Framework para Aplicação de *Model Checking* ao Projeto de Automação no Ambiente Industrial

Thiago G. R. Cruz [***] **Antonio E. C. da Cunha** [****]

Programa de Pós Graduação em Engenharia Elétrica
Instituto Militar de Engenharia, Rio de Janeiro, RJ
[***] ***e-mail:*** *thiagoguerra@ime.eb.br*
[****] ***e-mail:****carrilho@ime.eb.br)*

Abstract: This paper proposes a framework for applying model checking in industrial control design based on the GRAFCET language. The aim of this research was to develop a step-by-step to help create a digital twin with a cyberphysical aspect of a system composed of a control program and a plant to be controlled. For this, it is proposed, through manual inspection, the creation of templates of the most common modules that compose the internal and external environment of the control. The creation of models in state machine was used for the environments and for the control itself. The application of model checking is done through the NuSMV tool, which has its own language based on state evolution with property verification in Computation Tree Logic - CTL. For illustration purposes, a example of a valve with all the components found in more recent real projects in the oil and gas industry is introduced. The result was a model capable of experimenting with execution paths and analyzing them by verifying the security and operating properties of the system. Additionally, a validation of the initially designed control program was successfully carried out, being able to serve, in a real application, as a complement to a factory acceptance test before its implementation in an embedded PLC.

Resumo: Este artigo propõe um *framework* para aplicação de *model checking* em projetos de controle industrial baseados na linguagem GRAFCET. Buscou-se nesta pesquisa o desenvolvimento de um passo a passo para auxiliar a criação de um gêmeo digital com aspecto ciber-físico de um sistema composto por programa de controle e planta a ser controlada. Para isto, propõe-se, através de inspeção manual, a criação de *templates* de módulos mais comuns que compõem o ambiente interno e externo ao controle. Utilizou-se a criação de modelos em máquinas de estados para os ambientes e para o controle em si. A aplicação do *model checking* é feita através da ferramenta NuSMV que tem uma linguagem própria baseada em evolução de estados com verificação de propriedades em lógica temporal CTL. Para fins de ilustração é apresentado um *running example* de uma válvula com todos os componentes encontrados em projetos reais mais recentes na indústria de petróleo e gás. Obteve-se como resultado um modelo capaz de experimentar caminhos de execução e analisá-los pela verificação de propriedades de segurança e de funcionamento do sistema. Adicionalmente executou-se com sucesso uma validação do programa de controle projetado inicialmente, podendo servir, numa aplicação real, como complemento de um teste de aceitação de fábrica antes de sua implementação em um CLP embarcado.

Keywords: CLP; process control; GRAFCET; model checking; CTL.
Palavras-chaves: CLP; controle de processo; GRAFCET; model checking; CTL.

1. INTRODUÇÃO

A ausência na prática de métodos formais para validação do programa de CLP de processo, por diversos motivos, é um problema que se apresenta por não se utilizar um sistema de verificação exaustiva. Isto decorre em limitação do teste de aceitação de fábrica -TAF (*Factory Acceptancy Test* - FAT) por não apresentar todas as possibilidades, ficando limitado à um número exíguo de cenários de acordo com a experiência do projetista e a prazos muito curtos (Hallan William Veiga, 2018). O uso da técnica *model checking* se apresenta como uma solução para aliviar este problema pois é capaz de verificar propriedades na busca de possíveis inconsistências no projeto. Há recurso para avaliação automática de caminhos de execução nesta técnica, além de uma ampla capacidade de cobertura a partir da análise de diversas sequências de eventos.

Existe uma crescente preocupação com a exigência de qualidade em projetos para programas de CLP utilizados em sistemas de controle industriais, principalmente em sistemas críticos que exigem alta disponibilidade e que por natureza são de grande complexidade. Torna-se importante levar em consideração o conhecimento de engenheiros e técnicos na programação de CLPs (de Souza, 2010),

tendo como base o uso das cinco linguagens de programação descritas na norma internacional IEC - 61131-3 (IEC, 2013b).

Na seção 2 são apresentados alguns conceitos teóricos sobre estrutura *kripke*, diagramas *GRAFCET*, *model checking* e lógica CTL. Nesta seção também são apresentadas algumas conclusões sobre o estudo de padrões de especificação de propriedades, os tipos de falhas encontradas durante a execução da pesquisa e as diferenças entre os conceitos de verificação e validação na identificação de erros e inconsistências de projeto. Na seção 3 define-se e propõe-se a criação de um ***framework*** para aplicação da técnica *model checking* no programa de controle que será embarcado em um CLP de processo. Nesta seção é apresentado, como um *running example*, o controle de uma válvula para ilustração do método. Por fim na seção 4 são apresentadas as conclusões e as perspectivas para trabalhos futuros.

2. PRELIMINARES

2.1 O diagrama de projeto GRAFCET e a estrutura kripke correlata

Uma linguagem gráfica baseada nas redes de Petri interpretadas é o GRAFCET (IEC, 2013a), cujo nome deriva de *graph* (significa gráfico em inglês) e AFCET -*Association Française pour la Cybernétique Economique et Technique* (Johnsson, 1999), uma associação francesa que financiou os estudos para chegar a esta linguagem. O conceito que gerou a criação do GRAFCET inicialmente foi para especificação formal e um método de realização de projeto para CLP (Rene David, 2005). Hoje em dia é conhecido como um acrônimo francês para gráfico de comando etapa-transição (*Graphe Fonctionnel de Commande, Étapes Transitions*) sendo atualmente utilizado na indústria por um grande número de projetistas de máquinas automáticas aplicado a diversos contextos tecnológicos (Elaine Marques, 2007).

