas de modelagem.

Uma das vantagens do CLIPS é sua portabilidade. Seu código é escrito na linguagem C e pode ser executado em diferentes sistemas operacionais como Windows, macOS e Linux. Outro benefício de seu uso é sua facilidade de integração com códigos escritos em outras linguagens.

Com a ferramenta, os pesquisadores (NASER, ZAITER, 2008) construíram um sistema especialista para diagnóstico de doenças oculares e obtiveram retorno positivo de médicos e pacientes.

(SHARMA, 2013) projetou e desenvolveu um sistema de aconselhamento para estudantes através do CLIPS, com objetivo de auxiliar a escolha do ramo de engenharia, para alunos que se interessam em estudar a área. O sistema age através de perguntas específicas para cada ramo e responde indicando um fator de certeza, que indica a probabilidade de sucesso que o estudante teria na área. O autor concluiu que o sistema poderia ser expandido para outros cursos, para ajudar estudantes a escolherem suas carreiras.

2.6 USO DE REGRAS

Muitas vezes a tomada de decisão de um especialista tem como base principalmente sua experiência na área de atuação. A maneira mais intuitiva para representar esse conhecimento empírico é através do uso de regras, pois expressa qual decisão deve ser tomada a partir de um determinado cenário.

Os domínios em que o uso de regras demonstra ser a melhor abordagem para representação do conhecimento são inúmeros. Na escavação de poços do pré-sal, tema deste trabalho, os engenheiros fazem uso de modelos matemáticos e computacionais, mas também tomam decisões utilizando regras.

Na medicina, o uso de regras é ainda mais evidente. Médicos seguem protocolos utilizando sintomas e resultados de exames como principais condições para realizar o diagnóstico de doenças em pacientes e fornecer o tratamento adequado.

Muitos guias e treinamentos paras jogos como poker ou xadrez são escritos através de regras. Esse é outro exemplo de domínio em que regras são utilizadas para representar o conhecimento.

3 PROPOSTA

Neste capítulo o desenvolvimento do módulo de regras é abordado. Inicialmente, a programação da engine e do editor de regras é detalhada. Por fim, a integração e validação do módulo com o SO-BR são apresentadas.

3.1 ENGINE

A engine é o componente responsável pelo mecanismo de verificação e ativação das regras. Ela deve receber as sequências de valores para cada variável da perfuração, determinar suas tendências, e realizar um cruzamento entre as regras, informando quais foram ativadas. No código do módulo, a engine é representada pela classe *BaseDeRegras*, que faz uso da ferramenta CLIPS.

A figura 2 mostra um exemplo de regra no SO-BR, utilizando como condição o cenário em que as variáveis PSB e ROP possuem tendências crescente e decrescente, respectivamente, e possui como ação a indicação de um alerta de vibração, sugerindo um aumento de 20% na variável RPM.

```
(defrule Regra
        (or (and (variavel (nome PSB) (tendencia CRESCIMENTO))
                 (variavel (nome ROP) (tendencia DECRESCIMENTO))
            )
        )
=> (assert (alerta (tipo VIBRACAO) (variaveis RPM ) (porcentagens 20 ) (texto "regra de teste"))))
```

Figura 2 - Exemplo de regra no SO-BR

As regras são carregadas através do método *carregarRegras* da classe, que possui como parâmetro um *string* contendo as regras em formato CLIPS a serem utilizadas pela engine.

A ativação da engine ocorre através do método *ativar*, que é invocado pelo módulo de controle do SO-BR sempre que a ROP encontra-se fora do intervalo planejado, a fim de verificar se a situação em que a perfuração se encontra é coberta pela base de regras.

O método possui como parâmetro uma matriz de dados contendo as últimas observações de cada variável da perfuração. A partir destas, a tendência de cada variável é definida através do teste de Cox-Stuart, cujo algoritmo foi implementado na classe *UtilidadesRegras*, que possui diversas funções utilizadas pelo módulo.

As tendências definidas pelo teste são inseridas como fatos na memória de trabalho da ferramenta CLIPS. A partir daí, o motor de inferência da ferramenta é

executado, disparando as regras que tenham condições correspondentes com os fatos inseridos.

A ação de uma regra disparada é a inserção de um novo fato na memória da ferramenta, indicando a presença de um alerta no sistema. A ferramenta CLIPS permite que fatos possuam atributos. Os fatos que representam alertas possuem como atributos o seu tipo, sugestões de alterações em porcentagem nos parâmetros operacionais da perfuração e um texto.

Depois da execução do motor de inferência, a lista com todos os fatos presentes na ferramenta é percorrida, buscando aqueles que representam alertas. Quando um alerta é encontrado, o método verifica cada porcentagem sugerida para alteração de parâmetros operacionais e as aplica no intervalo de confiança do valor do parâmetro a ser alterado, calculado a partir das observações passadas como parâmetro para o método.

Para cada alerta ativado, um objeto da classe *ConfiguracaoAlerta* é criado, que possui como atributos um mapeamento com os novos valores de parâmetros operacionais, sugeridos pelo alerta, e o texto definido na criação do alerta para ser apresentado para o usuário.

O método por fim retorna um mapeamento com os tipos de alertas ativados e cada objeto de configuração correspondente, que é utilizado pelo módulo de controle do SO-BR para exibir a janela de alertas na interface gráfica de usuário.

Para validar o funcionamento da engine isoladamente, dois cenários de testes foram programados. Em ambos foram inseridas duas regras na engine, a primeira com a condição da variável RPM com tendência de crescimento, indicando um alerta de vibração, e a segunda com a RPM com tendência de decrescimento, indicando um alerta de desgaste. Também é passado como parâmetro para o método *ativar* uma matriz contendo 10 observações para cada variável de um conjunto de 5 variáveis. A diferença entre os cenários está nas observações da matriz.

A figura 3 mostra o código do primeiro cenário de teste, onde a coluna da matriz referente à variável RPM possui observações seguindo uma tendência crescente. A ferramenta de depuração do ambiente de desenvolvimento Visual Studio é utilizada para analisar o valor do resultado retornado pela engine. Como esperado, a primeira regra é ativada, indicando um alerta de vibração.

A figura 4 mostra o código do segundo cenário de teste, com a ordem das observações para a variável RPM invertidas, possuindo assim tendência decrescente. Novamente, o resultado esperado é obtido, com a ativação dessa vez da segunda regra, indicando um alerta de desgaste.

```
int main( int argc, char** argv )
{
    CLIPS::init();
    BaseDeRegras b;
    QString c1 = " (deftemplate alerta (slot tipo) (multislot variaveis) (multislot porcentagens) (slot texto))";
    QString c2 = "(deftemplate variavel (slot nome) (slot tendencia))";
    QString regra = " (defrule regra1 (or (and (variavel (nome RPM) (tendencia CRESCIMENTO)) ) )"
        "=> (assert (alerta (tipo VIBRACAO) (variaveis ) (porcentagens ) (texto \"\"))))";
    QString regra2 = " (defrule regra2 (or (and (variavel (nome RPM) (tendencia DECRESCIMENTO)) ) )"
        "=> (assert (alerta (tipo DESGASTE) (variaveis ) (porcentagens ) (texto \"\"))))";
    b.carregarRegras(c1 + " " + c2 + " " + regra + " " + regra2);
    std::vector<Variavel> v;
    v.push_back(Variavel::_rop);
    v.push_back(Variavel::_psb);
    v.push_back(Variavel::_rpm);
    v.push_back(Variavel::_hsi);
    v.push_back(Variavel::_ucs);
    dlib::matrix<double> M(10, 5);
    M = 1, 1, 10, 20, 1,
        1, 2, 11, 18, 2,
        1, 3, 12, 16, 3,
        1, 4, 13, 18, 2,
        1, 5, 14, 14, 5,
        1, 6, 15, 10, 6,
        1, 7, 16, 8, 7,
        1, 8, 17, 10, 6,
        1, 9, 18, 6, 8,
        1, 10, 19, 5, 9;
    BaseDeRegras::Resultado resultado = b.ativar(DadosV2(v, M));
    return 0;
}
```

```
resultado { size=1 }
[0]    (vibracao (0), {valoresVariaveis={ size=0 } texto= })
```

Figura 3 - Código do primeiro cenário de teste da engine

```
int main( int argc, char** argv )
{
    CLIPS::init();
    BaseDeRegras b;
    QString c1 = " (deftemplate alerta (slot tipo) (multislot variaveis) (multislot porcentagens) (slot texto))";
    QString c2 = "(deftemplate variavel (slot nome) (slot tendencia))";
    QString regra = " (defrule regra1 (or (and (variavel (nome RPM) (tendencia CRESCIMENTO)) ) )"
        "=> (assert (alerta (tipo VIBRACAO) (variaveis ) (porcentagens ) (texto \"\"))))";
    QString regra2 = " (defrule regra2 (or (and (variavel (nome RPM) (tendencia DECRESCIMENTO)) ) )"
        "=> (assert (alerta (tipo DESGASTE) (variaveis ) (porcentagens ) (texto \"\"))))";
    b.carregarRegras(c1 + " " + c2 + " " + regra + " " + regra2);
    std::vector<Variavel> v;
    v.push_back(Variavel::_rop);
    v.push_back(Variavel::_psb);
    v.push_back(Variavel::_rpm);
    v.push_back(Variavel::_hsi);
    v.push_back(Variavel::_ucs);
    dlib::matrix<double> M(10, 5);
    M = 1, 1, 19, 20, 1,
        1, 2, 18, 18, 2,
        1, 3, 17, 16, 3,
        1, 4, 16, 18, 2,
        1, 5, 15, 14, 5,
        1, 6, 14, 10, 6,
        1, 7, 13, 8, 7,
        1, 8, 12, 10, 6,
        1, 9, 11, 6, 8,
        1, 10, 10, 5, 9;
    BaseDeRegras::Resultado resultado = b.ativar(DadosV2(v, M));
    return 0;
}
```

```
resultado { size=1 }
[0]    (desgaste (2), {valoresVariaveis={ size=0 } texto= })
```

Figura 4 - Código do segundo cenário de teste da ativação da engine

Tendo em vista que para manter a consistência é necessário programar um algoritmo de verificação de exclusividade mútua entre regras, sendo duas regras

mutuamente exclusivas quando não há um cenário que ative-as simultaneamente, também foi implementado na classe *BaseDeRegras* o método *testaExclusividadeMutua*, que é utilizado pelo editor de regras para verificar se duas regras são mutuamente exclusivas.

O método recebe como parâmetro a lista de variáveis utilizadas nas regras e uma regra em formato CLIPS, que deve ter como condição a intersecção das condições das duas regras com que se deseja testar a exclusividade. A criação dessa regra composta é realizada pelo método *construirRegraClipsComposta* da classe *UtilidadesRegras*.

Para testar a exclusividade, uma instância da ferramenta CLIPS é criada localmente no método e a regra recebida como parâmetro é inserida em sua memória de trabalho. A partir desse ponto, o método executa o motor de inferência da ferramenta diversas vezes, fornecendo uma combinação diferente de tendências para as variáveis em cada execução, até que todas as combinações sejam testadas.

Como a regra composta possui como condição a interseção das condições das regras a serem testadas, se em uma das combinações testadas a regra composta for ativada, existe um cenário de perfuração que ativaria as duas regras e portanto elas não são exclusivas. Caso nenhuma das combinações de tendências ative a regra composta, as duas regras são exclusivas.

3.2 EDITOR GRÁFICO DE REGRAS

O editor gráfico de regras é uma ferramenta de interface gráfica, que possibilita que engenheiros criem regras sem conhecerem a linguagem da ferramenta CLIPS, utilizada pela engine. Ele deve traduzir as regras definidas a partir de informações inseridas em sua interface para o formato adequado à engine. No código do módulo, o editor gráfico de regras é representado pela classe *JanelaConjuntosDeRegras*.

Para o desenvolvimento do editor, componentes gráficos representando regras e conjuntos de regras foram programados através das classes *RegraWidget* e *ConjuntoDeRegrasWidget* respectivamente.

A figura 5 mostra a aparência do editor. Na coluna esquerda, são exibidos os conjuntos de regras da base de regras do SO-BR. Na coluna direita, são exibidas as regras do conjunto selecionado.

Nesse exemplo, há dois conjuntos de regras na base, sendo que o primeiro está selecionado e tem suas duas regras sendo exibidas. A fim de prover uma melhor usabilidade ao usuário, os componentes gráficos de regras possuem a borda pintada com cor de acordo com o tipo da regra, sendo verde para vibração, azul para enceramento e vermelho para desgaste. Na coluna esquerda do editor, o conjunto

selecionado para visualização e edição de regras é destacado com seu fundo pintado de cinza.

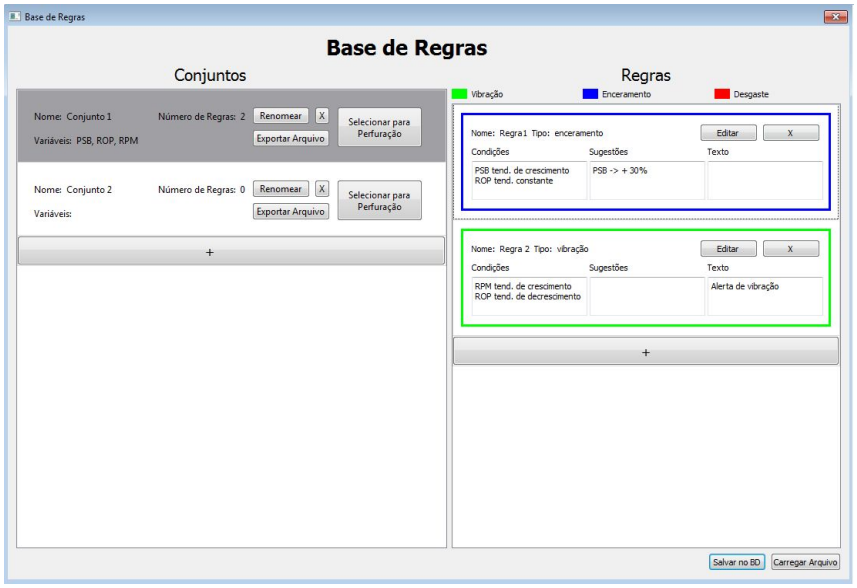

Figura 5 - Editor gráfico de regras

A figura 6 mostra a janela de criação de regra, programada através da classe *JanelaCriacaoRegra*, que é exibida quando o usuário clica no botão “+” na coluna de regras. Essa janela também é exibida quando o usuário clica no botão “Editar” do componente gráfico de uma regra, tendo seus campos pré-configurados com o nome, condições, sugestões e texto da regra.

