PROCEDIMENTOS DE MANUTENÇÃO SIMPREBAL

Gerar Regras de Produção para o Simprebal (Incrementar a Base de Conhecimento)

O Simprebal é um sistema especialista baseado em regras de produção. Todo sistema especialista possui uma base de conhecimento na qual fica armazenado todo o conhecimento do sistema. Durante o processamento inteligente é essa base de conhecimento que auxilia o raciocínio do sistema inteligente. Estes arquivos são gerados em formato .txt (do bloco de notas) usando a linguagem clips e depois são salvos com a extensão .clp.

As regras de produção, no geral, seguem a estrutura abaixo:

Se teste de condição 1
Então
Ação X

Elas são formadas por um conjunto de estruturas Se...Então... . A linguagem clips segue a mesma idéia, porém com sintaxe diferente.

```
(defrule SIMPREBAL-ON  ;;Nome da regra
        (Tag{label == "_simprebaloff" && value == "0"}) ;;Se teste de condição
        =>      ;;então
        (assert (SIMPREBAL ON)) );;Ação X
```

Toda regra em clips necessita ter um nome para que se possa diferenciar uma regra da outra. No exemplo acima o nome da regra é SIMPREBAL-ON. Após definido o nome da regra inicia-se o teste de condição. No exemplo acima o teste de condição verifica se o label de nome simprebaloff recebe o valor 0. Isso significa que o label simprebaloff recebe o valor 0 que é equivalente a false, então pode-se concluir que se simprebaloff é falso, então simprebal on é verdadeiro. É exatamente isso que a parte referente a ação (assert (SIMPREBAL ON)) quer dizer. Assim a ação está definida e a regra encerrada. Cuide com o fechamento correto dos parênteses, pois um parêntese aberto e não fechado (ou o contrário) pode ocasionar um erro e a regra não ser executada.

No caso do Simprebal a base de conhecimento é composta por 16 arquivos do tipo .clp Estes são divididos de acordo com o número de DFIs que existe na Usina (são três DFIs para cada UGH): 1a, 1b, 1c, 2a, 2b, 2c e assim sucessivamente até as DFIs referentes a UGH 5. Existe ainda um arquivo chamado regras.clp que testa a conexão do Simprebal.

Comandos usados nas regras

Segue uma descrição dos comando em clips que foram usados na elaboração das regras do Simprebal.

defrule – usado para definir uma regra;
assert – usado para criar fatos;
verificação de fatos – (OPCSRV-ON "dfibalbina");
&& – operador booleano E. Testa a ocorrência de duas ou mais variáveis simultaneamente;
== – comparação de variáveis;
O exemplo abaixo mostra a utilização dos comandos apresentados:

```
(DFIDevice{status == "connected" && name == "dfi1a"})
```

printout – imprime um texto no console ou em uma variável definida. Nesse último caso esse comando manda uma mensagem para outra variável. Veja os exemplos abaixo. No primeiro exemplo o código em vermelho é a variável (no caso do Simprebal uma variável Java) para a qual o comando printout manda o texto “dfi1a”. O segundo exemplo segue a mesma idéia do primeiro, porém o texto é

dividido por #. Cada texto desse será armazenado num lugar específico do banco de dados, por isso a necessidade de usar o caracter # para dividir esses textos.

(printout guidfi1a "dfi1a")
(printout gui34 "sad-amarelo#mgg-amarelo#mcb-amarelo#")

( ) – os parentêses dividem os comandos dentro da regra;
{ } – são usados nas comparações;

? – esse comando indica que o valor que está contido na variável que deve ser considerado. Veja o exemplo abaixo. Esse trecho de código significa que a variável label vai receber o valor atual da viavél label sinalizado por ?label.

(label ?label)

> – comparado relacional. Maior que;
< – menor que;
>= – maior ou igual;
<= – menor ou igual;
|| – operador booleano OU.
!= – difereça entre variáveis;
“” – declaração de valores de variáveis ou texto a ser impresso pela variável printout;
=> – comando equivalente ao ‘Então” da estrutura apresenta anteriormente. Esse comando é seguido por uma ação, normalmente definida pelo comando assert.

