MODELAGEM E VALIDAÇÃO DE CONTROLADORES INDUSTRIAIS USANDO UML/STATECHARTS

Raimundo Santos Moura*, Felipe Cesar Alves do Couto†, Luiz Affonso Guedes†

* *Universidade Federal do Piauí (UFPI) Teresina, PI, Brasil*

† *Universidade Federal do Rio Grande do Norte (UFRN) Natal, RN, Brasil*

Emails: rmoura@dca.ufrn.br, felipecouto@dca.ufrn.br, affonso@dca.ufrn.br

Abstract— Formal methods to validate and verify the controller software have been discussed enough in the scientific literature for reducing the dependence by PLC programmers that use the languages of the IEC-61313-3 standard. This paper presents an approach to model and validate controller software to industrial applications that use sequential and parallel operations. One typical example of the manufacturing application is described in the paper to illustrate our proposal.

Keywords— Modeling, Validation, Statecharts, Industrial applications.

Resumo— Métodos formais para validar e verificar o software de controle têm sido bastante explorados na área de informática industrial com objetivo de reduzir a dependência de programadores PLC especializados nas linguagens do padrão IEC-61131-3. Este artigo apresenta uma técnica para modelagem e validação do software de controle para aplicações industriais que envolvem operações seqüenciais e paralelas. Um exemplo típicos da área de manufatura é discutido como estudo de caso para ilustrar nossas idéias.

Keywords— Modelagem, Validação, Statecharts, Aplicações industriais.

1 Introdução

A área de automação utiliza conceitos da teoria de sistemas para controlar máquinas e processos industriais. Considerando o processo de automação baseado em Controladores Lógico Programáveis (do inglês: Programmable Logic Controllers - PLC), os **sensores** são instalados no módulo da planta e geram eventos que representam as variáveis de entrada do PLC. Os **atuadores** estão associados com as ações produzidas pelo programa PLC e representam as variáveis de saída. Em uma aplicação genérica podem existir k sensores and n atuadores, além de m dispositivos temporizadores e contadores implementados nos PLCs através de memórias auxiliares, k, n e m são valores inteiros não negativos.

Atualmente, a programação dos PLCs ainda é realizada por técnicos com conhecimentos específicos em uma das cinco linguagens definidas pelo padrão IEC-61131-3 (IEC, 1993) e que raramente utilizam tecnologias de desenvolvimento de software modernas. Além disso, os controladores são freqüentemente reprogramados durante a operação da planta para se adaptar a novos requisitos. Estes dois pontos justificam a afirmação de que: "não existe uma descrição formal para praticamente nenhum controlador implementado na indústria" (Bani Younis and Frey, 2006). Portanto, o uso de metodologias de mais alto nível na programação de controle constitui um grande desafio a ser conquistado.

Por outro lado, na comunidade acadêmica há várias propostas para sistematizar a síntese de controladores. Dentre eles, a que mais se destaca é a Teoria de Controle Supervisório, desenvolvida por Ramadge and Wonham nos anos 80 (Ramadge and Wonham, 1989). Ela se fundamenta nos aspectos formais dos autômatos para a síntese automática de controladores. No entanto, apesar dos avanços nesta área, os pesquisadores não conseguiram convencer os engenheiros industriais a migrar dos métodos de programação tradicionais para as novas abordagens. Skoldstam et. al. em (Skoldstam et al., 2008) indicam duas razões para o insucesso da Teoria de Controle Supervisório: i) discrepância entre a realidade baseada em sinais e os autômatos que são baseados em eventos; e ii) falta de uma representção compacta dos grandes modelos. Já Gourcuff et al. em (Gourcuff et al., 2006) destacam três pontos para a não utilização de métodos formais na indústria: i) dificuldade para especificar propriedades formais em lógica temporal ou na forma de autômatos temporais, torna a modelagem do problema uma tarefa extremamente árdua para muitos engenheiros; ii) os model-checkers fornecem, no caso de prova negativa, contra-exemplos que são difíceis de interpretar; e iii) os fabricantes de PLCs não propõem ferramentas comerciais capazes de traduzir automaticamente programas PLC em modelos formais. Dessa forma, o uso de metodologias de mais alto nível na programação de PLCs constitui um grande desafio a ser conquistado. Porém, manter o foco nos aspectos formais é importante para análise, validação e verificação dos modelos, mas eles devem ser abstraídos para o entendimento do sistema por parte do projetista. Na verdade, a solução ideal para a indústria da automação é o uso de um ambiente computacional que permita ao engenheiro industrial implementar, de forma natural, o sistema em uma linguagem que siga um

padrão internacional.