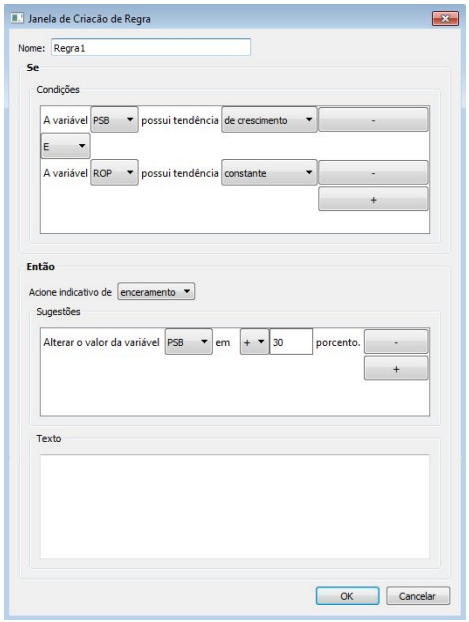

Figura 6 - Janela de criação de regra

Quando o usuário clica no botão “OK” para concluir a criação ou edição de uma regra, diversas validações são realizadas. Se o usuário esquecer de preencher algum campo, um aviso é exibido. Também é verificado se o nome escolhido para a regra não está sendo utilizado por outra regra do conjunto.

Outra validação importante que é realizada nesta etapa trata da consistência entre as condições da regra, impedindo que o usuário crie uma regra impossível de ser ativada. Se existem condições com mesma variável e tendências diferentes unidas pelo conector E, um aviso é exibido. A figura 7 exemplifica este caso.

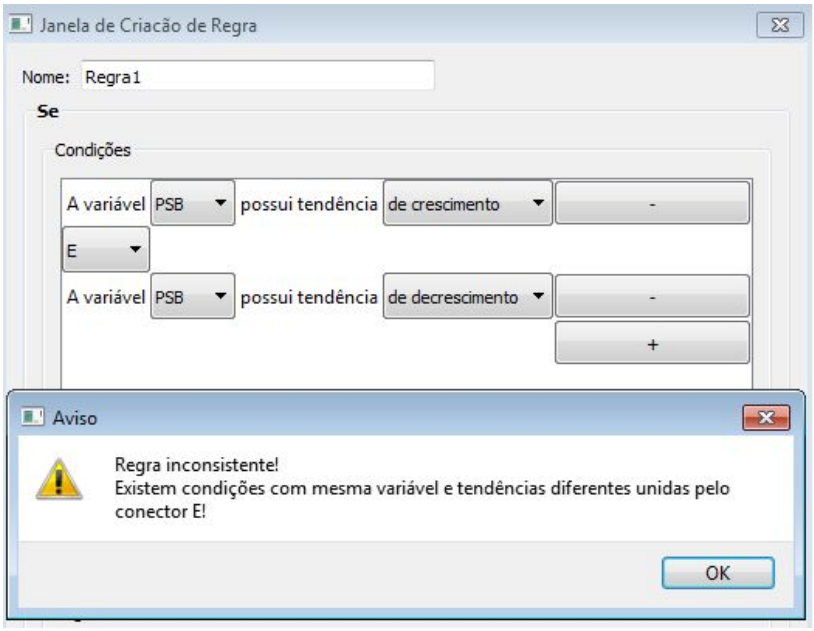

Figura 7 - Aviso de inconsistência entre condições de uma regra

A última validação realizada testa se a regra possui exclusividade mútua com cada uma das regras do conjunto de mesmo tipo, isto é, se não há um cenário que ative-as simultaneamente. A exclusividade mútua precisa ser garantida pois não faria sentido exibir simultaneamente mais de um alerta de mesmo tipo ao usuário, possivelmente com sugestões de alterações em parâmetros operacionais diferentes.

A primeira regra da Figura 5 é do tipo enceramento e possui como condição para ser ativada um cenário de perfuração onde a variável PSB tem tendência de crescimento e a variável ROP tem tendência constante. Ao tentar inserir uma nova regra de enceramento no mesmo conjunto, possuindo como condição um cenário não exclusivo com o da primeira regra, um aviso é exibido. A figura 8 exemplifica este caso, pois existe pelo menos um cenário que satisfaz ambas as condições da regra a ser inserida e da primeira regra do conjunto.

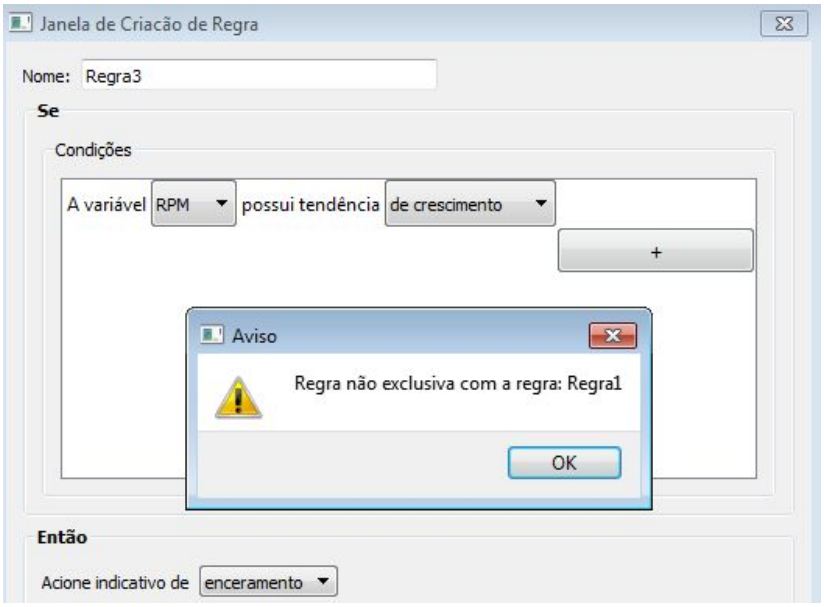

Figura 8 - Aviso da violação da exclusividade mútua entre duas regras

Com o editor, os usuários do SO-BR podem ter diferentes conjuntos de regras salvos na base de regras, selecionando em tempo real o mais adequado para o estágio em andamento da perfuração. Essa seleção é realizada através do botão “Selecionar para Perfuração”. Uma borda preta é pintada ao redor do conjunto selecionado, como exibe a figura 9.

Figura 9 - Borda preta indicando o conjunto selecionado para perfuração

Duas opções de persistência para a base de regras são providas para o usuário. Através do botão “Exportar Arquivo” do componente gráfico de um conjunto de regras, o usuário pode exportar um conjunto de regras da base para um arquivo, que pode posteriormente ser carregado para uma base através do botão “Carregar Arquivo”.

Essa opção possibilita que um usuário possa criar conjuntos de regras em um computador, exportá-los para uma mídia de armazenamento e carregá-los para outro computador.

A segunda opção para persistência pode ser acessada através do botão “Salvar no BD”, que armazena toda a base de regras no banco de dados do SO-BR. Sempre que o editor gráfico de regras é aberto, todos os conjuntos de regras salvos no banco de dados são carregados.

A principal função do editor gráfico de regras é possibilitar que os usuários do SO-BR gerenciem as regras utilizadas pela engine do módulo de regras sem conhecerem os detalhes de sua escrita em linguagem não natural. Para isso, foi definida uma classe *Regra* e foram programados métodos na classe *UtilidadesRegras*, que realizam a conversão entre objetos dessa classe e regras no formato CLIPS.

Para validar o funcionamento desses métodos, foi programado um teste cujo código é exibido pela figura 10.

```
int main(int argc, char *argv[])
{
    Regra regra1;
    regra1.nome = "regra1";
    Regra::Condicao c1(Variavel::_psb, Tendencia::crescimento);
    Regra::Condicao c2(Variavel::_rop, Tendencia::constante);
    regra1.condicoes = { c1, c2 };
    regra1.conectores = { Conector::e } ;
    regra1.tipo = TipoRegra::vibracao;
    regra1.sugestoes = { Regra::Sugestao(Variavel::_psb, 0.3) };
    regra1.texto = "texto1";

    Regra regra2;
    regra2.nome = "regra2";
    Regra::Condicao c3(Variavel::_rpm, Tendencia::crescimento);
    Regra::Condicao c4(Variavel::_rop, Tendencia::decrescimento);
    regra2.condicoes = { c3, c4 } ;
    regra2.conectores = { Conector::e };
    regra2.tipo = TipoRegra::enceramento;
    regra2.sugestoes = { Regra::Sugestao(Variavel::_rpm, 0.2) };
    regra2.texto = "texto2";

    QString baseDeRegrasClips = UtilidadesRegras::construirBaseDeRegrasClips({ regra1, regra2 });
    std::vector<Regra> regras = UtilidadesRegras::lerBaseDeRegrasClips(baseDeRegrasClips);
    return 0;
}
```

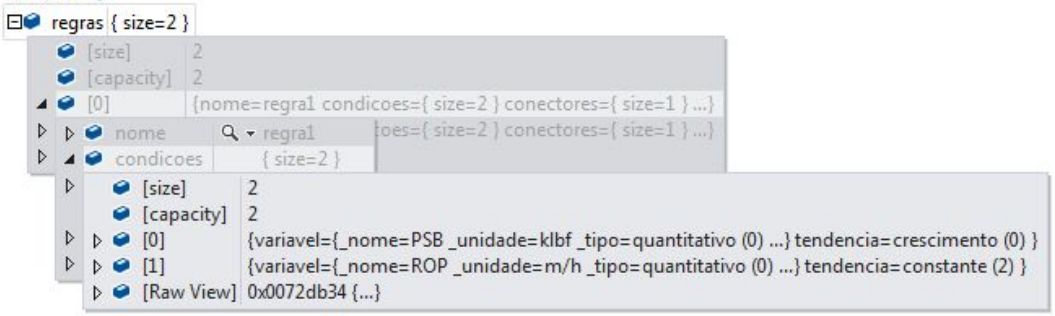

Figura 10 - Código de teste da conversão entre objetos de regras e o formato CLIPS

Nesse teste, dois objetos da classe *Regra* são criados e passados em um conjunto para o método *construirBaseDeRegrasClips* da classe *UtilidadesRegras*. Esse método realiza a conversão do conjunto de regras para um string no formato CLIPS que contém as regras do conjunto, apropriado para ser utilizado na engine. O string retornado é então passado para o método *lerBaseDeRegrasClips*, também da classe *UtilidadesRegras*, que faz a conversão inversa retornando uma lista de objetos da classe *Regra*.

A ferramenta de depuração do ambiente de desenvolvimento Visual Studio é utilizada para verificar que a lista possui as mesmas regras que foram utilizadas para a construção da base em formato CLIPS.

3.3 VALIDAÇÃO DO MÓDULO DE REGRAS INTEGRADO AO SO-BR

Após ambos os componentes, engine e editor de regras, terem sido validados isoladamente, prosseguiu-se para a integração e validação do módulo de regras no SO-BR, simulando o acompanhamento de uma perfuração real.

Para isso, modificou-se uma planilha de dados reais de uma perfuração, fornecida pela Petrobras (Cenpes), para que em um determinado ponto da simulação as variáveis PSB e Vazão apresentassem tendências crescente e constante respectivamente.

Então, com o SO-BR em execução, o editor de regras foi acessado pela interface do programa e uma regra contendo as condições esperadas para sua ativação (de acordo com as modificações no arquivo de dados) foi criada, indicando um alerta de vibração e sugerindo um aumento de 20% na variável RPM. A figura 11 exibe a janela do editor com a regra inserida em um conjunto selecionado para ser utilizado durante a perfuração.

A simulação foi iniciada e foram adicionados componentes gráficos na interface do SO-BR para o acompanhamento das observações e tendência das variáveis PSB e Vazão. Tendo passado um certo tempo de simulação, o programa exibiu o alerta esperado. A figura 12 mostra a exibição do alerta de vibração com a sugestão para a variável RPM. Pode-se verificar na figura que a sugestão está condizente com o acréscimo de 20% no valor mostrado pelo seu velocímetro. Pelos componentes gráficos com as últimas observações das variáveis PSB e Vazão, pode-se verificar que são exibidas as tendências esperadas.

O teste realizado valida o funcionamento do módulo de regras integrado ao SO-BR.