Na parte superior da figura 1 é apresentado um exemplo de diagrama GRAFCET. Este possui ações em cada etapa que podem ser usadas para atualização de *status* de variáveis internas ou externas (como, por exemplo, estados de contadores e comandos em atuadores respectivamente).

As **etapas (E0, E1, E2)** do GRAFCET determinam **ações (A0,A1,A2,S0,S1,S2,R0,R1,R2,T1,T2)** de controle configurando saídas/sinais que atuam sobre o ambiente. As **transições (T1,T2,T3,T4)** são sensibilizadas por um conjunto de **condições de transição** formadas por expressões booleanas sendo as entradas/sinais de leitura **(A,B,C,D,E)** as informações sensíveis do ambiente.

De acordo com de Tarso Guerra Oliveira (2010) uma estrutura **kripke** é composta por: um conjunto de **estados**; um conjunto de **estados iniciais**; uma **relação entre estes estados**; e uma **associação entre cada estado e um conjunto de valores proposicionais**.

Na parte inferior da figura 1 está ilustrada uma estrutura **kripke** a partir do GRAFCET com um conjunto de **estados**: (E0,E1,E2); conjunto de **estados iniciais**: (E0); **Relação entre estes estados** (A,B,C,D,E); **Associação entre cada estado e um conjunto de valores proposicionais**: (A0,A1,A2,S0,S1,S2,R0,R1,R2,T1,T2).

A estrutura **kripke** utilizada neste trabalho guarda as características de uma máquina de estados finita (MEF), comumente utilizada em modelagem de controle para programação de controladores lógico programáveis (CLP). Deste ponto em diante, por simplicidade textual, será adotada a nomenclatura *máquina de estados (ME)*.

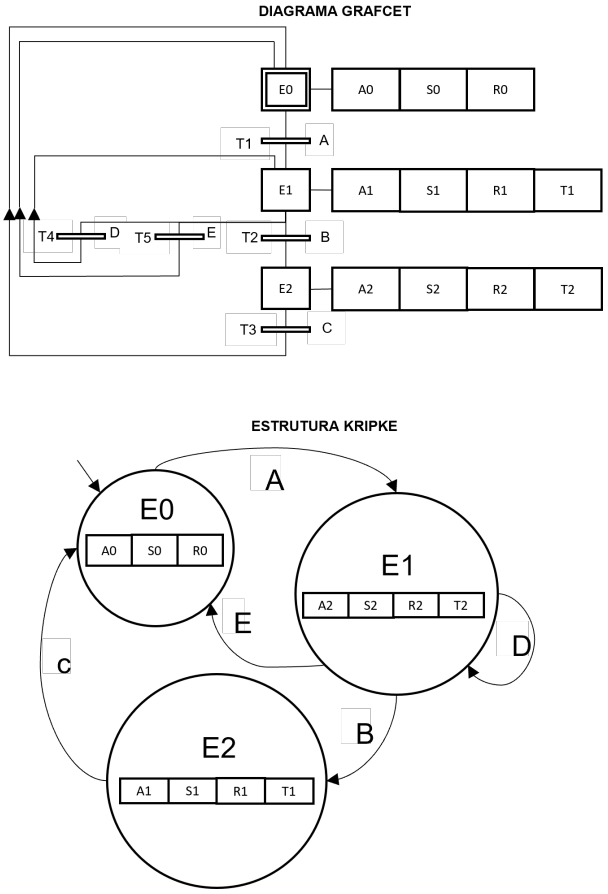

Figura 1. Diagrama GRAFCET e a estrutura *kripke*.

2.2 Model Checking

A técnica *model checking* é utilizada para verificação de sistemas concorrentes a estados finitos (Clarke Jr et al., 1999). Faz-se uma busca exaustiva no espaço de estados do sistema para determinar se uma dada especificação é atendida (Oliveira, 2006). Através da ferramenta de *model checking* são realizadas as **análises de propriedades** com uso de especificações de propriedades em alguma lógica temporal (Ferreira, 2005). No caso deste trabalho foi utilizada a ferramenta NuSMV e a lógica temporal CTL.

O *New Symbolic Model Verifier - NuSMV* (Cimatti et al., 1999) é uma ferramenta de verificação formal de especificações descritas em lógica temporal CTL sobre sistemas de estados finitos (Moreira, 2016). A linguagem de programação própria do NuSMV tem a sua sintaxe elaborada com o objetivo de reproduzir o comportamento de ME (Caldas, 2009).

2.3 A lógica de árvore da computação - CTL

A lógica temporal linear (Linear Temporal Logic - LTL) avalia uma proposição através do uso de operadores de

tempo (ou estado). Os operadores de tempo mais comuns são: **X**, na próxima vez (*next time*); **U**, até que (*until*), *a* **U** *b*, a propriedade *a* vale até valer *b*, onde *b* é garantido valer no futuro; **G**, globalmente ou em todos os estados (*globally*); **F**, no futuro (*in the future*). A lógica de árvore computacional (Computation Tree Logic - CTL), diferente da LTL, tem a capacidade de avaliar todas as possibilidades de caminhos diferentes pela análise de ramificações. Para isto utiliza operadores de caminho associados a operadores de tempo. Os operadores de caminho são: **E**, *existe* um caminho; **A**, para todos os caminhos (*always*). A combinação destes operadores serve como tradução de diversas especificações de funcionamento de sistemas, chamadas de **propriedades**, pelo estudo da evolução de estados do sistema modelado. Como por exemplo as fórmulas **EF**(p) e **AF**(p) que avaliam se p eventualmente é verdade no futuro e se é verdade em todos os caminhos no futuro respectivamente. Além de outras combinações como **EG**(p) e **AG**(p) (Globalmente verdade em um caminho e em todos os caminhos respectivamente).