Estrutura das regras

A estrutura defeinida para o desenvolvimento da base de regras do Simprebal, tem os seguintes níveis de organização do conhecimento. Vale ressaltar que a camada 1 (Aquisição de dados é feita pelo OPC e não está incluída na modelagem das regras):

Processamento de Sinal: Esta camada tem o objetivo de verificar o status do sinal fieldbus vindo dos devices. Caso a qualidade do sinal seja ruim, uma ordem de serviço é enviada para que se faça uma verificação no device. Se o status do sinal for bom, então o sistema prossegue a execução e avança para a segunda camada;

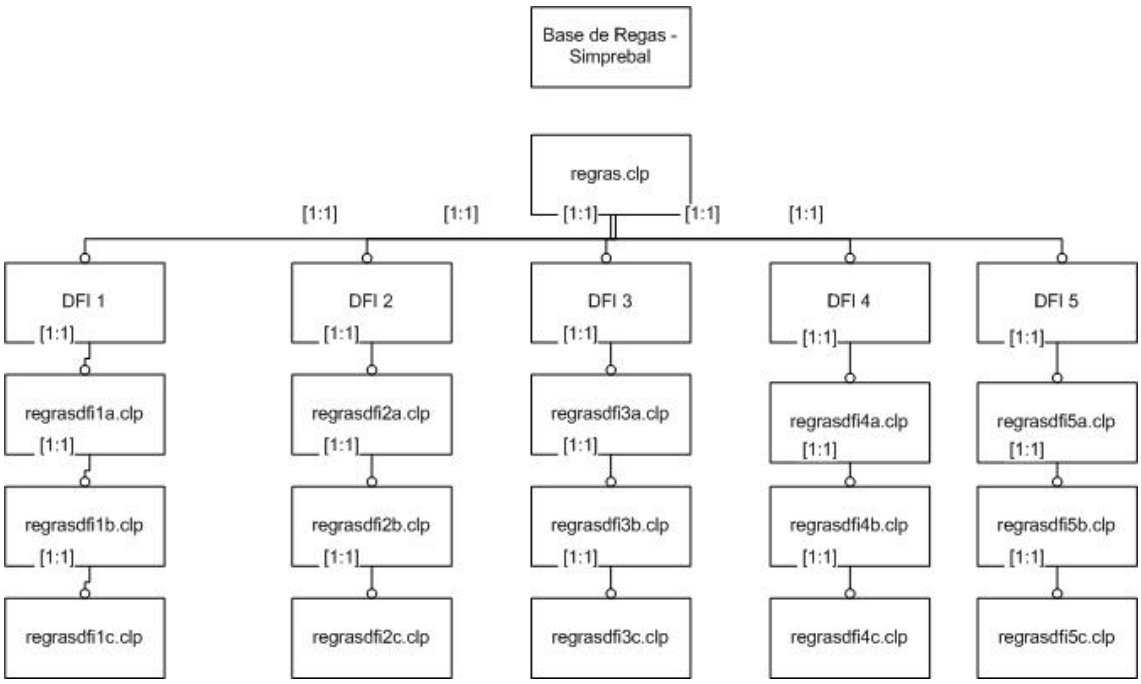
Monitoração de Condição: Essa camada tem a finalidade de monitorar a condição em que os equipamentos estão operando de acordo com a faixa de valores definida. As faixas de valores são divididas da seguinte forma: 1) faixa de alerta ação proativa dos sitema; 2) faixa de alarme definida pela Smar; e 3) valor de trip, também definido pela Smar . A saída dessa camada é o modo de falha do FMEA.

Diagnóstico: Essa camada recebe como entrada a saída da camada monitoração de condição efornece como saída informações como: a falha que ocorreu, causas prováveis da falha e o efeito dessa falha no equipamento).