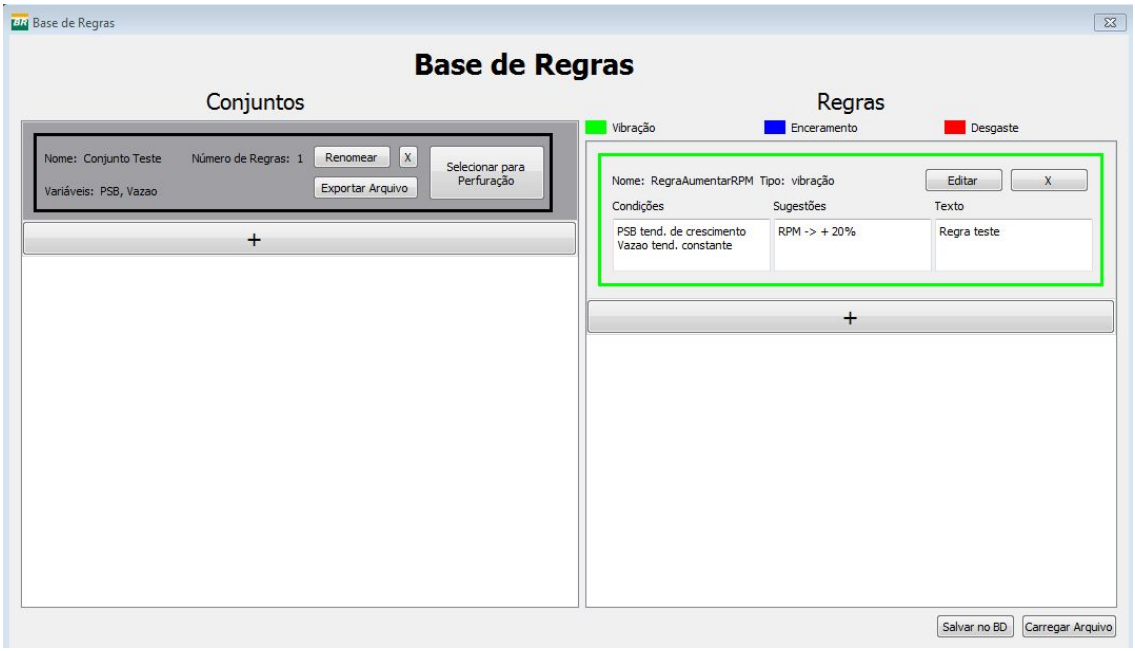

Figura 11 - Editor com a regra criada para a validação

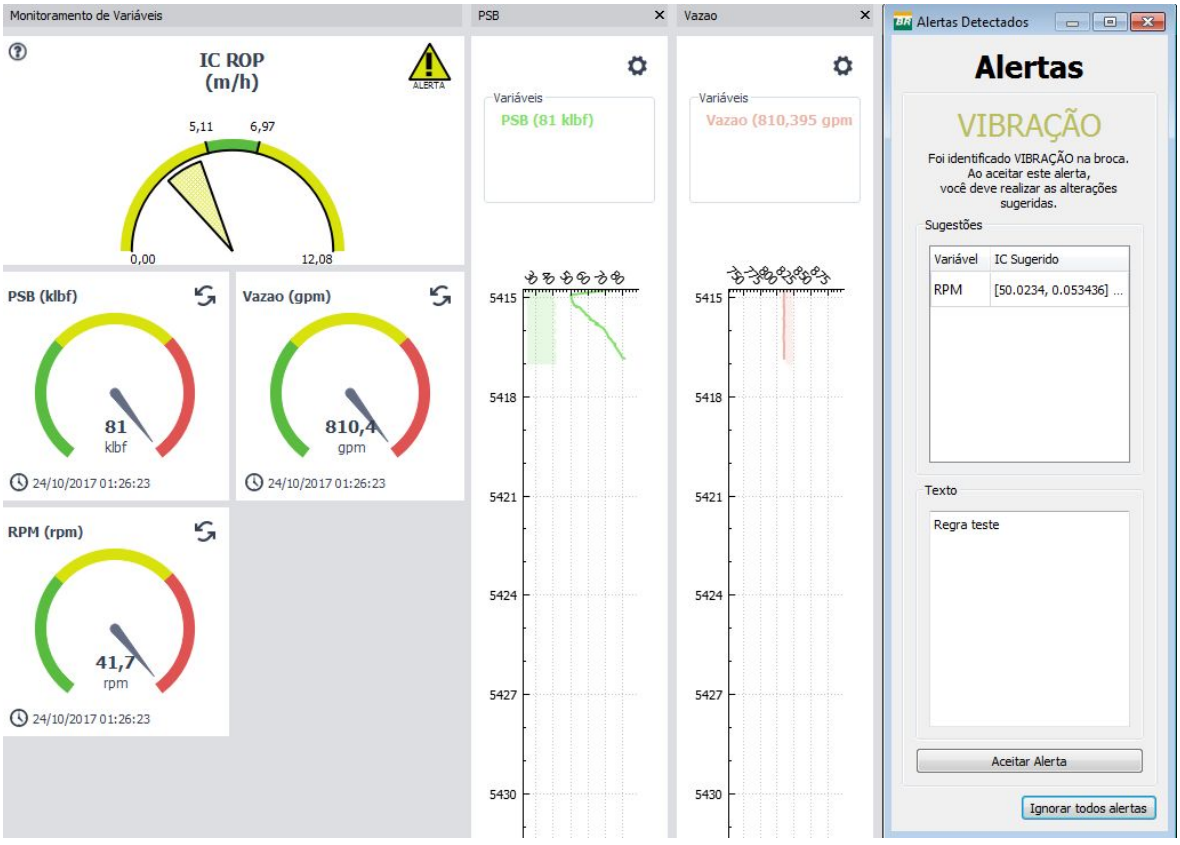

Figura 12 - Alerta da base de regras sendo exibido durante simulação no SO-BR

4 CONCLUSÃO

Este trabalho teve como objetivo o desenvolvimento de um módulo de regras integrado ao software SO-BR, um sistema de acompanhamento e apoio à tomada de decisão em perfurações de poços do pré-sal. Em alguns cenários de perfuração, a abordagem utilizada pelo software, que combina as técnicas de IA Neuro-Fuzzy e Redes Bayesianas, demonstra não ser adequada. Para melhorar a capacidade do software e contornar esses cenários, um módulo de regras foi projetado e desenvolvido.

O módulo foi dividido em dois componentes, uma engine, que provê o mecanismo de verificação das regras, e um editor gráfico, que permite a criação delas. Para a programação da engine, a ferramenta CLIPS foi utilizada.

Após ambos os componentes terem sido validados isoladamente, o módulo foi integrado ao SO-BR e validado em conjunto com seu módulo de controle, simulando uma perfuração real e verificando a ativação das regras definidas.

A ênfase deste trabalho foi na área de petróleo, na perfuração de poços do pré sal, porém a abordagem utilizada é aplicável a diversas outras áreas.

Como trabalhos futuros, algumas funcionalidades poderiam ser adicionadas ao módulo, como a criação de outros tipos de alertas e regras com condições e sugestões mais complexas.

REFERÊNCIAS

LUGER, George F.. **Inteligência artificial.** 6. ed. São Paulo: Pearson Education do Brasil, 2013.

REZENDE, Solange Oliveira. **Sistemas inteligentes: fundamentos e aplicações.** Barueri: Manole, 2003.

LUCAS, Peter J.f.; GAAG, Linda C. van Der. **Principles of Expert Systems.** Amsterdam: Addison-wesley, 1991.

BARR, Avron; FEIGENBAUM, Edward A.. **The handbook of artificial intelligence.** Los Altos: William Kaufmann, Inc., 1981.

COX, D. R.; STUART, A. **Some quick tests for trend in location and dispersion.** Biometrika, London, v. 42, p.80-95, 1955.

NASER; Samy S. ZAITER, Abu. **An Expert System For Diagnosing Eye Diseases Using CLIPS**, Faculty of Engineering & Information Technology, Al-Azhar University, Gaza, Palestine, Journal of Theoretical and Applied Information Technology, 2005-2008 JATIT

SHARMA, Tilotma. **A Rule-Based Expert System based on Certainty Factor and Backward Chaining Approach (student counseling system)** vol. 2, Issue 1, January 2013, ISSN 2319 - 4847, IJAIEM

SCHREIBER, Guus; WIELINGA, Bob; BREUKER, Joost. **KADS: A Principled Approach to Knowledge-based System Development**. vol. 11. Amsterdam: Academic Press, 1993

SYDOW, Achim. **Environmental Systems**. Vo. III. Oxford: Eolss Publishers, 2010.

JANKOWSKA, Beata. **Truth Maintenance in an Expert System with Uncertainty.** Poznan: Schedae Informaticae, 2005.

APÊNDICE B - Artigo

Desenvolvimento de um Módulo de Regras no SO-BR, Sistema de Acompanhamento e Apoio à Tomada de Decisão em Perfurações de Poços do Pré-sal

Arthur Hortmann Erpen

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil

arthurerpen@gmail.com

Abstract. ***SO-BR software is a monitoring and decision support system for pre-salt well drilling. In some drilling scenarios, the software approach, which combines artificial intelligence techniques Neuro-Fuzzy and Bayesian Networks, proves to be inadequate. This work presents the programming of a rules module that seeks to enable engineers to insert in SO-BR part of their knowledge through rules in order to improve the software's performance. The CLIPS tool was used to program the module engine, providing the mechanism for matching rules. A graphical rule editor has been programmed so that engineers do not need to master the unnatural language of the rules used by the engine. The emphasis of this work is in the area of oil but the approach used is applicable in other areas.***

Resumo. ***O software SO-BR é um sistema de acompanhamento e apoio à tomada de decisão em perfurações de poços do pré-sal. Em alguns cenários de perfuração, a abordagem utilizada pelo software, que combina as técnicas de IA Neuro-Fuzzy e Redes Bayesianas, demonstra não ser adequada. Este trabalho apresenta a programação de um módulo de regras que busca possibilitar que engenheiros insiram no SO-BR parte de seu conhecimento através de regras, a fim de aperfeiçoar a capacidade do software. A ferramenta CLIPS foi utilizada para a programação da engine do módulo, provendo o mecanismo de verificação das regras. Um editor gráfico de regras foi programado para que os engenheiros não precisem dominar a linguagem não-natural das regras utilizada pela engine. A ênfase deste trabalho é na área de petróleo porém a abordagem utilizada é aplicável a diversas outras áreas.***

1. Introdução

Enquanto que alguns dos problemas computacionais tratam apenas da manipulação de dados, como a ordenação de uma sequência de números, outros problemas envolvem o uso do raciocínio e pensamento humano, como jogar xadrez ou entender uma linguagem natural. A Inteligência Artificial é o ramo da Ciência da Computação que busca a automação do comportamento inteligente (LUGER, 2013).

Este trabalho tem como objetivo o desenvolvimento de um módulo de regras integrado ao software SO-BR, um sistema de acompanhamento e apoio à tomada de decisão em perfurações de poços do pré-sal, o qual é resultado de um projeto de pesquisa entre a UFSC

(PerformanceLab/INE) e a Petrobras (Cenpes) que teve início há cerca de cinco anos. Em seu desenvolvimento verificou-se que seria de grande valia se os engenheiros que utilizam o programa pudessem configurar alertas a serem disparados em determinadas condições. Por exemplo, sendo registrada uma tendência constante para a taxa de perfuração, simultânea a uma tendência crescente para o peso sobre a broca, um alerta de desgaste de broca poderia ser emitido, incluindo opcionalmente sugestões de alterações em variáveis operacionais da perfuração.

Para a programação do módulo, uma pesquisa foi realizada em busca da ferramenta que melhor se adequasse ao SO-BR. Por prover um ambiente completo para a construção de sistemas baseados em regras, possuir código fonte aberto e ser escrita em linguagem compatível ao ambiente de desenvolvimento do programa, a ferramenta CLIPS foi escolhida.

Como as regras interpretadas pela ferramenta são escritas em sua própria linguagem não natural, também é objetivo do trabalho a programação de um editor gráfico, para facilitar a criação e edição das regras, de modo que os engenheiros não precisem estudar a ferramenta CLIPS para utilizar o sistema.

Embora a ênfase do trabalho seja na área de petróleo, é importante salientar que as técnicas empregadas podem ser utilizadas em diversas outras instâncias.

1.1 Justificativa

O SO-BR faz uso de sistemas Neuro-Fuzzy e Redes Bayesianas para apoiar a tomada de decisão durante a perfuração de poços do pré-sal. Quando a taxa de perfuração se encontra distante do esperado, o programa indica alterações a serem realizadas em parâmetros operacionais, como o peso sobre a broca.

Em algumas situações, essa abordagem demonstra não ser adequada. Em um caso de desgaste na broca, por exemplo, pode ser mais interessante iniciar o processo de sua substituição ao invés de continuar a perfuração com alterações nos parâmetros.

Para cobrir essas situações, é necessário inserir o conhecimento técnico do engenheiro no programa, de modo que alertas possam ser exibidos quando determinadas condições, envolvendo a tendência de variáveis da perfuração, são verdadeiras. O uso de regras demonstra ser uma ótima alternativa para a solução desse problema.

Um editor gráfico para a criação das regras demonstra ser essencial. Sem o mesmo, os engenheiros que utilizam o programa precisariam estudar a fundo a sintaxe da linguagem da ferramenta, não natural e de difícil compreensão.

2. Objetivos

O objetivo deste trabalho é desenvolver um módulo de regras para o programa SO-BR, composto por uma engine e um editor gráfico de regras. A engine é responsável pelo mecanismo de verificação e ativação das regras. Ela deve receber as sequências de valores para cada variável da perfuração, determinar suas tendências, e realizar um cruzamento entre as regras, informando quais foram ativadas.

O editor gráfico de regras é uma ferramenta de interface gráfica, que possibilita que engenheiros criem regras sem conhecerem a linguagem da ferramenta CLIPS, utilizada pela engine. Ele deve traduzir as regras definidas a partir de informações inseridas em sua interface para o formato adequado à engine.

3. Fundamentação

Dentre os temas abordados neste trabalho, três merecem destaque: sistemas de produção, teste de Cox-Stuart e CLIPS.

3.1. Sistemas de Produção

Nos Sistemas de Produção (SPs), o conhecimento especialista é representado através de regras, chamadas de produções, na forma "se uma determinada condição for verdadeira, então uma ação correspondente deve ser executada". Um SP é composto por uma base de regras, uma memória de trabalho e um interpretador (BARR; FEIGENBAUM, 1981).

A base de regras armazena as produções do sistema, que são constituídas de duas partes. A parte “se” das produções é chamada de condição, ou lado esquerdo, enquanto que a parte “então” é chamada de ação, ou lado direito da produção. Para uma produção ser acionada, sua condição deve estar presente na memória de trabalho.