2.4 Considerações sobre padrões de especificação

Segundo Dwyer et al. (1998), a transformação da especificação informal (textual) para a formal (com formalismo matemático), tem sido um grande desafio para a verificação de estados finitos como o *model checking* utilizando verificador de modelos simbólicos (Symbolic Model Verifier - SMV) na prática. Pode ser visto nos trabalhos de Dwyer et al. um esforço feito para encontrar padrões de especificações formais (Dwyer et al., 1999) (Dwyer et al., 1998). Foram estabelecidas algumas formas padronizadas de especificar propriedades mais comuns, de forma que os usuários da técnica de verificação *model checking*, por exemplo, não precisem se preocupar com o aprendizado aprofundado de formalismos baseados em estados como a lógica CTL. Um resumo pode ser visto em Silva (2008), onde os padrões utilizados são organizados em uma hierarquia conforme certos intervalos de eventos. Foi possível perceber que os padrões **existência (AF(p))** e **universalidade (AG(p))** podem ser aplicados à **verificação** e os padrões **resposta** (**AG(p -> AF(s))** - o evento **s** sempre ocorre após **p** ocorrer) e **precedência** (**(!E[!s U (p & !s)])** - o evento **p** ocorre somente após ter ocorrido **s**) podem ser aplicados à **validação**.

Na literatura também são encontrados certos padrões **gerais** ou **estruturais** que podem ser vistos resumidamente em Oliveira (2006). Estas propriedades especificadas em lógica CTL podem ser vistas em maiores detalhes no trabalho de Clarke Jr et al. (1999) que são: **Segurança - AG(p)** onde p é sempre válida para o sistema ou sendo usada da forma **AG(!p)** informalmente significando nada bom/ruim vai acontecer; **Vivacidade - AF(p)** onde p sempre será verdade no futuro ou informalmente significando algo bom/ruim vai acontecer; **Alcançabilidade - EF(p)** eventualmente p será verdade no futuro ou informalmente significando algo bom/ruim pode acontecer.

Foi possível perceber que alguns padrões de propriedades, sem avaliação de intervalos de eventos de Dwyer et al. (1999), coincidem com algumas propriedades de Clarke Jr et al. (1999). Os padrões denominados **universalidade** (**AG(p)**) e **ausência** (**AG(!p)**) coincidem com a propriedade de **segurança** e, o padrão **existência AF(p)** coincide com a propriedade **vivacidade**.

2.5 Identificação de falhas do sistema

Foram identificados durante a pesquisa três tipos de falhas, sendo duas específicas (detectável pelo equipamento com sensores dedicados ou pela comunicação com o CLP) e uma genérica (detectável pela temporização da etapa do controle). Nesta pesquisa estão sendo abordados todos os tipos de falhas citados, sendo mais comum a falha específica causada pela falta de comunicação com o CLP.

Foi considerada uma falha específica aquela que possui causa já conhecida com identificação por detecção do equipamento ou do controle. Já a falha genérica foi considerada aquela que não é identificada diretamente por medição ou monitoramento dedicado. Um exemplo de falha genérica que pode ser citada é a não confirmação do fim de curso da válvula por desacoplamento do eixo com o motor da válvula, após uma contagem de tempo pré determinado para aquela movimentação de abrir ou fechar, o sistema entende que passou muito tempo sem confirmação de abertura ou fechamento e que ocorreu um problema, informando falha genérica captada pela temporização da etapa do controle. Conforme poderá ser visto nas seções subsequentes, a temporização é abstraída no modelo de evolução de estados considerado-se o não determinismo da passagem do tempo. Desenvolveu-se um modelo com os estados de um temporizador sem levar em consideração a passagem do tempo em si.

2.6 Diferenciação entre verificação e validação e seu uso na identificação de erros de modelagem e projeto

Conforme pode ser visto na norma IEEE 24765:2017 (ISO, 2017) a definição de **verificação** pode ser entendida como o processo de avaliação de conformidade de um determinado sistema a requisitos de funcionamento bem conhecidos. De forma semelhante ao que foi proposto por Gonçalves (2021) pode-se fazer a pergunta: O modelo está bem construído?

De acordo com a norma IEEE 1012:2016 (IEE, 2016) entende-se como **validação** o processo de prover evidências de que um sistema resolve o problema apresentado de forma correta satisfazendo o uso pretendido e as necessidades dos usuários. Pode-se fazer a seguinte pergunta: O controle faz o que se espera?

O conjunto de técnicas de verificação automática chamada ***model checking***, que é traduzido para o português como **"verificação de modelos"**, consiste em analisar o código buscando o cumprimento de especificações de propriedades (Ferreira, 2005). Neste trabalho foi escolhido manter o termo em inglês para não causar confusão com o conceito de **"verificação"** utilizado para análise quanto ao funcionamento correto do sistema modelado, se diferenciando do conceito de **"validação"** que busca avaliar se a solução implementada atende aos requisitos de projeto. No contexto deste artigo, tanto a **verificação** quanto a **validação** são realizado durante a execução do ***model checking***.

3. PROPOSTA DE *FRAMEWORK* PARA APLICAÇÃO DE *MODEL CHECKING*

Propõe-se portanto um **framework** para aplicação de *model checking* com a utilização, para efeito de ilustração, do exemplo do projeto de controle de uma válvula.

3.1 Visão Geral do framework proposto

A visão geral e detalhada do *framework* pode ser vista na figura 2.