Em atenção aos problemas mencionados, o objetivo deste artigo é propor uma técnica para modelagem e validação do software de controle para aplicações da área de manufatura que envolvem operações seqüenciais, paralelas e temporais. No processo de modelagem utiliza-se o formalismo *UML/Statecharts*, proposto originalmente por David Harel nos anos 80 (Harel, 1987). Enquanto que no processo de validação tem-se utilizado simulações através do ambiente de execução SCXML, que foi implementado pelo projeto Jakarta (SCXML, 2006). Como o modelo do software de controle não representa o próprio controlador, a tradução deste modelo para uma linguagem de programação aceita por PLCs também tem sido realizada. Aqui, tem-se utilizado os *diagramas Ladder* por ser uma das linguagens do padrão internacional IEC-61131-3 mais usadas na indústria. No entanto, esses modelos podem ser convertidos para qualquer uma das outras linguagens do referido padrão. Apesar de existir diversos formalismos para modelagem de DESs, optou-se pelo uso dos *UML/Statecharts* por três razões principais: i) possibilitar a validação formal dos modelos; ii) ser um formalismo de mais alto nível, tornando o projeto dos controladores de forma mais natural para os projetistas; e iii) ter sido incorparado pela *Unified Modeling Language (UML)* para descrever aspectos dinâmicos da visão comportamental do sistema.

O restante deste artigo está organizado da seguinte maneira: a seção 2 descreve o processo de modelagem do software de controle para aplicações da área de manufatura usando o formalismo *Statecharts*. A seção 3 destaca a técnica de validação utilizada em nossos modelos baseada no ambiente de execução SCXML. Em seguida, a seção 4 apresenta o procedimento de tradução do modelo do controlador para a *linguagem Ladder*. Um exemplo de aplicação típica que envolve operações seqüenciais e paralelas dos componentes atuadores é apresentado como estudo de caso. Finalmente, a seção 6 conclui este artigo e destaca alguns trabalhos futuros.

2 Software de Controle: Modelagem

Metodologias para programação de PLCs baseados em modelos têm sido bastante exploradas na área acadêmica. Normalmente, essas metodologias seguem a idéia de i) projetar o modelo da planta; ii) projetar o modelo do controlador; iii) validar e/ou verificar o comportamento modelo da planta em conjunto com o do controlador; e iv) traduzir o modelo do controlador para uma das linguagens aceitas pelos PLCs.

O modelo da planta permite simular o funcionamento não controlado dos componentes sensores e atuadores, produzindo um número elevado de possíveis estados. Por exemplo, se um sistema é composto de três atuadores c1, c2 e c3 com três estados cada, o número total de possíveis estados do conjunto é igual a $27 = 3 \times 3 \times 3$. No artigo (Moura and Guedes, 2007) discutem-se uma metodologia de alto nível para sistematizar o processo de modelagem da planta de aplicações industriais da área de manufatura usando diagramas da UML. Neste trabalho trata-se apenas da modelagem do software controlador.

Na área de manufatura, os componentes atuadores são controlados, normalmente, por eventos disparados por dispositivos do tipo botoeiras, sensores e temporizadores. Tais componentes devem ser definidos no modelo do controlador através de variáveis auxiliares. Portanto, o controlador pode ser modelado pela composição de componentes de três tipos básicos: a) **atuadores** que devem ser modelados através de componentes com dois estados: OFF e ON; b) **temporizadores** que devem ser modelados através de componentes com três estados: OFF, START e ON, sendo que no estado START deve-se iniciar a temporização e a transição tm(ts) do estado START para ON representa o evento de fim da temporização; e c) **variáveis** associadas aos sensores e chaveadores auxiliares. A Figura 1 mostra o modelo básico para esses elementos. A Figura 1-c destaca a área de dados usando a sintaxe da especificação SCXML. No exemplo, são definidas duas variáveis booleanas s1 e s2 com valores "false".