O interpretador é o componente que realiza a correspondência entre as produções e a memória de trabalho, definindo quais produções devem ser acionadas. Para isso, um algoritmo de encadeamento progressivo costuma ser utilizado.

Ainda segundo (BARR; FEIGENBAUM, 1981), uma das principais qualidades de SPs é a modularidade, tendo em vista que as produções da base de regras podem ser adicionadas, removidas ou alteradas sem afetarem outras produções diretamente, comportando-se como parcelas independentes de conhecimento. Outra vantagem é a naturalidade na expressão de certos tipos de conhecimento, pois afirmações do tipo “tal ação deve ser realizada quando uma condição for verdadeira” são naturalmente traduzidas para produções.

3.2. Teste de Cox-Stuart

O método proposto por (Cox & Stuart, 1995) é utilizado para testar tendência monótonas, isto é, tendências crescentes ou decrescentes, em sequências de observações. Esse teste requer que o número de observações seja par, por isso, remove-se a observação central quando o número é ímpar.

O método consiste num teste de hipóteses, onde a hipótese nula é de que não há tendência e a hipótese alternativa é de que há tendência. Dada uma sequencia X1, X2, …, Xm, Xm+1, … Xn, sendo m = n/2, deve-se calcular as diferenças Xm+1 - X1, Xm+2 - X2, …, Xn - Xm. Conta-se então a quantidade de diferenças positivas e a quantidade de diferenças negativas. Espera-se uma tendência positiva quando a quantidade de diferenças positivas for superior e negativa caso contrário.

O p-valor do teste é obtido calculando a probabilidade de ocorrência da quantidade de diferenças positivas, caso a tendência esperada seja positiva, ou negativas, caso a tendência

esperada seja negativa, encontrados nas observações, através da distribuição Binomial com parâmetros p = 0.5 e n = m.

3.3. CLIPS

CLIPS, acrônimo em inglês para “C Language Integrated Production System”, é uma ferramenta de domínio público para desenvolvimento de sistemas especialistas baseados em regras. Foi lançada em 1986 e desde então vem sendo aprimorada.

A ferramenta opera mantendo uma lista de fatos e um conjunto de regras que trabalha sobre os fatos. As regras consistem em condições, que normalmente envolvem fatos, e ações, que podem inserir ou remover fatos da memória de trabalho. A figura 1 mostra a definição de dois fatos e uma regra através de sua linguagem. O motor de inferência da ferramenta trabalha buscando a correspondência entre fatos da memória de trabalho e condições das regras do sistema.

```
(deftemplate car_problem
    (slot name)
    (slot status)
)
(deffacts trouble_shooting
    (car_problem (name ignition_key) (status on))
    (car_problem (name engine) (status wont_start))
    (car_problem (name headlights) (status work))
 )
(defrule rule1
    (car_problem (name ignition_key) (status on))
    (car_problem (name engine) (status wont_start))
     =>
    (assert (car_problem (name starter) (status faulty)))
 )
```

Figura 1 - Definições na linguagem da ferramenta CLIPS

4. Proposta

Neste capítulo o desenvolvimento do módulo de regras é abordado. Inicialmente, a programação da engine e do editor de regras é detalhada. Por fim, a integração e validação do módulo com o SO-BR são apresentadas.

4.1. Engine

A engine é o componente responsável pelo mecanismo de verificação e ativação das regras. Ela deve receber as sequências de valores para cada variável da perfuração, determinar suas tendências, e realizar um cruzamento entre as regras, informando quais foram ativadas.

As regras são carregadas através de um método que possui como parâmetro um string contendo as regras em formato CLIPS a serem utilizadas pela engine.

A ativação da engine é invocada pelo módulo de controle do SO-BR sempre que a taxa de perfuração encontra-se fora do intervalo planejado, a fim de verificar se a situação em que a perfuração se encontra é coberta pela base de regras.

Uma matriz de dados contendo as últimas observações de cada variável da perfuração é utilizada como parâmetro para a ativação. A partir destas, a tendência de cada variável é definida através do teste de Cox-Stuart.

As tendências definidas pelo teste são inseridas como fatos na memória de trabalho da ferramenta CLIPS. A partir daí, o motor de inferência da ferramenta é executado, disparando as regras que tenham condições correspondentes com os fatos inseridos.

A engine retorna um mapeamento com os tipos de alertas ativados e outros detalhes necessários para que o módulo de controle do SO-BR exiba a janela de alertas na interface gráfica de usuário.

4.2. Editor gráfico de regras

O editor gráfico de regras é uma ferramenta de interface gráfica, que possibilita que engenheiros criem regras sem conhecerem a linguagem da ferramenta CLIPS, utilizada pela engine. Ele deve traduzir as regras definidas a partir de informações inseridas em sua interface para o formato adequado à engine.

A figura 2 mostra a aparência do editor. Na coluna esquerda, são exibidos os conjuntos de regras da base de regras do SO-BR. Na coluna direita, são exibidas as regras do conjunto selecionado.

Nesse exemplo, há dois conjuntos de regras na base, sendo que o primeiro está selecionado e tem suas duas regras sendo exibidas. A fim de prover uma melhor usabilidade ao usuário, os componentes gráficos de regras possuem a borda pintada com cor de acordo com o tipo da regra, sendo verde para vibração, azul para enceramento e vermelho para desgaste. Na coluna esquerda do editor, o conjunto selecionado para visualização e edição de regras é destacado com seu fundo pintado de cinza.

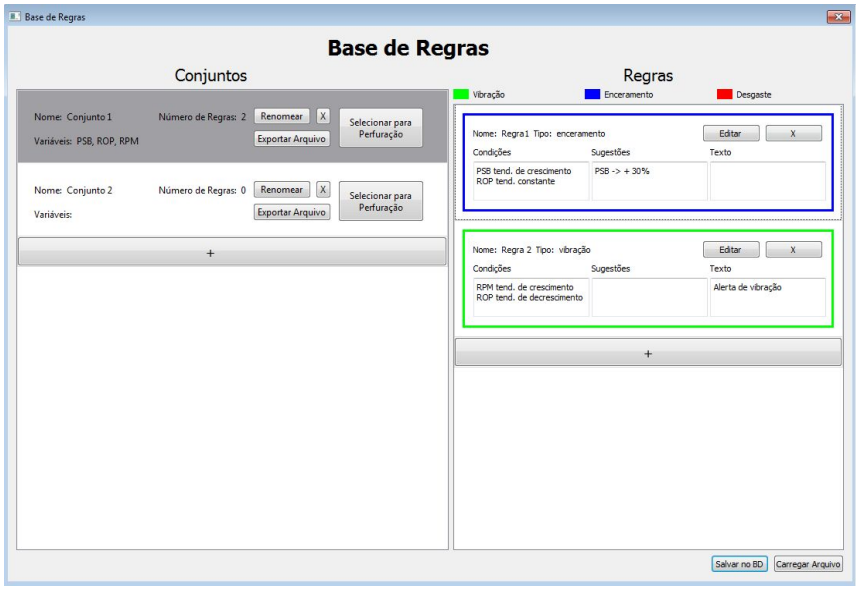

Figura 2 - Editor gráfico de regras

A figura 3 mostra a janela de criação de regra, que é exibida quando o usuário clica no botão “+” na coluna de regras. Essa janela também é exibida quando o usuário clica no botão “Editar” do componente gráfico de uma regra, tendo seus campos pré-configurados com o nome, condições, sugestões e texto da regra.

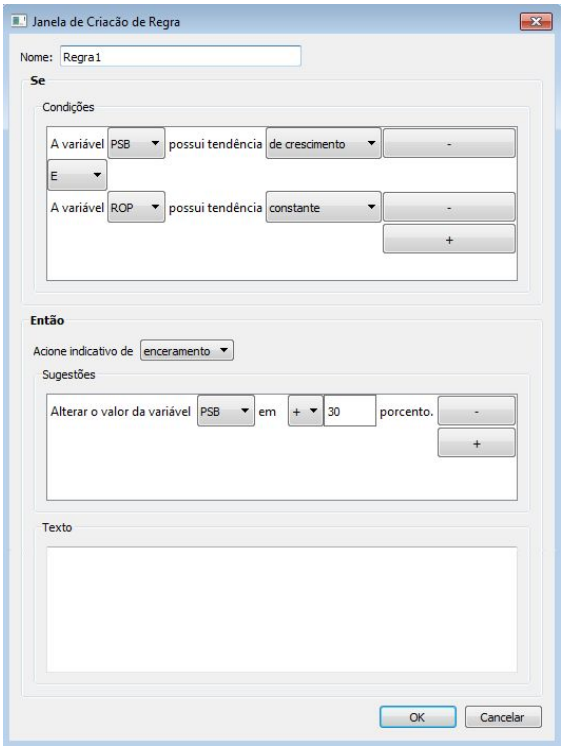

Figura 3 - Janela de criação de regra

Com o editor, os usuários do SO-BR podem ter diferentes conjuntos de regras salvos na base de regras, selecionando em tempo real o mais adequado para o estágio em andamento da perfuração. Essa seleção é realizada através do botão "Selecionar para Perfuração". Uma borda preta é pintada ao redor do conjunto selecionado, como exibe a figura 4.

Figura 4 - Borda preta indicando o conjunto selecionado para perfuração

Duas opções de persistência para a base de regras são providas para o usuário. Através do botão "Exportar Arquivo" do componente gráfico de um conjunto de regras, o usuário pode

exportar um conjunto de regras da base para um arquivo, que pode posteriormente ser carregado para uma base através do botão “Carregar Arquivo”.

Essa opção possibilita que um usuário possa criar conjuntos de regras em um computador, exportá-los para uma mídia de armazenamento e carregá-los para outro computador.

A segunda opção para persistência pode ser acessada através do botão “Salvar no BD”, que armazena toda a base de regras no banco de dados do SO-BR. Sempre que o editor gráfico de regras é aberto, todos os conjuntos de regras salvos no banco de dados são carregados.

4.3. Validação do módulo de regras integrado ao SO-BR

Após ambos os componentes, engine e editor de regras, terem sido validados isoladamente, prosseguiu-se para a integração e validação do módulo de regras no SO-BR, simulando o acompanhamento de uma perfuração real.

Para isso, modificou-se uma planilha de dados reais de uma perfuração, fornecida pela Petrobras (Cenpes). Então, com o SO-BR em execução, o editor de regras foi acessado pela interface do programa e uma regra contendo as condições esperadas para sua ativação (de acordo com as modificações no arquivo de dados) foi criada.

A simulação foi iniciada e foram adicionados componentes gráficos na interface do SO-BR para o acompanhamento das observações e tendência das variáveis. Tendo passado um certo tempo de simulação, o programa exibiu o alerta esperado. A figura 5 mostra a exibição do alerta de vibração.

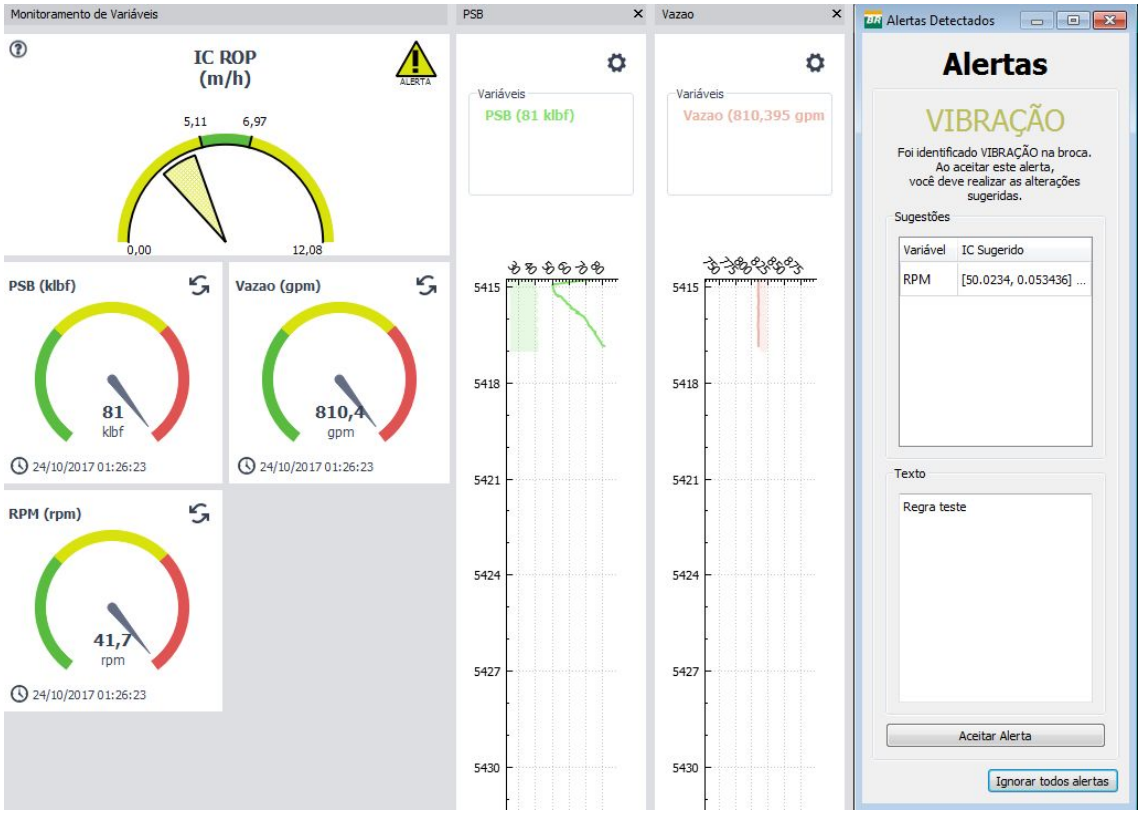

Figura 5 - Alerta da base de regras sendo exibido durante simulação no SO-BR

6. Conclusão

Este trabalho teve como objetivo o desenvolvimento de um módulo de regras integrado ao software SO-BR, um sistema de acompanhamento e apoio à tomada de decisão em perfurações de poços do pré-sal.