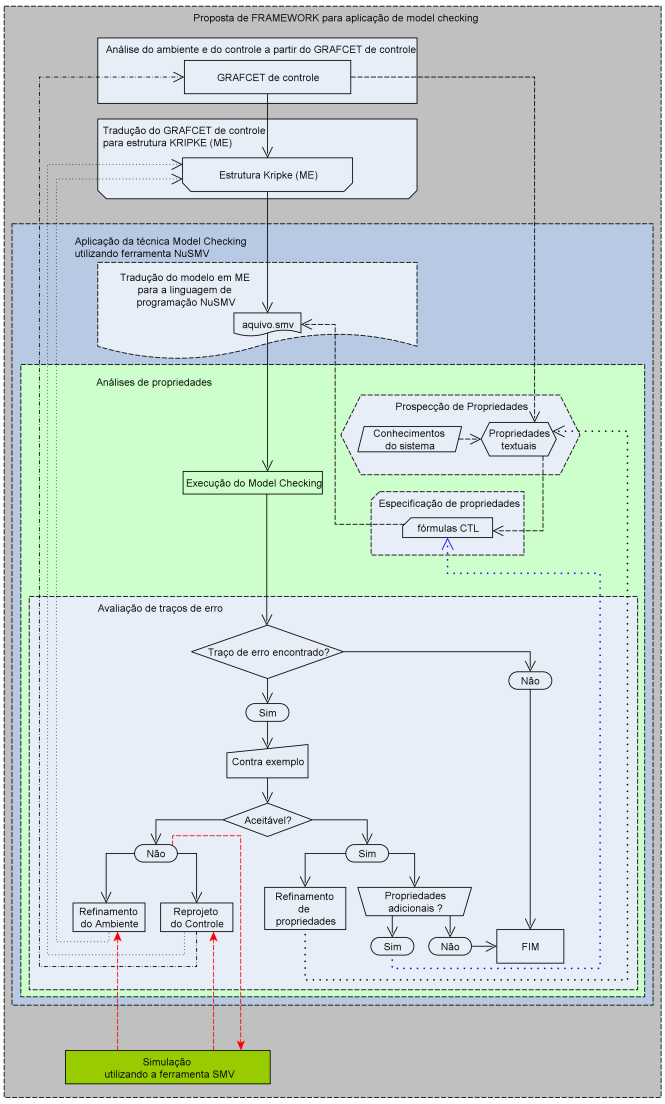

Figura 2. Visão geral da proposta de ***framework*** para aplicação de *model checking*.

A modelagem do exemplo da válvula foi preparada pensando-se num projeto de controle de processo existente na indústria de petróleo e gás, guardando as mesmas características que são implementadas no caso real. O intuito é de facilitar a adaptação dos conceitos e módulos criados na modelagem de outros sistemas mais complexos contendo o comportamento do programa e dos ambientes com os quais este interage. Ao longo do restante do texto será explorada cada etapa deste *framework*.

3.2 Análise do ambiente e do controle a partir do GRAFCET de controle

O primeiro procedimento é analisar o ambiente e o controle a partir do GRAFCET de controle. O projeto de automação na indústria de petróleo e gás tem sido realizado recentemente com o uso do diagrama GRAFCET e a linguagem SFC para a estruturação e implementação do código embarcado no CLP. A partir deste documento de projeto em GRAFCET é possível realizar uma verificação detalhada do funcionamento da lógica de automação aliada ao conhecimento da planta que está sendo automatizada. A figura 3 apresenta um projeto de controle de uma válvula realizado no diagrama GRAFCET. Observa-se que este projeto é composto por vários elementos que serão explicados nas etapas seguintes do *framework*. Basicamente o projeto consiste no controle de abrir e fechar uma válvula com indicação de estados aberta, fechada e falha, e comandos de abrir e fechar.

É possível através do **conhecimento do sistema** e da análise do **diagrama GRAFCET de controle**, encontrar quais são os componentes físicos e cibernéticos deste sistema. Esta diferenciação é essencial para que seja feita uma modelagem correta. Geralmente algumas tabelas são utilizadas como forma de complemento de informações sobre as etapas e transições.

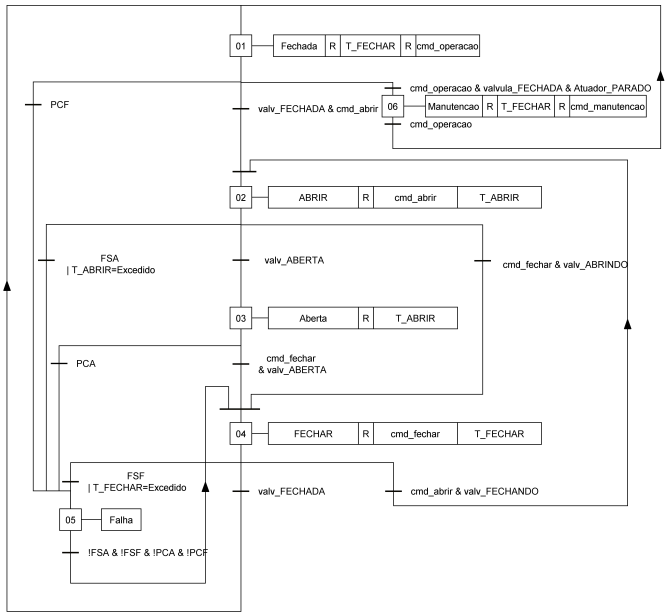

Figura 3. Exemplo de projeto de automação do controle de uma válvula elaborado com diagrama GRAFCET

*3.3 Tradução do GRAFCET de controle para a estrutura **kripke***

Nesta etapa, a partir da documentação de projeto e do conhecimento do funcionamento geral da planta, modela-se o sistema realizando a tradução do GRAFCET de controle para a estrutura **kripke** (ou ME). Juntamente com o controle também é elaborada nesta etapa a modelagem do ambiente interno e externo ao controle. Durante o projeto de elaboração da automação de processo um documento é gerado incluindo tabelas descrevendo etapas e transições

do GRAFCET com detalhes dos componentes da planta que não aparecem diretamente no diagrama GRAFCET, porém podem ajudar numa modelagem mais completa em ME com o ganho futuro de facilitar a criação de propriedades para uso do *model checking*.