Arquivo regras.clp

O arquivo regras.clp tem a finalidade de testar a comunicação do Simprebal com o servidor OPC. Se estiver conectado, o Simprebal testa a comunicação com as DFIs. Testa também se as tags estão mudando de valor a cada 10 minutos, se isso não estiver acontecendo significa que o servidor Simprebal está travado. Após esses testes e se tudo estiver funcionando corretamente, os outros arquivos de regras serão chamados.

O teste começa com o bloco de regras chamado “OPC Servers”, em seguida vem as regras referentes a conexão do Simprebal (bloco de regras chamado SIMPREBAL), e finalmente as regras referentes ao teste das DFIs, um bloco de regras para testar se está online e outro para testar se está offline para cada uma das 15 DFIs. A figura abaixo apresenta como os arquivos de regras estão organizados e a quantidade de arquivos que a base de conhecimento contém.



Arquivo regrasdfi1a.clp

O arquivo regrasdfi1a.clp faz o teste para todas as tags que estão presentes na DFI 1a. O primeiro bloco de regras testa o processamento de sinal OPC da DFI, o segundo bloco de regras testa o processamento de sinal Fieldbus da DFI. Depois da camada de processamento de sinal vem a camada de monitoração de condição que tem a finalidade de monitorar as faixas dos valores de operação das tags. Quatro valores são testados: um valor indicando a condição normal; um intervalo de valores indicando a condição alto/baixo, esse estado implica no estado de alerta do sistema e compreende uma faixa de valores próxima a faixa de alarme; o valor de alarme, que na verdade é um intervalo de valores; e o valor de trip. Os valores de alarme e trip são definidos com base nos manuais da Smar, os valores normal e de alerta foram definidos para o Simprebal e o valor de alerta tem a finalidade de permitir a ação pro-ativa do sistema.

Para fazer uma alteração na faixa de valores de uma regra basta procurar o label da tag no arquivo config.ini. Uma vez encontrado esse label procure pelo mesmo no aqruivo de regras e prossiga com a alteração:

```
(defrule UGH1-GEP-MonitoracaoCondicao-1
        (Tag {label == "g1.gep.t.enrol.a" && value != "" && value  <= "105"})
        =>
        (assert (condition-NORMAL ?label))
)
```

O label é a variável em vermelho, ele indica o nome que foi usado na regra para referenciar a tag real. O valor da tag encontra-se em azul na regra acima. Se for alterado um valor referente a condição NORMAL cuide para que as regras seguintes, referentes as condições HIGH/LOW, ALARM, TRIP, não percam a coerência.

Ainda na monitoração de condição, existe um conjunto de regras responsável por atribuir uma condição a tag de cada label usado nas regras descritas anteriormente. Veja a regra abaixo:

```
(defrule UGH1-GEP-MonitoracaoCondicao-23
        (signal-GOOD "g1.gep.st.t.enrol.a")
        (condition-NORMAL "g1.gep.t.enrol.a")
        =>
        (assert (condition G149G1A-normal))
)
```

A finalidade dessa regra é testar de forma combinada a condição do sinal (observe o label em vermelho na regra a cima), e a condição de operação (observe o label em azul na regra a cima). Se o sinal estiver bom o tag (em roxo na regra) recebe a condição definida, que pode ser normal, alto/baixo, alarme e trip.

Esse mesmo procedimento se repete para cada uma das tags que fazem parte dessa DFI e estão correlacionados com as faixas de valores definidas para os labels referentes as tags, conforme foi explicado mais acima.