O módulo foi dividido em dois componentes, uma engine, que provê o mecanismo de verificação das regras, e um editor gráfico, que permite a criação delas. Para a programação da engine, a ferramenta CLIPS foi utilizada.

Após ambos os componentes terem sido validados isoladamente, o módulo foi integrado ao SO-BR e validado em conjunto com seu módulo de controle, simulando uma perfuração real e verificando a ativação das regras definidas.

A ênfase deste trabalho foi na área de petróleo, na perfuração de poços do pré sal, porém a abordagem utilizada é aplicável a diversas outras áreas.

Como trabalhos futuros, algumas funcionalidades poderiam ser adicionadas ao módulo, como a criação de outros tipos de alertas e regras com condições e sugestões mais complexas.

References

Luger, George F.. Inteligência artificial. 6. ed. São Paulo: Pearson Education do Brasil, 2013.

Barr, Avron; Feigenbaum, Edward A.. The handbook of artificial intelligence. Los Altos: William Kaufmann, Inc., 1981.

Cox, D. R.; Stuart, A. Some quick tests for trend in location and dispersion. Biometrika, London, v. 42, p.80-95, 1955.

APÊNDICE C - Código Fonte

```
#ifndef CONJUNTO_DE_REGRAS_WIDGET_H
#define CONJUNTO_DE_REGRAS_WIDGET_H

#include <QWidget>
#include "ui_conjunto_de_regras_widget.h"

#include "variavel.h"

class ConjuntoDeRegrasWidget : public QWidget
{
	Q_OBJECT

public:
	ConjuntoDeRegrasWidget(QWidget *parent = Q_NULLPTR);
	ConjuntoDeRegrasWidget(QString nome, int numero, const
std::vector<Variavel> variaveis, QWidget *parent = Q_NULLPTR);

	void nome(QString nome);
	void numeroRegras(int numero);
	void variaveis(const std::vector<Variavel> variaveis);

signals:
	void renomearPressionado();
	void fecharPressionado();
	void exportarPressionado();
	void selecionarPressionado();

private:
	Ui::ConjuntoDeRegrasWidget ui;
};

#endif

#ifndef JANELA_CONJUNTO_DE_REGRAS_H
#define JANELA_CONJUNTO_DE_REGRAS_H

#include <QDialog>
#include "./GeneratedFiles/ui_janela_conjuntos_de_regras.h"
```

```
#include "regra.h"
#include "variavel.h"
#include "conjunto_de_regras.h"

class JanelaConjuntosDeRegras : public QDialog
{
	Q_OBJECT

public:
	JanelaConjuntosDeRegras(const std::vector<Variavel> &variaveis, QWidget *parent = Q_NULLPTR);
	void setVariaveis(const std::vector<Variavel> &variaveis);

private slots:
	void adicionarConjunto();
	void excluirConjunto();
	void renomearConjunto();
	void exportarArquivo();
	void selecionarConjunto();

	void adicionarRegra();
	void editarRegra();
	void excluirRegra();

	void salvarNoBD();
	void carregarArquivo();

	void conjuntoSelecionado(int linha, int coluna);

signals:
	void regrasSelecionadas(QString regras);

private:
	void adicionarConjuntoDeRegrasWidget(const ConjuntoDeRegras &conjunto);
	void adicionarRegraWidget(const Regra &regra);
	QString proximoNomeConjunto();

	int indiceConjuntoSelecionado;
```

```
        std::vector<std::shared_ptr<ConjuntoDeRegras>> conjuntos;

        std::vector<Variavel> variaveis;

        Ui::JanelaConjuntosDeRegras ui;
};

#endif

#ifndef JANELA_CRIACAO_REGRA_H
#define JANELA_CRIACAO_REGRA_H

#include <QDialog>
#include "./GeneratedFiles/ui_janela_criacao_regra.h"

#include "enums.h"
#include "regra.h"

class Variavel;

class JanelaCriacaoRegra : public QDialog
{
        Q_OBJECT

public:
        JanelaCriacaoRegra(std::vector<Variavel> variaveis, std::vector<Regra>
regrasExistentes, QWidget *parent = Q_NULLPTR);
        JanelaCriacaoRegra(Regra regra, std::vector<Variavel> variaveis,
std::vector<Regra> regrasExistentes, QWidget *parent = Q_NULLPTR);
        Regra regra();

private slots:
        void accept();
        void adicionarCondicao();
        void removerCondicao();
        void adicionarSugestao();
        void removerSugestao();
        void comboBoxVariavelSugestaoChanged(int index);
        void comboBoxTipoChanged(int index);
```

```
private:
	void setup();

	bool condicoesConsistentes();

	std::vector<Variavel> variaveis;
	std::vector<QString> variaveisNaoSelecionadas;
	std::vector<Tendencia> tendencias;
	std::vector<Conector> conectores;
	std::vector<TipoRegra> tipos;

	QString nomeAntigo;
	std::vector<Regra> regrasExistentes;

	Ui::JanelaCriacaoRegra ui;
};

#endif

#ifndef REGRA_WIDGET_H
#define REGRA_WIDGET_H

#include <QWidget>
#include "GeneratedFiles\ui_regra_widget.h"

#include "regra.h"

class RegraWidget : public QWidget
{
	Q_OBJECT

public:
	RegraWidget(QWidget *parent = Q_NULLPTR);
	RegraWidget(Regra regra, QWidget *parent = Q_NULLPTR);

signals:
	void editarPressionado();
	void fecharPressionado();
```

```
private:
	Ui::RegraWidget ui;
};

#endif

#ifndef BASE_DE_REGRAS_H
#define BASE_DE_REGRAS_H

#include <vector>
#include <map>

#include <lib_clipsmm/clipsmm.h>

#include <QObject>

#include "configuracao_alerta.h"
#include "enums.h"
#include "type_defs.h"
#include <QString>

#include <string>


class BaseDeRegras : public QObject
{
	Q_OBJECT

public:
	typedef std::map<TipoRegra, ConfiguracaoAlerta> Resultado;

	Resultado ativar(DadosV2 dados);

	static bool testaExclusividadeMutua(const std::vector<Variavel> &v, QString regraClipsComposta);

public slots:
	void carregarRegras(QString regras);
```

```
private:
	CLIPS::Environment environment;
};

#endif

#include "conjunto_de_regras_widget.h"

ConjuntoDeRegrasWidget::ConjuntoDeRegrasWidget(QWidget *parent)
	: QWidget(parent)
{
	ui.setupUi(this);

	connect(ui.pushButton, static_cast<void (QPushButton::*)(bool)>(&QPushButton::clicked), this, &ConjuntoDeRegrasWidget::renomearPressionado);
	connect(ui.pushButtonFechar, static_cast<void (QPushButton::*)(bool)>(&QPushButton::clicked), this, &ConjuntoDeRegrasWidget::fecharPressionado);
	connect(ui.pushButton_6, static_cast<void (QPushButton::*)(bool)>(&QPushButton::clicked), this, &ConjuntoDeRegrasWidget::exportarPressionado);
	connect(ui.pushButton_2, static_cast<void (QPushButton::*)(bool)>(&QPushButton::clicked), this, &ConjuntoDeRegrasWidget::selecionarPressionado);
}

ConjuntoDeRegrasWidget::ConjuntoDeRegrasWidget(QString nome, int numero, const std::vector<Variavel> variaveis, QWidget *parent)
	: QWidget(parent)
{
	ui.setupUi(this);

	connect(ui.pushButton, static_cast<void (QPushButton::*)(bool)>(&QPushButton::clicked), this, &ConjuntoDeRegrasWidget::renomearPressionado);
```

```
	connect(ui.pushButtonFechar, static_cast<void (QPushButton::*)(bool)>(&QPushButton::clicked), this, &ConjuntoDeRegrasWidget::fecharPressionado);
	connect(ui.pushButton_6, static_cast<void (QPushButton::*)(bool)>(&QPushButton::clicked), this, &ConjuntoDeRegrasWidget::exportarPressionado);
	connect(ui.pushButton_2, static_cast<void (QPushButton::*)(bool)>(&QPushButton::clicked), this, &ConjuntoDeRegrasWidget::selecionarPressionado);

	this->nome(nome);
	this->numeroRegras(numero);
	this->variaveis(variaveis);
}

void ConjuntoDeRegrasWidget::nome(QString nome)
{
	ui.label_2->setText(nome);
}

void ConjuntoDeRegrasWidget::numeroRegras(int numero)
{
	ui.label_7->setText(QString::number(numero));
}

void ConjuntoDeRegrasWidget::variaveis(const std::vector<Variavel> variaveis)
{
	QString vars;
	for (Variavel v : variaveis) {
		vars += v.nome() + ", ";
	}

	ui.label->setText(vars.left(vars.size() - 2));
}

#include "janela_conjuntos_de_regras.h"

#include <QInputDialog>
#include <QDir>
```

```
#include <QMessageBox>
#include <QFileDialog>
#include <QTextStream>

#include "conjunto_de_regras.h"
#include "conjunto_de_regras_widget.h"
#include "janela_criacao_regra.h"
#include "utilidades_regras.h"
#include "regra_widget.h"
#include "dao_conjunto_de_regras.h"

JanelaConjuntosDeRegras::JanelaConjuntosDeRegras(const std::vector<Variavel>
&variaveis, QWidget *parent)
	: indiceConjuntoSelecionado(0), variaveis(variaveis), QDialog(parent)
{
	ui.setupUi(this);

	this->setWindowFlags(this->windowFlags() &
~Qt::WindowContextHelpButtonHint);

	connect(ui.pushButton_5, &QPushButton::clicked, this,
&JanelaConjuntosDeRegras::salvarNoBD);
	connect(ui.pushButton_6, &QPushButton::clicked, this,
&JanelaConjuntosDeRegras::carregarArquivo);
	connect(ui.tableWidget_2, static_cast<void(QTableWidget::*)(int,
int)>(&QTableWidget::cellClicked), this,
static_cast<void(JanelaConjuntosDeRegras::*)(int,
int)>(&JanelaConjuntosDeRegras::conjuntoSelecionado));

	ui.tableWidget_2->setColumnCount(1);

	ui.tableWidget_2->verticalHeader()->setVisible(false);

ui.tableWidget_2->verticalHeader()->setSectionResizeMode(QHeaderView::ResizeToC
ontents);
	ui.tableWidget_2->horizontalHeader()->setVisible(false);

ui.tableWidget_2->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeT
oContents);
```

```
	ui.tableWidget_2->horizontalHeader()->setStretchLastSection(true);
	ui.tableWidget_2->setSelectionMode(QAbstractItemView::SingleSelection);

	ui.tableWidget->setColumnCount(1);

	ui.tableWidget->verticalHeader()->setVisible(false);

ui.tableWidget->verticalHeader()->setSectionResizeMode(QHeaderView::ResizeToCont
ents);
	ui.tableWidget->horizontalHeader()->setVisible(false);

ui.tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToC
ontents);
	ui.tableWidget->horizontalHeader()->setStretchLastSection(true);

	QPalette palette = ui.tableWidget_2->palette();
	palette.setBrush(QPalette::Highlight, QBrush(Qt::gray));
	palette.setBrush(QPalette::HighlightedText, QBrush(Qt::black));
	ui.tableWidget_2->setPalette(palette);

	QPushButton *p = new QPushButton(this);
	QFont font;
	font.setPointSize(16);
	p->setFont(font);
	p->setText("+");
	p->setMinimumHeight(30);
	ui.tableWidget_2->setRowCount(ui.tableWidget_2->rowCount() + 1);
	ui.tableWidget_2->setCellWidget(ui.tableWidget_2->rowCount() - 1, 0, p);
	connect(p,                    &QPushButton::clicked,                    this,
&JanelaConjuntosDeRegras::adicionarConjunto);

	conjuntos = DaoConjuntoDeRegras::getAll();
	for (std::shared_ptr<ConjuntoDeRegras> conjunto : conjuntos) {
		adicionarConjuntoDeRegrasWidget(*conjunto);
	}
}

void JanelaConjuntosDeRegras::setVariaveis(const std::vector<Variavel> &variaveis)
{
```

```
	this->variaveis = variaveis;
}

void JanelaConjuntosDeRegras::adicionarConjunto()
{
	QString nome = proximoNomeConjunto();


conjuntos.push_back(std::make_shared<ConjuntoDeRegras>(ConjuntoDeRegras(nom
e)));

	adicionarConjuntoDeRegrasWidget(ConjuntoDeRegras(nome));
}

void JanelaConjuntosDeRegras::excluirConjunto()
{
	for (int i = 0; i < ui.tableWidget_2->rowCount(); ++i) {
		if (sender() == ui.tableWidget_2->cellWidget(i, 0)) {
			ui.tableWidget_2->removeRow(i);
			conjuntos.erase(conjuntos.begin() + i);
		}
	}

	ui.tableWidget->setRowCount(0);
}

void JanelaConjuntosDeRegras::renomearConjunto()
{
	bool ok;

	while (true) {
		QString nome = QInputDialog::getText(this, tr("Renomear Conjunto"),
			tr("Insira o novo nome:"), QLineEdit::Normal, "", &ok);

		if (!ok) {
			return;
		}

		for (int i = 0; i < conjuntos.size(); ++i) {
```

```
                        if (nome == (*conjuntos[i]).nome) {
                                QMessageBox::warning(this,  "Aviso",  QString::fromUtf8("Já
existe um conjunto com o nome inserido, escolha outro. "));
                                break;
                        }

                        if (i == conjuntos.size() - 1) {
                                for (int i = 0; i < ui.tableWidget_2->rowCount(); ++i) {
                                        if (sender() == ui.tableWidget_2->cellWidget(i, 0)) {

qobject_cast<ConjuntoDeRegrasWidget*>(ui.tableWidget_2->cellWidget(i,
0))->nome(nome);
                                                (*conjuntos[i]).nome = nome;
                                        }
                                }

                                return;
                        }
                }
        }
}

void JanelaConjuntosDeRegras::exportarArquivo()
{
        QString caminho = QFileDialog::getSaveFileName(this,
                tr("Salvar Regras"), "",
                tr("Arquivo de Regras (*.regras);;All Files (*)"));

        if (caminho.isEmpty()) {
                return;
        }

        QFile file(caminho);

        if (file.open(QIODevice::ReadWrite))
        {
                QTextStream stream(&file);

                for (int i = 0; i < ui.tableWidget_2->rowCount(); ++i) {
```

```