São construídas então as ME dos módulos identificados na etapa anterior. As conexões entre os módulos são mostradas na figura 4, bem como as conexões de entrada e saída de dados entre eles. Esta etapa é necessária para que seja traduzida a estrutura de automação projetada e resumida no diagrama GRAFCET para a linguagem que será inserida na ferramenta de *model checking* que é baseada em ME. Uma vez esta tradução sendo feita um programa poderá analisar o sistema de acordo com uma lógica temporal que faça a busca por inconsistências fazendo uma *verificação* de propriedades. Nesta etapa então serão concentrados esforços de reproduzir com maior fidelidade possível (porém com a mínima complexidade necessária) o comportamento da lógica de controle e do ambiente em que o sistema a ser controlado está imerso.

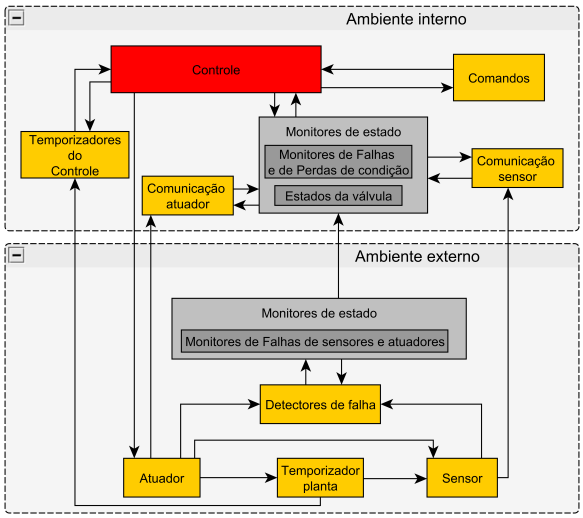

Figura 4. Módulos e conexões com ambiente interno e externo.

As ME encontradas no exemplo da válvula para o ambiente interno estão descritas na figura 4. Por questões de limitação de espaço serão mostradas somente algumas ME para ilustração. A ME do **programa de controle** (figura 5) é interpretada de forma que os seus estados correspondam às ações principais das próprias etapas do GRAFCET de controle. A **comunicação** é um módulo que possui 2 estados que indicam se os equipamentos estão se comunicando ou não com o CLP. Os **sensores** (6) possuem uma quantidade de estados que depende da aplicação, sendo interpretados pela análise de documentos complementares do diagrama de controle em GRAFCET. Um olhar a partir do conhecimento do sistema pode identificar com mais clareza o papel do sensor, uma vez que documentos complementares são fontes de informação sobre a planta. O **atuador**, de forma semelhante ao sensor, também pode ser interpretado a partir do projeto em GRAFCET e documentos complementares, no exemplo da válvula foram utilizados 3 estados: trânsito abrir (**TABRIR**), trânsito fechar (**TFECHAR**) e **PARADO**. Os **monitores de estados** (figura 8) não são ME, mas sim um conjunto de variáveis combinacionais que variam conforme os estados das ME anteriores, sendo utilizados nas transições de estados em etapas específicas do programa de controle. Os **temporizadores do controle** (figura 7) podem ser interpretados como sendo abstrações de eventos discretos a partir dos estados **Parado**, **Contando** e **Excedido**. Eles podem ser inicializados como uma ação em um estado do controle e *"resetados"* no estado seguinte que se espera evoluir este controle. Já os **temporizadores da válvula** contém abstrações da contagem das transições da válvula (abrir e fechar) com estados **Parado**, **Início** e **Fim** da contagem para tornar mais realístico o comportamento dos sensores que podem aguardar esta contagem antes de mudarem suas posições no modelo. Os comandos são abstraídos como advindos de um sistema supervisório e tratados para serem aceitos no programa de controle conforme a evolução de estados deste. Como o interesse neste caso está nos eventos discretos, o modelo desenvolvido para realizar esta tarefa está baseado somente nos eventos de comando ativado ou desativado.

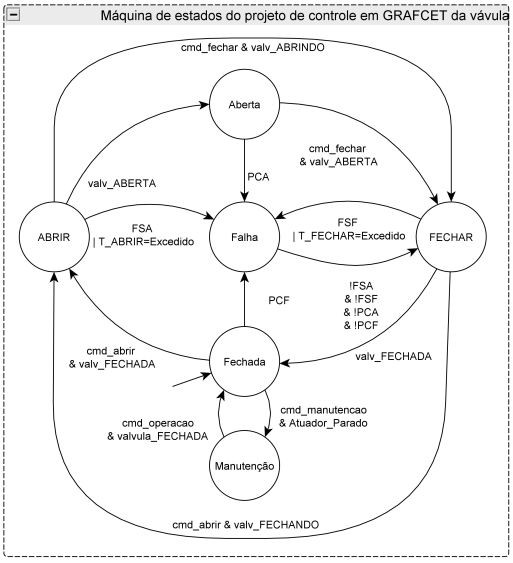

Figura 5. Máquina de estados do projeto de controle em GRAFCET da válvula.

No caso do projeto de controle da válvula sendo utilizado, não há ativação de várias etapas simultaneamente. Há a possibilidade de ativação múltipla no caso de projetos que envolvam diversos processos simultâneos. Nestes casos o GRAFCET geral pode ser subdividido em várias ME, uma para cada ramo paralelo (figura 9).