Outra camada do arquivo de regras regrasdfi1a.clp é a camada de diagnóstico. Essa camada recebe como entrada a saída da camada monitoração de condição e fornece como saída informações como: a falha que ocorreu, causas prováveis da falha e o efeito dessa falha no equipamento). O primeiro diagnóstico realizado nessa camada está relacionado com a comunicação OPC, testa se todos os canais de comunicação estão conectados e caso ocorra alguma falha nesses canais de comunicação o sistema envia uma mensagem para o banco de dados. Veja regra baixo:

```
(defrule UGH1-OPC-Diagnostico-1
        (or (COM-BAD-0 "g1.srg.st.t.arfrio.rad1")
        (COM-BAD-0 "g1.srg.st.t.arfrio.rad2")
        (COM-BAD-0 "g1.srg.st.t.arfrio.rad3")
        (COM-BAD-0 "g1.srg.st.t.arfrio.rad4")
        (COM-BAD-0 "g1.srg.st.t.arfrio.rad5")
        (COM-BAD-0 "g1.srg.st.t.arfrio.rad6")
        (COM-BAD-0 "g1.srg.st.t.arfrio.rad7")
        (COM-BAD-0 "g1.srg.st.t.arfrio.rad8")
        (COM-BAD-0 "g1.srg.st.t.arquente")
        (COM-BAD-0 "g1.gep.st.t.enrol.a")
        (COM-BAD-0 "g1.gep.st.t.enrol.b")
        (COM-BAD-0 "g1.gep.st.t.enrol.v")
        (COM-BAD-0 "g1.gep.st.t.nucleo.a")
        (COM-BAD-0 "g1.gep.st.t.nucleo.b")
        (COM-BAD-0 "g1.gep.st.t.nucleo.v")
        (COM-BAD-0 "g1.mgg.st.t.metal.mguia.sup1")
        (COM-BAD-0 "g1.mgg.st.t.metal.mguia.sup2"))
        =>
        (assert (diagnostic G1A-canal1-0))
)
```

A regra testa o sinal OPC de todas as tags que estão contidas nesse canal de comunicação (veja os labels contidos na regra, em vermelho), se houver erro o sistema envia um código para o banco de dados (código em azul) indicando que ocorreu uma falha de comunicação no canal especificado, da UGH especificada.

Uma vez testadas as falhas nos canais de comunicação as regras começam a testar as falhas do sinal Fieldbus. Veja regra abaixo:

```
(defrule UGH1-GEP-Diagnostico-1
        (signal-BAD-0 "g1.gep.st.t.enrol.a")
        =>
        (printout gui14 "gep-amarelo#")
        (printout gui12 "G149G1ASIND0#electronics#")
)
```

O código em vermelho indica que a falha testada é de sinal. Para cada label existe cinco regras variando de 0 a 4. Se a falha ocorrer as regras mandam para o sistema uma mensagem (printout gui14 "gep-amarelo#") indicando que a janela referente ao equipamento deve piscar em amarelo para sinalizar a falha. As regras ainda enviam uma outra mensagem para o banco de dados, com o código da falha, em azul na regra acima, e qual a equipe responsável deve ser acionada por email, em roxo na regra acima.

O diagnótico das falhas nos equipamentos é feito seguindo as faixas de valores determinadas na camada de monitoração de condição. Então, existe uma regra na camada de diagnóstico para os estados de alerta, alarme e tripe (em vermelho nas regras) de cada tag. Veja regras abaixo:

```
(defrule UGH1-GEP-Diagnostico-31
        (condition G149G1A-alto)
    =>
        (printout gui14 "gep-amarelo#")
        (printout gui11 "G149G1AD1#operators#")
)

(defrule UGH1-GEP-Diagnostico-32
        (condition G149G1A-alarme)
```

```
    =>
        (printout gui14 "gep-vermelho#")
        (printout gui11 "G149G1AD2#electricians#")
)

(defrule UGH1-GEP-Diagnostico-33
        (condition G149G1A-trip)
    =>
        (printout gui14 "gep-vermelho#")
        (printout gui11 "G149G1AD3#electricians#")
)
```

Se a falha ocorrer as regras mandam para o sistema uma mensagem (printout gui14 ...) indicando que a janela referente ao equipamento deve piscar, na cor indicada na mensagem, para sinalizar a falha. As regras ainda enviam uma outra mensagem para o banco de dados, com o código da falha, em azul na regra acima, e qual a equipe responsável deve ser acionada por email, em roxo na regra acima.

Todos os procedimentos descritos para o arquivo regrasdfi1a.clp valem para todos os outros arquivos de regras de cada uma das DFIs.