                        if (sender() == ui.tableWidget_2->cellWidget(i, 0)) {
                                stream << (*(conjuntos[i])).regras_clips << endl;
                        }
                }
        }
}

void JanelaConjuntosDeRegras::selecionarConjunto()
{
        for (int i = 0; i < ui.tableWidget_2->rowCount(); ++i) {
                if (sender() == ui.tableWidget_2->cellWidget(i, 0)) {
                        ui.tableWidget_2->cellWidget(i,          0)->setStyleSheet("#frame
{border-color: black; border-width: 3px; border-style: solid;}");
                        emit regrasSelecionadas((*(conjuntos[i])).regras_clips);
                }
                else {
                        ui.tableWidget_2->cellWidget(i, 0)->setStyleSheet("");
                }
        }
}

void JanelaConjuntosDeRegras::adicionarRegra()
{
        std::vector<Regra>                              regras                          =
UtilidadesRegras::lerBaseDeRegrasClips((*conjuntos[indiceConjuntoSelecionado]).regr
as_clips);

        JanelaCriacaoRegra *janela = new JanelaCriacaoRegra(variaveis, regras, this);

        if (janela->exec() == QDialog::Rejected) {
                return;
        }

        Regra regraNova = janela->regra();

        regras.push_back(regraNova);

        (*conjuntos[indiceConjuntoSelecionado]).regras_clips                            =
UtilidadesRegras::construirBaseDeRegrasClips(regras);
```

```
	ConjuntoDeRegrasWidget *conjuntoWidget =
qobject_cast<ConjuntoDeRegrasWidget*>(ui.tableWidget_2->cellWidget(indiceConjunt
oSelecionado, 0));
	conjuntoWidget->numeroRegras(regras.size());

conjuntoWidget->variaveis(UtilidadesRegras::variaveisDoConjuntoDeRegras((*conjunto
s[indiceConjuntoSelecionado])));

	adicionarRegraWidget(regraNova);
}

void JanelaConjuntosDeRegras::editarRegra()
{
	for (int i = 0; i < ui.tableWidget->rowCount(); ++i) {
		if (sender() == ui.tableWidget->cellWidget(i, 0)) {
			std::vector<Regra> regras =
UtilidadesRegras::lerBaseDeRegrasClips((*conjuntos[indiceConjuntoSelecionado]).regr
as_clips);

			JanelaCriacaoRegra *janela = new JanelaCriacaoRegra(regras[i],
variaveis, regras, this);

			if (janela->exec() == QDialog::Rejected) {
				return;
			}

			Regra regra = janela->regra();

			regras[i] = regra;

			(*conjuntos[indiceConjuntoSelecionado]).regras_clips =
UtilidadesRegras::construirBaseDeRegrasClips(regras);

			ConjuntoDeRegrasWidget *conjuntoWidget =
qobject_cast<ConjuntoDeRegrasWidget*>(ui.tableWidget_2->cellWidget(indiceConjunt
oSelecionado, 0));
```

```
conjuntoWidget->variaveis(UtilidadesRegras::variaveisDoConjuntoDeRegras((*conjuntos[indiceConjuntoSelecionado])));

                        RegraWidget *regraWidget = new RegraWidget(regra, this);

                        if (regra.tipo == TipoRegra::vibracao) {
                                regraWidget->setStyleSheet("#frame {border-color: green; border-width: 3px; border-style: solid;}");
                        }
                        else if (regra.tipo == TipoRegra::enceramento) {
                                regraWidget->setStyleSheet("#frame {border-color: blue; border-width: 3px; border-style: solid;}");
                        }
                        else if (regra.tipo == TipoRegra::desgaste) {
                                regraWidget->setStyleSheet("#frame {border-color: red; border-width: 3px; border-style: solid;}");
                        }

                        connect(regraWidget, &RegraWidget::editarPressionado, this, &JanelaConjuntosDeRegras::editarRegra);
                        connect(regraWidget, &RegraWidget::fecharPressionado, this, &JanelaConjuntosDeRegras::excluirRegra);

                        ui.tableWidget->setCellWidget(i, 0, regraWidget);
                }
        }
}

void JanelaConjuntosDeRegras::excluirRegra()
{
        for (int i = 0; i < ui.tableWidget->rowCount(); ++i) {
                if (sender() == ui.tableWidget->cellWidget(i, 0)) {
                        ui.tableWidget->removeRow(i);
                        std::vector<Regra> regras = UtilidadesRegras::lerBaseDeRegrasClips((*conjuntos[indiceConjuntoSelecionado]).regras_clips);
                        regras.erase(regras.begin() + i);
```

```
                    (*conjuntos[indiceConjuntoSelecionado]).regras_clips =
UtilidadesRegras::construirBaseDeRegrasClips(regras);

                    ConjuntoDeRegrasWidget *conjuntoWidget =
qobject_cast<ConjuntoDeRegrasWidget*>(ui.tableWidget_2->cellWidget(indiceConjunt
oSelecionado, 0));
                    conjuntoWidget->numeroRegras(regras.size());

conjuntoWidget->variaveis(UtilidadesRegras::variaveisDoConjuntoDeRegras((*conjunto
s[indiceConjuntoSelecionado])));
                }
        }
}

void JanelaConjuntosDeRegras::adicionarConjuntoDeRegrasWidget(const
ConjuntoDeRegras &conjunto)
{
        ConjuntoDeRegrasWidget *conjuntoWidget = new
ConjuntoDeRegrasWidget(conjunto.nome, 0, {}, this);

conjuntoWidget->numeroRegras(UtilidadesRegras::lerBaseDeRegrasClips(conjunto.reg
ras_clips).size());

conjuntoWidget->variaveis(UtilidadesRegras::variaveisDoConjuntoDeRegras(conjunto))
;

        ui.tableWidget_2->insertRow(ui.tableWidget_2->rowCount() - 1);
        ui.tableWidget_2->setCellWidget(ui.tableWidget_2->rowCount() - 2, 0,
conjuntoWidget);

        connect(conjuntoWidget, &ConjuntoDeRegrasWidget::fecharPressionado, this,
&JanelaConjuntosDeRegras::excluirConjunto);
        connect(conjuntoWidget, &ConjuntoDeRegrasWidget::renomearPressionado,
this, &JanelaConjuntosDeRegras::renomearConjunto);
        connect(conjuntoWidget, &ConjuntoDeRegrasWidget::exportarPressionado, this,
&JanelaConjuntosDeRegras::exportarArquivo);
        connect(conjuntoWidget, &ConjuntoDeRegrasWidget::selecionarPressionado,
this, &JanelaConjuntosDeRegras::selecionarConjunto);
}
```

```
void JanelaConjuntosDeRegras::adicionarRegraWidget(const Regra &regra)
{
	RegraWidget *regraWidget = new RegraWidget(regra, this);

	if (regra.tipo == TipoRegra::vibracao) {
		regraWidget->setStyleSheet("#frame {border-color: rgb(0, 255, 0); border-width: 3px; border-style: solid;}");
	}
	else if (regra.tipo == TipoRegra::enceramento) {
		regraWidget->setStyleSheet("#frame {border-color: blue; border-width: 3px; border-style: solid;}");
	}
	else if (regra.tipo == TipoRegra::desgaste) {
		regraWidget->setStyleSheet("#frame {border-color: red; border-width: 3px; border-style: solid;}");
	}

	connect(regraWidget, &RegraWidget::editarPressionado, this, &JanelaConjuntosDeRegras::editarRegra);
	connect(regraWidget, &RegraWidget::fecharPressionado, this, &JanelaConjuntosDeRegras::excluirRegra);

	ui.tableWidget->insertRow(ui.tableWidget->rowCount() - 1);
	ui.tableWidget->setCellWidget(ui.tableWidget->rowCount() - 2, 0, regraWidget);
}

void JanelaConjuntosDeRegras::salvarNoBD()
{
	std::vector<std::shared_ptr<ConjuntoDeRegras>> conjuntosNoBD = DaoConjuntoDeRegras::getAll();
	try
	{
		DBConnection con;
		con.begin();

		for (std::shared_ptr<ConjuntoDeRegras> conjunto : conjuntosNoBD) {
			for (int i = 0; i < conjuntos.size(); ++i) {
				if ((*conjunto).id() == (*conjuntos[i]).id()) {
```

```
                                        break;
                                }

                                if (i == conjuntos.size() - 1) {
                                        DaoConjuntoDeRegras::remove((*conjunto).nome,
&con);
                                }
                        }
                }

                con.commit();
        }
        catch (DBError &e)
        {
                DaoObject::erroBD(Q_FUNC_INFO, e);
        }

        for (std::shared_ptr<ConjuntoDeRegras> conjunto : conjuntos) {
                DaoConjuntoDeRegras::put(conjunto, true);
        }
}

void JanelaConjuntosDeRegras::carregarArquivo()
{
        QString caminho = QFileDialog::getOpenFileName(this,
                tr("Salvar Regras"), "",
                tr("Arquivo de Regras (*.regras);;All Files (*)"));

        if (caminho.isEmpty()) {
                return;
        }

        QFile file(caminho);

        QString baseDeRegras;

        if (file.open(QIODevice::ReadWrite))
        {
                QTextStream stream(&file);
```

```
            baseDeRegras = stream.readAll();
        }

        adicionarConjunto();

        (*conjuntos[conjuntos.size() - 1]).regras_clips = baseDeRegras;


qobject_cast<ConjuntoDeRegrasWidget*>(ui.tableWidget_2->cellWidget(conjuntos.size
() - 1,
0))->numeroRegras(UtilidadesRegras::lerBaseDeRegrasClips(baseDeRegras).size());

qobject_cast<ConjuntoDeRegrasWidget*>(ui.tableWidget_2->cellWidget(conjuntos.size
() - 1,
0))->variaveis(UtilidadesRegras::variaveisDoConjuntoDeRegras(ConjuntoDeRegras("",
baseDeRegras)));
}

void JanelaConjuntosDeRegras::conjuntoSelecionado(int linha, int coluna)
{
        indiceConjuntoSelecionado = linha;

        ui.tableWidget->setRowCount(0);

        QPushButton *p = new QPushButton(this);
        QFont font;
        font.setPointSize(16);
        p->setFont(font);
        p->setText("+");
        p->setMinimumHeight(30);
        ui.tableWidget->setRowCount(ui.tableWidget->rowCount() + 1);
        ui.tableWidget->setCellWidget(ui.tableWidget->rowCount() - 1, 0, p);
        connect(p, &QPushButton::clicked, this,
&JanelaConjuntosDeRegras::adicionarRegra);

        std::vector<Regra> regrasClips =
UtilidadesRegras::lerBaseDeRegrasClips((*conjuntos[indiceConjuntoSelecionado]).regr
as_clips);
```

```
        for (Regra r : regrasClips) {
                adicionarRegraWidget(r);
        }
}


QString JanelaConjuntosDeRegras::proximoNomeConjunto()
{
        QString nome = "Conjunto 1";
        int n = 1;

        if (conjuntos.empty()) {
                return nome;
        }

        while (true) {
                for (int i = 0; i < conjuntos.size(); ++i) {
                        if (nome == (*conjuntos[i]).nome) {
                                nome = "Conjunto " + QString::number(++n);
                                break;
                        }

                        if (i == conjuntos.size() - 1) {
                                return nome;
                        }
                }
        }
}

#include "janela_criacao_regra.h"

#include <QComboBox>
#include <QPushButton>
#include <QLineEdit>
#include <QMessageBox>

#include <vector>
#include <set>
#include <algorithm>
```

```
#include "variavel.h"
#include "utilidades.h"
#include "utilidades_regras.h"
#include "base_de_regras.h"

JanelaCriacaoRegra::JanelaCriacaoRegra(std::vector<Variavel> variaveis,
std::vector<Regra> regrasExistentes, QWidget *parent)
	: variaveis(variaveis), regrasExistentes(regrasExistentes), QDialog(parent)
{
	for (int i = 0; i < variaveis.size(); ++i) {
		variaveisNaoSelecionadas.push_back(variaveis[i].nome());
	}

	setup();

	adicionarCondicao();
	ui.tableWidget->removeRow(0);
	QWidget *deletar = ui.tableWidget->cellWidget(0, 4);
	ui.tableWidget->setCellWidget(0, 4, 0);
	delete deletar;

	QPushButton *pushButtonAdicionar = new QPushButton("+", ui.tableWidget2);
	connect(pushButtonAdicionar, static_cast<void
(QPushButton::*)(bool)>(&QPushButton::clicked), this,
&JanelaCriacaoRegra::adicionarSugestao);
	ui.tableWidget2->setCellWidget(0, 6, pushButtonAdicionar);
}

JanelaCriacaoRegra::JanelaCriacaoRegra(Regra regra, std::vector<Variavel> variaveis,
std::vector<Regra> regrasExistentes, QWidget *parent)
	: nomeAntigo(regra.nome), variaveis(variaveis),
regrasExistentes(regrasExistentes), QDialog(parent)
{
	for (int i = 0; i < variaveis.size(); ++i) {
		variaveisNaoSelecionadas.push_back(variaveis[i].nome());
	}

	setup();
```

```
        ui.lineEdit->setText(regra.nome);
        int indice = ui.comboBox->findText(mapTipoRegraQStringGUI[regra.tipo]);
        ui.comboBox->setCurrentIndex(indice);
        ui.textEdit->setPlainText(regra.texto);

        for (int i = 0; i < regra.condicoes.size(); ++i) {
                adicionarCondicao();
                Regra::Condicao condicao = regra.condicoes[i];
                QComboBox *comboBoxVariavel = qobject_cast<QComboBox
*>(ui.tableWidget->cellWidget(2 * i + 1, 1));
                int indice = comboBoxVariavel->findText(condicao.variavel.nome());

comboBoxVariavel->setCurrentIndex(comboBoxVariavel->findText(condicao.variavel.no