3.4 Aplicação da técnica model checking

Neste momento aplica-se a técnica *model checking* utilizando alguma ferramenta de verificação de modelos. Ocorre então a **tradução do modelo em ME para a linguagem de programação específica**. Se faz necessária a criação de variáveis associadas a **componentes** do sistema na linguagem NuSMV.

As variáveis do módulo principal ***MODULE main*** do programa NuSMV são tais que seguem a identificação realizada na modelagem em **ME** da etapa anterior, sendo assim criados os elementos do sistema modelado utilizando-

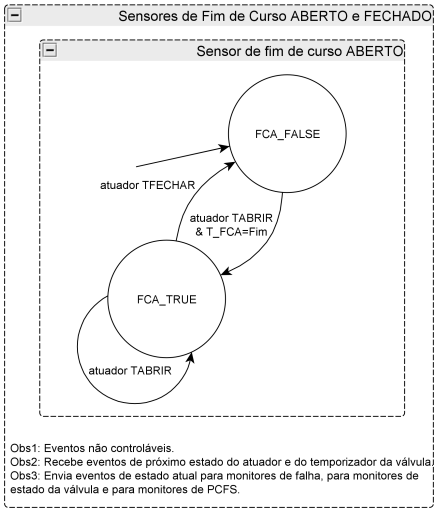

Figura 6. Sensor de fim de curso ABERTO.

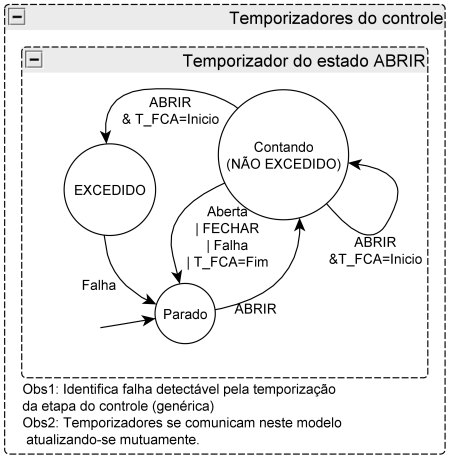

Figura 7. Temporizador do estado ABRIR.

se a declaração **VAR** do módulo principal do programa com reutilização dos módulos já criados. Estes módulos, que por convenção estão em **caixa alta**, possuem parâmetros de entrada que são descritos entre parêntesis. Uma variável declarada pela reutilização de um módulo guarda as características deste, podendo ser reutilizada como parâmetro de entrada em outras declarações. A declaração **DEFINE** é utilizada para criar uma variável utilizada nos monitores de estado, podendo combinar estados de vários módulos diferentes utilizando os operadores booleanos **&**,**|** e **!** (**AND**, **OR** e **NOT** respectivamente).

O código abaixo serve como ilustração e guarda este formato descrito acima, servindo como um modelo de como chamar módulos reutilizáveis no módulo principal do programa.

```
MODULE main
VAR
        moduloA: NOMEDOMODULO(moduloB.status=X1,
            moduloC.status=X2,...,moduloN.status=
            Xn);
DEFINE
        variavel:=(moduloE.status=X4 & moduloF.
            status=X5);
```

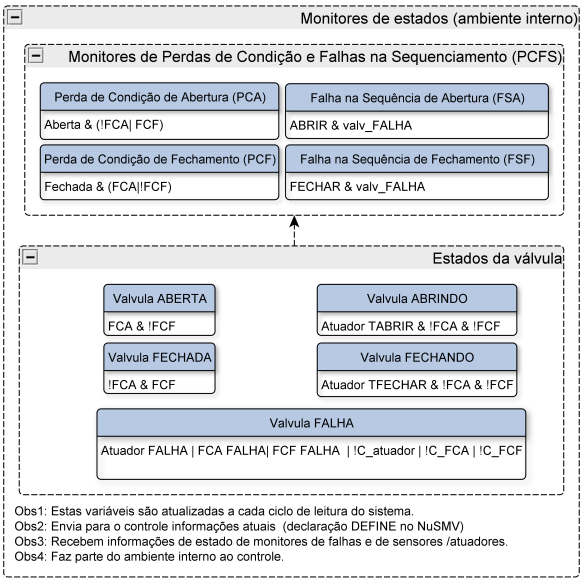

Figura 8. Variáveis combinacionais como monitores de estados (ambiente interno).

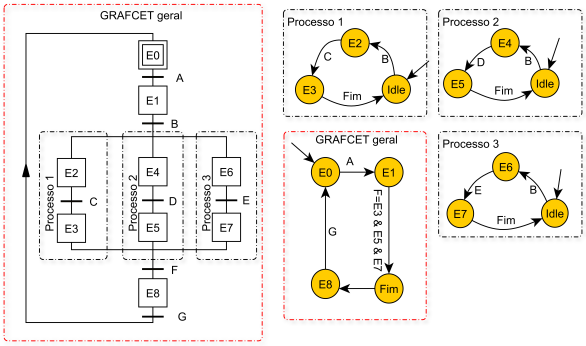

Figura 9. Modelagem em ME de etapas simultâneas no GRAFCET.

Como ilustração, segue somente um trecho do código do exemplo da válvula abaixo. **FCA** e **FCF** significa sensor de fim de curso de abertura e de fechamento respectivamente. O estado do atuador **TFECHAR** significa *em trânsito para fechar*. O módulo **T_FCA** significa temporizador para atuar o fim de curso de abertura.

```
MODULE main
VAR
    --sensor de Fim de Curso de Abertura
        FCA: SENSORFCA (atuador.status=TFECHAR,
            atuador.status=TABRIR,T_FCA.status=
            Fim);
(...)
DEFINE
        --monitor de estados da valvula
        valv_ABERTA:=(FCA.status=ATIVADO & FCF.
            status=DESATIVADO);
(...)
```

A declaração de um módulo é feita através da *keyword* **MODULE** seguido do seu nome e de parâmetros de entrada entre parêntesis. Estes parâmetros de entrada correspondem àqueles inseridos entre parêntesis na declaração de variáveis dentro do módulo principal.