Os requisitos operacionais dos atuadores são inseridos no modelo como transições entre os estados na forma geral: "evento [condição] / ação". As condições são expressões booleanas formadas pelos sensores e chaveadores auxiliares, interligados por conectores lógicos ¬ (negação), ‖ (disjunção) e & (conjunção). As ações podem ser, por exemplo, o disparo de um evento interno ao modelo da planta e/ou a atribuição de valor para uma variável. Para restringir a atuação do modelo e gerar o funcionamento desejado dos componentes (comportamento seqüencial e/ou paralelo), o modelo do controlador deve ser concebido através da inclusão de requisitos operacionais que serão exemplificados nas subseções a seguir.

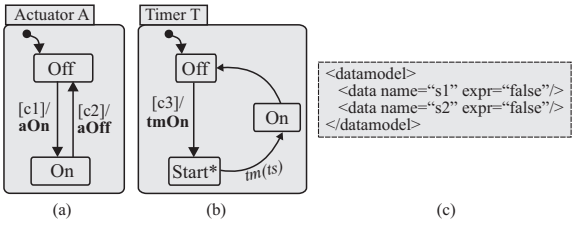

Figura 1: Atuadores: modelo básico

2.1 Operação seqüencial

Considere uma planta composta de dois atuadores A_i e A_j que executam seqüencialmente, um após o outro, ou seja, $A_i; A_j$. Essa seqüência é executada continuamente de maneira cíclica até uma intervenção manual do operador. O comportamento seqüencial de A_i e A_j é obtido através da execução de ações no atuador A_i que provoca o disparo de eventos internos ao atuador A_j. De modo geral, uma ação em um atuador pode provocar mudanças de estado em outros atuadores.

A Figura 2-a mostra o diagrama *Statechart* que modela o comportamento seqüencial entre os atuadores A_i e A_j discutidos acima. Na figura, c1 e c2 são condições de guarda quaisquer, "start" e "ev" são eventos internos ao modelo do controle, usados para obter a funcionalidade desejada do sistema. No caso, o evento "ev" é disparado como uma ação pelo atuador A_i indicando o seu fim de atuação. Este evento é percebido pelo atuador A_j que inicia a sua operação, gerando o comportamento seqüencial entre os mesmos. Ao final da atuação de A_j, o evento "start" é disparado, gerando assim o comportamento cíclico do modelo. Na configuração inicial, o modelo se encontra com os atuadores desligados (estado "Off") e a partida do sistema é realizada através do disparo do evento "start", que pode ser feito, por exemplo, pelo pressionamento de um botão.

Para se obter uma nova configuração do modelo, os eventos que representam dependências entre componentes são implicitamente forçados e disparados. Essa característica diminui o número de estados possíveis do sistema. No entanto, ela pode gerar "loop infinito" no modelo semântico dos *Statecharts*. Por exemplo, sejam dois atuadores A e B com estados "Off" e "On" e transições "a/b" entre os estados de A e "b/a" entre os estados de B (ver Figura 2-b). O disparo da transição "a/b" no atuador A provoca o disparo da transição "b/a" no atuador B, que provoca o disparo da transição "a/b" em A, e assim sucessivamente, gerando um loop. Para evitar essa inconsistência, um mecanismo para disparar apenas uma transição por atuador em cada passo de execução do modelo deve ser implementado. Assim, as ações que disparam eventos em atuadores que já sofreram evolução no mesmo passo devem ser armazenadas para atuarem no passo seguinte do modelo. Há a necessidade de se determinar ordem de disparo de modo a evitar não-determinismo. A semântica utilizada em nossos trabalhos considera os diagramas da esquerda para a direita e de cima para baixo. Considerando a representação no formato SCXML, a ordem é obtida a partir da definição dos estados no arquivo XML.