me()));
                QComboBox *comboBoxTendencia = qobject_cast<QComboBox
*>(ui.tableWidget->cellWidget(2 * i + 1, 3));
                indice =
comboBoxTendencia->findText(mapTendenciaQStringGUI[condicao.tendencia]);
                comboBoxTendencia->setCurrentIndex(indice);
        }

        for (int i = 0; i < regra.conectores.size(); ++i) {
                QComboBox *comboBoxConector = qobject_cast<QComboBox
*>(ui.tableWidget->cellWidget(2 * i + 2, 0));
                int indice =
comboBoxConector->findText(Utilidades::stdStringToQString(mapConectorString[regra.
conectores[i]]));
                comboBoxConector->setCurrentIndex(indice);
        }

        ui.tableWidget->removeRow(0);
        QWidget *deletar = ui.tableWidget->cellWidget(0, 5);
        ui.tableWidget->setCellWidget(0, 5, 0);
        delete deletar;

        QPushButton *pushButtonAdicionar = new QPushButton("+", ui.tableWidget2);
```

```
	connect(pushButtonAdicionar, static_cast<void (QPushButton::*)(bool)>(&QPushButton::clicked), this, &JanelaCriacaoRegra::adicionarSugestao);
	ui.tableWidget2->setCellWidget(0, 6, pushButtonAdicionar);

	for (int i = 0; i < regra.sugestoes.size(); ++i) {
		adicionarSugestao();
		QComboBox *comboBoxVariavel = qobject_cast<QComboBox *>(ui.tableWidget2->cellWidget(i, 1));
		int indice = comboBoxVariavel->findText(regra.sugestoes[i].variavel.nome());
		comboBoxVariavel->setCurrentIndex(indice);

		if (regra.sugestoes[i].porcentagem >= 0) {
			qobject_cast<QComboBox *>(ui.tableWidget2->cellWidget(i, 3))->setCurrentIndex(1);
		}
		else {
			qobject_cast<QComboBox *>(ui.tableWidget2->cellWidget(i, 3))->setCurrentIndex(2);
		}

		qobject_cast<QLineEdit *>(ui.tableWidget2->cellWidget(i, 4))->setText(QString::number(regra.sugestoes[i].porcentagem));
	}
}

void JanelaCriacaoRegra::setup()
{
	ui.setupUi(this);

	this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint);

	tendencias = std::vector<Tendencia>({ Tendencia::crescimento, Tendencia::constante, Tendencia::decrescimento });
	conectores = std::vector<Conector>({ Conector::e, Conector::ou });
	tipos = std::vector<TipoRegra>({ TipoRegra::vibracao, TipoRegra::enceramento, TipoRegra::desgaste });
```

```
	connect(ui.pushButtonOK, &QPushButton::clicked, this, &JanelaCriacaoRegra::accept);
	connect(ui.pushButtonCancelar, &QPushButton::clicked, this, &JanelaCriacaoRegra::reject);
	connect(ui.comboBox, static_cast<void (QComboBox::*)(int)>(&QComboBox::currentIndexChanged), this, static_cast<void (JanelaCriacaoRegra::*)(int)>(&JanelaCriacaoRegra::comboBoxTipoChanged));

	ui.tableWidget->verticalHeader()->setVisible(false);
	ui.tableWidget->horizontalHeader()->setVisible(false);

ui.tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);
	ui.tableWidget->horizontalHeader()->setStretchLastSection(true);

	ui.tableWidget2->verticalHeader()->setVisible(false);
	ui.tableWidget2->horizontalHeader()->setVisible(false);

ui.tableWidget2->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);
	ui.tableWidget2->horizontalHeader()->setStretchLastSection(true);

	ui.tableWidget->setColumnCount(5);
	ui.tableWidget->setRowCount(1);

	ui.tableWidget2->setColumnCount(7);
	ui.tableWidget2->setRowCount(1);
}

Regra JanelaCriacaoRegra::regra()
{
	Regra regra;

	regra.nome = ui.lineEdit->text();

	for (int i = 0; i < ui.tableWidget->rowCount() - 1; ++i) {
		if (i % 2 == 0) {
			Regra::Condicao condicao;
```

```
                condicao.variavel = variaveis[qobject_cast<QComboBox
*>(ui.tableWidget->cellWidget(i, 1))->currentIndex() - 1];
                condicao.tendencia = tendencias[qobject_cast<QComboBox
*>(ui.tableWidget->cellWidget(i, 3))->currentIndex() - 1];

                regra.condicoes.push_back(condicao);
            }
            else {

regra.conectores.push_back(conectores[qobject_cast<QComboBox
*>(ui.tableWidget->cellWidget(i, 0))->currentIndex()]);
            }
        }

        regra.tipo = tipos[ui.comboBox->currentIndex() - 1];

        for (int i = 0; i < ui.tableWidget2->rowCount() - 1; ++i) {
            Regra::Sugestao sugestao;

            QString nomeVariavel = qobject_cast<QComboBox
*>(ui.tableWidget2->cellWidget(i, 1))->currentText();

            for (int n = 0; n < variaveis.size(); ++n) {
                if (variaveis[n].nome() == nomeVariavel) {
                    sugestao.variavel = variaveis[n];
                    break;
                }
            }

            sugestao.porcentagem = qobject_cast<QLineEdit
*>(ui.tableWidget2->cellWidget(i, 4))->text().toDouble();
            if (qobject_cast<QComboBox *>(ui.tableWidget2->cellWidget(i,
3))->currentIndex() == 2) {
                sugestao.porcentagem *= -1; // se selecionou porcentagem
negativa
            }

            regra.sugestoes.push_back(sugestao);
        }
```

```
	regra.texto = ui.textEdit->toPlainText();

	return regra;
}

void JanelaCriacaoRegra::adicionarCondicao()
{
	QWidget *deletar = ui.tableWidget->cellWidget(ui.tableWidget->rowCount() - 1,
4);
	ui.tableWidget->setCellWidget(ui.tableWidget->rowCount() - 1, 4, 0);
	delete deletar;

	QComboBox *comboBoxConectores = new QComboBox(ui.tableWidget);

comboBoxConectores->addItem(Utilidades::stdStringToQString(mapConectorString[Co
nector::e]));

comboBoxConectores->addItem(Utilidades::stdStringToQString(mapConectorString[Co
nector::ou]));

	ui.tableWidget->setCellWidget(ui.tableWidget->rowCount() - 1, 0,
comboBoxConectores);

	ui.tableWidget->setRowCount(ui.tableWidget->rowCount() + 2);

	QTableWidgetItem *item1 = new QTableWidgetItem("A variável");
	ui.tableWidget->setItem(ui.tableWidget->rowCount() - 2, 0, item1);

	QComboBox *comboBoxVariaveis = new QComboBox(ui.tableWidget);
	comboBoxVariaveis->addItem("");
	for (size_t i = 0; i < variaveis.size(); ++i) {
		comboBoxVariaveis->addItem(variaveis[i].nome());
	}

	ui.tableWidget->setCellWidget(ui.tableWidget->rowCount() - 2, 1,
comboBoxVariaveis);

	QTableWidgetItem *item2 = new QTableWidgetItem("possui tendência");
```

```
        ui.tableWidget->setItem(ui.tableWidget->rowCount() - 2, 2, item2);

        QComboBox *comboBoxTendencias = new QComboBox(ui.tableWidget);
        comboBoxTendencias->addItem("");
        for (size_t i = 0; i < tendencias.size(); ++i) {

comboBoxTendencias->addItem(mapTendenciaQStringGUI[tendencias[i]]);
        }
        ui.tableWidget->setCellWidget(ui.tableWidget->rowCount() - 2, 3,
comboBoxTendencias);

        QPushButton *pushButtonRemover = new QPushButton("-", ui.tableWidget);

pushButtonRemover->setObjectName(QString::number(ui.tableWidget->rowCount() -
2)); //a linha eh guardada no objectname
        connect(pushButtonRemover, static_cast<void
(QPushButton::*)(bool)>(&QPushButton::clicked), this,
&JanelaCriacaoRegra::removerCondicao);
        QWidget *botaoVelho = ui.tableWidget->cellWidget(ui.tableWidget->rowCount() -
2, 4);
        ui.tableWidget->setCellWidget(ui.tableWidget->rowCount() - 2, 4,
pushButtonRemover);
        delete botaoVelho;

        QPushButton *pushButtonAdicionar = new QPushButton("+", ui.tableWidget);
        connect(pushButtonAdicionar, static_cast<void
(QPushButton::*)(bool)>(&QPushButton::clicked), this,
&JanelaCriacaoRegra::adicionarCondicao);
        ui.tableWidget->setCellWidget(ui.tableWidget->rowCount() - 1, 4,
pushButtonAdicionar);
}

void JanelaCriacaoRegra::removerCondicao()
{
        // atualiza os indexes
        for (int i = 2; i < ui.tableWidget->rowCount(); i += 2) {
                QPushButton *pushButton =
qobject_cast<QPushButton*>(ui.tableWidget->cellWidget(i, 4));
```

```
            pushButton->setObjectName(QString::number(i)); //a linha eh guardada no objectname
        }

        QPushButton *pushButtonRemover = dynamic_cast<QPushButton*>(QObject::sender());
        int linha = pushButtonRemover->objectName().toInt(); //a linha do push button eh armazenada na propriedade objectName
        ui.tableWidget->removeRow(linha);
        ui.tableWidget->removeRow(linha - 1);
}

void JanelaCriacaoRegra::adicionarSugestao()
{
        if (variaveisNaoSelecionadas.size() == 0) {
            QMessageBox::warning(this, "Aviso", "Não há mais variáveis disponíveis para sugestões!");
            return;
        }

        QTableWidgetItem *item = new QTableWidgetItem("Alterar o valor da variável ");
        ui.tableWidget2->setItem(ui.tableWidget2->rowCount() - 1, 0, item);

        QComboBox *comboBoxVariaveis = new QComboBox(ui.tableWidget2);
        comboBoxVariaveis->addItem("");
        for (size_t i = 0; i < variaveisNaoSelecionadas.size(); ++i) {
            comboBoxVariaveis->addItem(variaveisNaoSelecionadas[i]);
        }
        connect(comboBoxVariaveis, static_cast<void (QComboBox::*)(int)>(&QComboBox::currentIndexChanged), this, &JanelaCriacaoRegra::comboBoxVariavelSugestaoChanged);
        ui.tableWidget2->setCellWidget(ui.tableWidget2->rowCount() - 1, 1, comboBoxVariaveis);

        QTableWidgetItem *item2 = new QTableWidgetItem("em");
        ui.tableWidget2->setItem(ui.tableWidget2->rowCount() - 1, 2, item2);

        QComboBox *comboBoxSinal = new QComboBox(ui.tableWidget2);
        comboBoxSinal->addItem("");
```

```
        comboBoxSinal->addItem("+");
        comboBoxSinal->addItem("-");
        ui.tableWidget2->setCellWidget(ui.tableWidget2->rowCount() - 1, 3,
comboBoxSinal);

        QLineEdit *lineEditValor = new QLineEdit(ui.tableWidget2);
        lineEditValor->setMaximumWidth(50);
        ui.tableWidget2->setCellWidget(ui.tableWidget2->rowCount() - 1, 4,
lineEditValor);

        QTableWidgetItem *item3 = new QTableWidgetItem("porcento.");
        ui.tableWidget2->setItem(ui.tableWidget2->rowCount() - 1, 5, item3);

        QPushButton *pushButtonRemover = new QPushButton("-", ui.tableWidget2);

pushButtonRemover->setObjectName(QString::number(ui.tableWidget2->rowCount() -
1)); //a linha eh guardada no objectname
        connect(pushButtonRemover, static_cast<void
(QPushButton::*)(bool)>(&QPushButton::clicked), this,
&JanelaCriacaoRegra::removerSugestao);
        QWidget *botaoVelho =
ui.tableWidget2->cellWidget(ui.tableWidget2->rowCount() - 1, 6);
        ui.tableWidget2->setCellWidget(ui.tableWidget2->rowCount() - 1, 6,
pushButtonRemover);
        delete botaoVelho;

        QPushButton *pushButtonAdicionar = new QPushButton("+", ui.tableWidget2);
        connect(pushButtonAdicionar, static_cast<void
(QPushButton::*)(bool)>(&QPushButton::clicked), this,
&JanelaCriacaoRegra::adicionarSugestao);
        ui.tableWidget2->setRowCount(ui.tableWidget2->rowCount() + 1);
        ui.tableWidget2->setCellWidget(ui.tableWidget2->rowCount() - 1, 6,
pushButtonAdicionar);
}

void JanelaCriacaoRegra::removerSugestao()
{
        QPushButton *pushButtonRemover =
dynamic_cast<QPushButton*>(QObject::sender());
```

```
	int linha = pushButtonRemover->objectName().toInt(); //a linha do push button eh armazenada na propriedade objectName
	ui.tableWidget2->removeRow(linha);

	// atualiza os indexes
	for (int i = 0; i < ui.tableWidget2->rowCount(); ++i) {
		QPushButton *pushButton = qobject_cast<QPushButton*>(ui.tableWidget2->cellWidget(i, 6));
		pushButton->setObjectName(QString::number(i)); //a linha eh guardada no objectname
	}
}

void JanelaCriacaoRegra::comboBoxVariavelSugestaoChanged(int index)
{
	QComboBox *comboBoxChanged = dynamic_cast<QComboBox*>(QObject::sender());
	QString nomeVariavel = comboBoxChanged->currentText();

	std::vector<QString> variaveisSelecionadas;
	for (int i = 0; i < ui.tableWidget2->rowCount() - 1; ++i) {
		QComboBox *comboBox = qobject_cast<QComboBox *>(ui.tableWidget2->cellWidget(i, 1));
		if (comboBox->currentText() != "" && comboBox->currentText() != nomeVariavel) {
			variaveisSelecionadas.push_back(comboBox->currentText());
		}
	}

	std::vector<QString> nomesVariaveis;
	for (int i = 0; i < variaveis.size(); ++i) {
		nomesVariaveis.push_back(variaveis[i].nome());
	}

	std::vector<QString> variaveisSelecionadasAnteriormente = UtilidadesRegras::diferencaVetores(nomesVariaveis, variaveisNaoSelecionadas);
```

```
	std::vector<QString> variavelDesselecionada = UtilidadesRegras::diferencaVetores(variaveisSelecionadasAnteriormente, variaveisSelecionadas);
	if (variavelDesselecionada.size() > 0) {