Na declaração **VAR** de cada módulo são criadas **variáveis de estado** que podem ser de vários tipos. Serão utilizados tipos **booleanos** e ***strings***. Foi estabelecido um padrão para as variáveis de estado sendo nomeadas ***status***. Cada variável de estado corresponde a um estado da **MEF** correspondente.

A seção ***ASSIGN*** declara o estado inicial com ***init*** (atribuição de estado inicial) e as relações de transição de estados são inseridos em ***next*** (atribuição de próximo estado). As expressões booleanas usadas em ***next*** são as condições inseridas nas transições modeladas em **MEF**, por exemplo:

variavel_de_entrada1 & !variavel_de_entrada2

Pode-se ir de um estado para vários estados, bastando inserir estes estados seguintes entre chaves após ***":"***. Ao final o texto ***"TRUE:status;"*** significa que em qualquer outra condição o estado permanece o mesmo. A declaração de ***next*** termina com o texto ***esac;***. O código abaixo procura mostrar de forma resumida e genérica o formato proposto neste parágrafo e serve de exemplo para a aplicação no *framework*.

```
MODULE NOMEDOMODULO (variavel_de_entrada1,
    variavel_de_entrada2,...,
    variavel_de_entradan)

VAR
        status: {estado1,estado2,estado3...};

ASSIGN
        init(status):=estado1;
        next(status):=case
                status=estado1 & expressao1:
                    estado2;
                status=estado2 & expressao2: {
                    estado3, estado4};
                status=estadon & expressaon:
                    estado1;
                TRUE:status;
        esac;
```

Como ilustração da tradução das ME para o código NuSMV, segue o módulo **sensor de fim de curso de abertura** abaixo.

```
MODULE SENSORFCA (atuador_tfechar,atuador_tabrir,
    temp)
VAR
        status: boolean;
ASSIGN
        init(status):=FALSE;
        next(status):=case
                status=FALSE & atuador_tabrir &
                    temp: TRUE;
                status=FALSE & atuador_tfechar:
                    FALSE;
                status=TRUE & atuador_tfechar:
                    FALSE;
                status=TRUE & atuador_tabrir:
                    TRUE;
                TRUE:status;
        esac;
```

3.5 Prospecção e especificação de propriedades

Em seguida faz-se uma prospecção de propriedades do sistema para avaliação do projeto quanto ao funcionamento esperado e a aderências aos requisitos do projeto. Nesta etapa são descritas **textualmente** as propriedades que serão avaliadas. As propriedades são encontradas através do **conhecimento do sistema** e do entendimento do **funcionamento do GRAFCET** analisado. Em seguida é feita a especificação de propriedades sendo **escritas em lógica CTL**. Abaixo seguem algumas propriedades aplicadas ao exemplo da válvula estudado nesta seção.

Exemplo 1 (Verificação):
Textual: O controle terá o estado E1 ativado em pelo menos um caminho no futuro.
Em CTL: $EF\,(controle = E1)$
Exemplo 2 (Verificação):
Textual: O sensor de abertura da válvula não pode atuar junto com o sensor de fechamento.
Em CTL: $AG!(FCA = TRUE\ \&\ FCF = TRUE)$
Exemplo 3 (Validação):
Textual: A válvula somente abre após ser dado o comando de abertura?
Em CTL: $!E[valvula = ABERTA\ U(comando\_abrir = TRUE\ \&\ !(valvula = ABERTA))]$
Exemplo 4 (Validação):
Textual: Tendo o controle ido para o estado ABRIR a válvula efetivamente abre?
Em CTL: $AG(controle = ABRIR -> AF(valvula = ABERTA))$

É possível que um erro de modelagem do ambiente ou mesmo de tradução do controle interfira no resultado da análise da especificação. Portanto primeiro executa-se a **verificação** tanto do ambiente quanto do controle para primeiro corrigir os erros de construção do modelo, seja para verificar se a modelagem do controle está bem feita ou para verificar se há comportamento realístico de componentes físicos.

Após correções de erros por verificação é possível avaliar o correto comportamento do controle quanto ao seu funcionamento realizando portanto a **validação** posteriormente.

Para algumas verificações prévias do correto funcionamento do modelo do controle e do ambiente foram utilizadas algumas *propriedades estruturais*. Como a **vivacidade AF(p)** que se mostrou mais abrangente que a propriedade **alcançabilidade (EF(p))**. Dizer que "a válvula vai abrir"ou "em todos os caminhos a válvula abre"é mais abrangente que dizer que "a válvula eventualmente abre"ou em outras palavras "a válvula vai abrir pelo menos em um caminho". Utilizou-se também a combinação de segurança e vivacidade **AG(AF(p))** ("A válvula sempre abrirá frequentemente no futuro") sendo esta combinação ainda mais abrangente.

3.6 Avaliação de traços de erro

Por último faz-se a avaliação de traços de erro em que, uma vez a propriedade sendo atendida, há o retorno da especificação como verdadeira. Quando a propriedade não é atendida a especificação retorna FALSA e é apresentado um contra exemplo com um traço de erro indicando um problema. Neste momento então é concluída a análise da causa do problema, podendo ser refeita a modelagem ou a especificação. No caso de uma alteração no modelo do controle, haverá a necessidade de alteração do projeto em

GRAFCET. Se houver necessidade de alterações, no ambiente ou nas especificações, haverá necessidade somente de refinamentos do ambiente para ajustar melhor o modelo de forma a representar melhor o sistema. Pode ocorrer de haver retorno de especificações falsas aceitáveis, sendo complementadas ou refinadas com outras especificações para garantia de que não haverá um comportamento não desejado.