Observação:
Padronização dos printouts: O comando *printout guiXY "texto"* envia uma mensagem para o código JAVA contendo o *"texto"*. Neste caso, *X* deverá ser um número de 1 a 5, correspondente ao número do unidade geradora para a qual a regra é destinada, e *Y* deverá ser o número 1, 2 ou 4, que possuem os respectivos significados:

1: A falha em questão é relativa a um equipamento da unidade geradora.
2: A falha em questão é relativa à instrumentação.
4: Indica uma informação interna para fazer o cliente piscar em amarelo ou vermelho, de acordo com o tipo de falha (ALERTA, ALARME, TRIP, STATUS BAD, QUALITY BAD, OFFLINE).

Arquivos serializáveis

Arquivo de falhas – o arquivo de falhas, denominado *FailureFile.txt*, presente na pasta *SimprebalServer,* é de fácil alteração. O usuário pode realizar alterações diretamente nesse arquivo sem precisar mexer no código fonte do sistema. Após realizada a alteração, basta salvar o arquivo e executar um outro arquivo que se chama *serialization.bat*. Esse procedimento atualiza as alterações realizadas.

Veja o trecho de código abaixo, ele faz parte do arquivo de falhas e tem um conjunto de códigos que será explicado em seguida:

G1ACH1D1#Falha na comunicação OPC com UGH-01A canal1-H1#2#DFI1A canal1 - Quality BAD#Erro na configuração do servidor OPC#1#4

Note que o trecho de código acima contém códigos separados pelo caractere #. O primeiro se refere ao código da falha, o segundo é a descrição da falha, o terceiro é o ID do equipamento conforme cadastrado no banco de dados. O quarto é modo de falha (tag - condição), o quinto é a causa da falha, o sexto o fator de detecção da falha, e finalmente o sétimo é a severidade da falha (estes dois últimos campos são valores referentes ao FMEA).

Arquivo de decisão – o arquivo de decisão, denominado *DecisionFile.txt*, presente na pasta *SimprebalClient*, também pode ser alterado independente do código fonte do sistema. As informações desse arquivo são comandos para o cliente apenas, e não estão armazenadas no banco de dados. Para atualizar esse arquivo é necessário que o usuário crie um código de falha, no arquivo de falha, e então utilize este código no arquivo de decisão, referente a nova falha inserida. As mensagens gravadas no arquivo de decisão seguem a forma abaixo:

G1OPCSRV#Falha ao conectar com o servidor OPC#Conectar o cabo Ethernet do servidor simprebal ao servidor OPC.

Note que o trecho de código acima contém códigos separados pelo caractere #. O primeiro se refere ao código da decisão, o segundo é a descrição da falha, o terceiro é a descrição da decisão.

Procedimento para adicionar uma nova tag ao arquivo de configuração

Para adicionar uma nova tag no arquivo config.ini tem que seguir o formato seguinte:

-labelvalue = DFI*UGH.SISTEMA.EQUIPAMENTO*TAGVALUE

-labelstatus = DFI*UGH.SISTEMA.EQUIPAMENTO*TAGSTATUS

A onde labelvalue é um mneumônico associado a uma determinada tag, que descreve o que essa tag significa, e o labelstatus é o mesmo do que o labelvalue mais uma letra st depois do segundo ponto (mneumônico para o status de uma tag).

exemplo

g1.srg.t.arfrio.rad1 = DFI1A*UGH1.GEP.SRG*126GAF1_AI1.PV.VALUE

g1.srg.st.t.arfrio.rad1 = DFI1A*UGH1.GEP.SRG*126GAF1_AI1.PV.STATUS

Para adicionar uma nova tag, tem que ter informação, a que DFI pertence esta tag, a que unidade geradora, sistema e equipamento, e é necessário que haja VALUE e STATUS.

Procedimento para criar uma nova regra

Procedimento para visualizar o disparo de uma nova regra no Cliente Simprebal

Procedimento de Backup e restauração do banco de dados