		variaveisNaoSelecionadas.push_back(variavelDesselecionada[0]);

		for (int i = 0; i < ui.tableWidget2->rowCount() - 1; ++i) {
			QComboBox *comboBox = qobject_cast<QComboBox *>(ui.tableWidget2->cellWidget(i, 1));
			if (comboBox->findText(variavelDesselecionada[0]) == -1) {
				comboBox->addItem(variavelDesselecionada[0]);
			}
		}
	}

	if (nomeVariavel != "") {
		for (int i = 0; i < ui.tableWidget2->rowCount() - 1; ++i) {
			QComboBox *comboBox = qobject_cast<QComboBox *>(ui.tableWidget2->cellWidget(i, 1));
			if (comboBox->currentText() != nomeVariavel) {

comboBox->removeItem(comboBox->findText(nomeVariavel));
			}
		}

		for (int i = 0; i < variaveisNaoSelecionadas.size(); ++i) {
			if (nomeVariavel == variaveisNaoSelecionadas[i]) {

variaveisNaoSelecionadas.erase(variaveisNaoSelecionadas.begin() + i);
			}
		}
	}
}

void JanelaCriacaoRegra::comboBoxTipoChanged(int index)
{
	if (index == 3) {
		while (ui.tableWidget2->rowCount() > 1) {
			ui.tableWidget2->removeRow(0);
```

```
                }

                ui.textEdit->setText("");
        }

        ui.tableWidget2->setEnabled(index != 3);
        ui.textEdit->setEnabled(index != 3);
}

void JanelaCriacaoRegra::accept()
{
        if (ui.lineEdit->text() == "") {
                QMessageBox::warning(this, "Aviso", "Preencha o nome da regra!");
                return;
        }

        for (int i = 0; i < ui.tableWidget->rowCount(); i += 2) {
                QComboBox *comboBoxVariavel = qobject_cast<QComboBox
*>(ui.tableWidget->cellWidget(i, 1));
                QComboBox *comboBoxTendencia = qobject_cast<QComboBox
*>(ui.tableWidget->cellWidget(i, 3));
                if (comboBoxVariavel->currentIndex() == 0 ||
comboBoxTendencia->currentIndex() == 0) {
                        QMessageBox::warning(this, "Aviso", "Preencha todos os combo
boxes!");
                        return;
                }
        }

        if (ui.comboBox->currentIndex() == 0) {
                QMessageBox::warning(this, "Aviso", "Escolha o indicativo que deve ser
acionado!");
                return;
        }

        for (int i = 0; i < ui.tableWidget2->rowCount() - 1; ++i) {
                QComboBox *comboBoxVariavel = qobject_cast<QComboBox
*>(ui.tableWidget2->cellWidget(i, 1));
```

```
            QComboBox *comboBoxSinal = qobject_cast<QComboBox *>(ui.tableWidget2->cellWidget(i, 3));
            if (comboBoxVariavel->currentIndex() == 0 || comboBoxSinal->currentIndex() == 0) {
                QMessageBox::warning(this, "Aviso", "Preencha todos os combo boxes!");
                return;
            }
            QLineEdit *porcentagem = qobject_cast<QLineEdit *>(ui.tableWidget2->cellWidget(i, 4));
            if (porcentagem->text() == "") {
                QMessageBox::warning(this, "Aviso", "Preencha todas as porcentagens!");
                return;
            }
        }

        for (int i = 0; i < regrasExistentes.size(); ++i) {
            if (ui.lineEdit->text() == regrasExistentes[i].nome && ui.lineEdit->text() != nomeAntigo) {
                QMessageBox::warning(this, "Aviso", QString::fromUtf8("Já existe uma regra com o nome ") + ui.lineEdit->text() + ", escolha outra nome!");
                return;
            }
        }

        if (!condicoesConsistentes()) {
            QMessageBox::warning(this, "Aviso", QString::fromUtf8("Regra inconsistente!\nExistem condições com mesma variável e tendências diferentes unidas pelo conector E!"));
            return;
        }

        Regra regra = this->regra();

        for (int i = 0; i < regrasExistentes.size(); ++i) {
            if (regra.nome == regrasExistentes[i].nome || nomeAntigo == regrasExistentes[i].nome) {
                continue;
```

```
                }

                if (regra.tipo == regrasExistentes[i].tipo) {
                        QString regraClips =
UtilidadesRegras::construirRegraClipsComposta(regra, regrasExistentes[i]);

                        std::set<Variavel> variaveis;

                        for (int n = 0; n < regra.condicoes.size(); ++n) {
                                variaveis.insert(regra.condicoes[n].variavel);
                        }

                        for (int n = 0; n < regrasExistentes[i].condicoes.size(); ++n) {
                                variaveis.insert(regrasExistentes[i].condicoes[n].variavel);
                        }

                        if
(!BaseDeRegras::testaExclusividadeMutua(std::vector<Variavel>(variaveis.begin(),
variaveis.end()), regraClips)) {
                                QMessageBox::warning(this, "Aviso",
QString::fromUtf8("Regra não exclusiva com a regra: ") + regrasExistentes[i].nome);
                                return;
                        }
                }
        }

        QDialog::accept();
}

bool JanelaCriacaoRegra::condicoesConsistentes()
{
        Regra regra = this->regra();

        std::vector<std::vector<Regra::Condicao>> grupos =
UtilidadesRegras::agrupar(regra.condicoes, regra.conectores);

        for (int n = 0; n < grupos.size(); ++n) {
                for (int i = 0; i < grupos[n].size() - 1; ++i)
                {
```

```
                        for (int j = i + 1; j < grupos[n].size(); ++j) {
                                if (grupos[n][i].variavel == grupos[n][j].variavel &&
grupos[n][i].tendencia != grupos[n][j].tendencia) {
                                        return false;
                                }
                        }
                }
        }

        return true;
}

#include "regra_widget.h"

#include "enums.h"
#include "utilidades.h"

RegraWidget::RegraWidget(QWidget *parent)
        : QWidget(parent)
{
        ui.setupUi(this);

        connect(ui.pushButtonEditar,                                    static_cast<void
(QPushButton::*)(bool)>(&QPushButton::clicked),                                 this,
&RegraWidget::editarPressionado);
        connect(ui.pushButtonFechar,                                    static_cast<void
(QPushButton::*)(bool)>(&QPushButton::clicked),                                 this,
&RegraWidget::fecharPressionado);

}

RegraWidget::RegraWidget(Regra regra, QWidget *parent)
        : QWidget(parent)
{
        ui.setupUi(this);

        connect(ui.pushButtonEditar,                                    static_cast<void
(QPushButton::*)(bool)>(&QPushButton::clicked),                                 this,
&RegraWidget::editarPressionado);
```

```
	connect(ui.pushButtonFechar, static_cast<void (QPushButton::*)(bool)>(&QPushButton::clicked), this, &RegraWidget::fecharPressionado);

	ui.labelNome->setText(regra.nome);
	ui.labelTipo->setText(mapTipoRegraQStringGUI[regra.tipo]);

	QString condicoes;

	for (int i = 0; i < regra.condicoes.size(); ++i) {
		QString variavel = regra.condicoes[i].variavel.nome();

		condicoes += variavel + " tend. " + (mapTendenciaQStringGUI[regra.condicoes[i].tendencia]) + "\n";
	}

	ui.textCondicoes->append(condicoes);

	QString sugestoes;

	for (int i = 0; i < regra.sugestoes.size(); ++i) {
		QString variavel = regra.sugestoes[i].variavel.nome();

		sugestoes += variavel + " -> ";
		regra.sugestoes[i].porcentagem > 0 ? sugestoes += "+ " : sugestoes += "-";
		sugestoes += QString::number(regra.sugestoes[i].porcentagem) + "%\n";

	}

	ui.textSugestoes->append(sugestoes);
	ui.textTexto->append(regra.texto);
}

#include "base_de_regras.h"

#include <math.h>

#include <lib_clipsmm/clipsmm.h>
```

```
#include <string>

#include <vector>

#include "dlib/matrix.h"

#include "utilidades.h"

#include "utilidades_regras.h"

#include <fstream>

void BaseDeRegras::carregarRegras(QString regras)
{
	ofstream file;
	file.open("regras.temp");
	file << regras.toStdString();
	file.close();

	environment.reset();
	bool b = environment.load("regras.temp");
}

BaseDeRegras::Resultado BaseDeRegras::ativar(DadosV2 dados)
{
	//environment.reset();

	std::vector<Variavel> variaveis = dados.first;
	dlib::matrix<double> valores = dados.second;

	for (int i = 0; i < variaveis.size(); ++i) {
		dlib::matrix<double> coluna = dlib::colm(valores, i);

		std::vector<double> v(coluna.begin(), coluna.end());

		Tendencia t =
UtilidadesRegras::coxStuart(std::vector<double>(coluna.begin(), coluna.end()));
```

```
                environment.assert_fact("(variavel (nome " +
Utilidades::qStringToStdString(variaveis[i].nome()) + ") (tendencia " +
mapTendenciaString[t] + "))");
        }

        environment.run();

        CLIPS::Fact::pointer fact = environment.get_facts();

        BaseDeRegras::Resultado resultado;

        while (fact)
        {
                if (fact->get_template()->name() == "alerta") {
                        std::string tipo = fact->slot_value("tipo")[0];
                        std::vector<CLIPS::Value> variaveisSugeridas =
fact->slot_value("variaveis");
                        std::vector<CLIPS::Value> porcentagens =
fact->slot_value("porcentagens");
                        QString texto =
Utilidades::stdStringToQString(fact->slot_value("texto")[0]);

                        std::map <Variavel, std::pair<double, double>> valoresSugeridos;

                        for (int i = 0; i < variaveisSugeridas.size(); ++i) {

                                std::string var = variaveisSugeridas[i].as_string();

                                // definir a coluna na matrix referente a variavel
                                int n = 0;
                                while (var !=
Utilidades::qStringToStdString(variaveis[n].nome())) {
                                        ++n;
                                }
                                dlib::matrix<double> coluna = dlib::colm(valores, n);

                                // calcular o IC da coluna
```

```
                                std::pair<double, double> IC =
UtilidadesRegras::intervaloConfiancaComPorcentagem(std::vector<double>(coluna.beg
in(), coluna.end()), 0);

                                // descobrir o novo valor utilizando a sugestao
                                // e inserir nos valores sugeridos
                                // ATENCAO: VERIFICAR SE EH REALMENTE DESSA
FORMA
                                valoresSugeridos[variaveis[n]] = std::make_pair(IC.first +
IC.first * porcentagens[i].as_float() / 100.0, IC.second);;
                        }

                        // instanciar ConfiguracaoAlerta e inserir no resultado
                        resultado[mapStringTipoRegra[tipo]] =
ConfiguracaoAlerta(valoresSugeridos, texto);
                }

                fact = fact->next();
        }

        return resultado;
    //BaseDeRegras::Resultado resultado;

    //std::map <Variavel, std::pair<double, double>> valoresVariaveis;
    //QString texto;

    //valoresVariaveis[Variavel::_psb] = std::make_pair(5, 0.1);
    //valoresVariaveis[Variavel::_hsi] = std::make_pair(8, 0.2);

    //texto = "aumentar PSB e HSI";
    //resultado[TipoRegra::vibracao] = ConfiguracaoAlerta(valoresVariaveis, texto);

    //valoresVariaveis[Variavel::_rpm] = std::make_pair(4, 0.1);
    //valoresVariaveis[Variavel::_vazao] = std::make_pair(10, 0.3);
    //texto = "diminuir RPM e aumentar Vazao";

    //resultado[TipoRegra::enceramento] = ConfiguracaoAlerta(valoresVariaveis, texto);

    //resultado[TipoRegra::desgaste] = ConfiguracaoAlerta();
```

```
    //return resultado;
}

bool BaseDeRegras::testaExclusividadeMutua(const std::vector<Variavel> &v, QString regraClips)
{
	CLIPS::Environment environment2;

	QString str = "(deftemplate variavel (slot nome) (slot tendencia))";
	environment2.build(str.toStdString());
	environment2.build(regraClips.toStdString());

	std::vector<Tendencia> t(v.size());

	for (int i = 0; i < std::pow(3, t.size()); ++i) {
		environment2.reset();

		for (int n = 0; n < v.size(); ++n) {
			environment2.assert_fact("(variavel (nome " + Utilidades::qStringToStdString(v[n].nome()) + ") (tendencia " + mapTendenciaString[t[n]] + "))");
		}

		environment2.run();

		CLIPS::Fact::pointer fact = environment2.get_facts();

		while (fact)
		{
			if (fact->get_template()->name() == "RegraNaoExclusiva") {
				return false;
			}

			fact = fact->next();
		}

		UtilidadesRegras::somaVetorTendencia(t, 0);
	}
```

```
        return true;
}
```