Concluiu-se a aplicação do método proposto no exemplo da válvula mostrado nesta seção após terem sido realizadas várias verificações e validações. Houve retorno de propriedades VERDADEIRAS e FALSAS aceitáveis.

Alguns contraexemplos podem ser de difícil interpretação, ou não indicar bem o problema que está ocorrendo, então adicionalmente há o recurso de simulação que pode ser utilizado em caminhos específicos nos quais se deseja explorar o comportamento. As simulações podem fornecer informações necessárias para realização de correções do controle ou do ambiente.

4. CONCLUSÃO

O *framework* proposto serviu adequadamente como um guia para a criação de um modelo capaz de representar um sistema composto por programa de controle e planta a ser controlada com seus respectivos ambientes. Para o ambiente interno foram criados módulos que descrevem o comportamento de componentes dentro do CLP além do programa de controle, como temporizadores, gerador de comandos, comunicação e variáveis combinacionais utilizadas para decisão de determinadas ações do controle sobre a planta. Para o ambiente externo, além da planta formada por componentes físicos (atuador e sensor), foram também criados módulos cibernéticos como detectores de falha, temporizadores da planta e monitores de falha. A partir da análise dos estados destes módulos foi possível realizar as verificações de propriedades, aplicando o *model checking* com uso da lógica CTL, executando uma validação do programa inicialmente projetado em GRAFCET. O *framework* proposto foi elaborado atendendo necessidades de modularidade e escalabilidade com o intuito de ser utilizado em aplicações mais complexas em programas de CLP para controle de processos industriais. Destina-se a servir, numa aplicação real, como complemento de um teste de aceitação de fábrica antes da implementação em um CLP embarcado. Foi também possível revisar e categorizar padrões de especificação de propriedades em lógica temporal CTL, encontrando padrões específicos para realização de verificação e validação de sistemas.

Como perspectiva para trabalhos futuros sugere-se a criação de um artefato capaz de gerar, de forma automática, os modelos em ME bem como traduzi-las para a linguagem NuSMV. Esta ferramenta computacional permitiria visualização de animação gráfica dos estados sendo marcados a partir de traços de erro e simulações obtidas na ferramenta NuSMV.

AGRADECIMENTOS

Gratidão ao consultor André Marino que deu contribuições valiosas para a confecção deste trabalho.

REFERÊNCIAS

(2013a). *IEC 60848 - GRAFCET specification language for sequential function charts*. International Electrotechnical Commission, Genebra, Suíça.

(2013b). *IEC 61131-3 - Programmable controllers Part 3: Programming languages*. International Electrotechnical Commission, Genebra, Suíça.

(2016). *IEEE 1012 - Standard for System, Software, and Hardware Verification and Validation*. Institute of Electrical and Electronics Engineers, New York, USA.

(2017). *IEC 24765 - Systems and software engineering Vocabulary*. International Electrotechnical Commission, Genebra, Suíça.

Caldas, R.B. (2009). *Modelagem, verificação formal e codificação de sistemas reativos autônomos*. Ph.D. thesis, Universidade Federal de Minas Gerais, Minas Gerais.

Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M. (1999). Nusmv: A new symbolic model verifier. In *International conference on computer aided verification*, 495–499. Springer.

Clarke Jr, E.M., Grumberg, O., Kroening, D., Peled, D., and Veith, H. (1999). *Model checking*. The MIT Press, Cabridge, Massachusetts, 1st edition.

de Souza, M.F. (2010). Modelagem e verificação de programas de clp escritos em diagrama ladder.

de Tarso Guerra Oliveira, P. (2010). Revisão de modelos ctl. Dezembro de 2010.

Dwyer, M.B., Avrunin, G.S., and Corbett, J.C. (1998). Property specification patterns for finite-state verification. In *Proceedings of the second workshop on Formal methods in software practice*, 7–15.

Dwyer, M.B., Avrunin, G.S., and Corbett, J.C. (1999). Patterns in property specifications for finite-state verification. In *Proceedings of the 21st international conference on Software engineering*, 411–420.

Elaine Marques, Nilson Junior, B.F. (2007). Especificação de sistemas automatizados utilizando o grafcet.

Ferreira, N.F.G. (2005). *Verificação formal de sistemas modelados em estados finitos*. Ph.D. thesis, Universidade de São Paulo.

Gonçalves, M.V. (2021). Escalonamento de caminhões-tanque em uma distribuidora de combustíveis utilizando controle preditivo baseado em modelo max-plus.

Hallan William Veiga, o. (2018). Método para teste automatizado de sistemas instrumentados de segurança em plataformas de petróleo.

Johnsson, C. (1999). *A Graphical Language for Batch Control*. Department of Automatic Control Lund Institute of Technology, Lund, Suécia.

Moreira, A.L.G.S.A. (2016). Modelagem e verificação automática de um protocolo de controle de fluxo adaptativo usando traços de execução.

Oliveira, C.H.C. (2006). *Verificaçao de modelos aplicada ao projeto de sistemas industriais automatizados por controladores lógicos programáveis*. Ph.D. thesis, Dissertação de Mestrado. Instituto Militar de Engenharia, Rio de Janeiro–RJ.

Rene David, H.A. (2005). *Discrete, Continuous, and Hybrid Petri Nets*. Springer Berlin Heidelberg New York, Germany.

Silva, A.M. (2008). *Aplicaçao de verificaçao de modelos a programas de clp: Explorando a temporizaçao*. Ph.D. thesis, Instituto Militar de Engenharia.