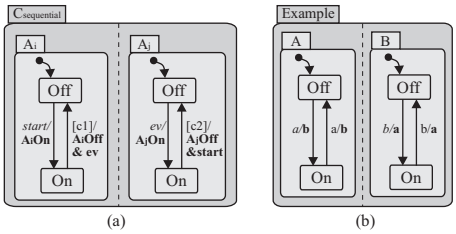

Figura 2: Modelo de controle: operação sequencial

2.2 Operação Paralela

Paralelismo é uma característica inerente ao formalismo *Statecharts*, sendo realizado através de decomposição-AND. No entanto, a sincronização entre os componentes necessita de mecanismos adicionais. Considere, por exemplo, três atuadores A_i, A_j e A_k, onde A_i e A_j podem executar em paralelo, porém A_k executa somente após a execução dos dois primeiros, ou seja, $(A_i||A_j); A_k$. Essa seqüência é executada continuamente de maneira cíclica até a intervenção do operador. O comportamento paralelo dos atuadores A_i e A_j é obtido naturalmente, porém, é necessário o disparo de eventos internos ao atuador A_k para indicar o fim da execução dos outros atuadores. Assim, o atuador A_k espera por tais eventos para que possa iniciar sua operação. Após a execução de A_k, eventos internos devem ser disparados para permitir a execução de um novo ciclo do sistema.

A Figura 3 mostra o diagrama *Statechart* que modela o comportamento paralelo entre os atuadores A_i e A_j discutido acima, e o atuador A_k que executa após a operação dos mesmos. Na figura, $c_i (i = 1 \ldots 5)$ são condições de guarda quaisquer, "ev_i" e "ev_j" são eventos internos ao modelo, utilizados para obter a funcionalidade desejada do sistema. No caso, o evento "evi" é disparado como uma ação pelo atuador A_i indicando o seu fim de atuação e "ev_j" é o evento disparado para indicar o fim de operação do atuador A_j. Estes eventos são percebidos pelo atuador A_k que inicia a sua operação, gerando o sincronismo entre os mesmos. Ao final da operação do atuador A_k os eventos "ev_i" e "ev_j" devem ser "resetados" para a execução de um novo ciclo do sistema.

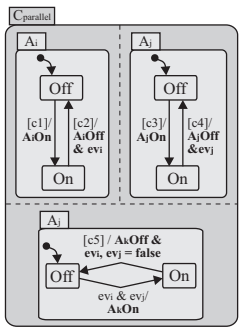

Figura 3: Modelo de controle: operação paralela

2.3 Timed Operation

O uso de temporizadores e contadores é bastante comum, por exemplo: i) um atuador deve executar por um tempo especificado; ii) um atuador só deve executar após um determinado tempo; iii) o sistema deve executar k vezes antes de disparar um alarme; entre outras. Temporizadores e contadores são modelados através de componentes e os valores atuais podem ser utilizados para compor as condições de guarda das transições. Além disso, eles podem ser iniciados e/ou resetados por alguma ação do modelo.

Considere, por exemplo, um sistema composto de um atuador A_i e um temporizador T_k, onde A_i deve atuar por t segundos, antes de ser desligado. A Figura 4 mostra o diagrama *Statechart* que modela o comportamento temporal do atuador A_i, controlado pelo temporizador T_k. Na figura, c1 e c2 são condições de guarda para iniciar a operação do atuador A_i e do temporizador T_k, respecitivamente e "tk.tm" é um evento interno ao modelo, utilizado para indicar o fim da temporização de T_k e é disparado como uma ação de entrada no estado "On" do temporizador T_k. O evento que inicia a temporização de T_k é disparado como uma ação de entrada no estado "On" do atuador A_i. Esses eventos não são explicitamente representados no modelo gráfico, sendo apenas indicados com um "*" (asterisco) na figura. Note que na criação do temporizador ele deve ser devidamente configurado com o tempo desejado.

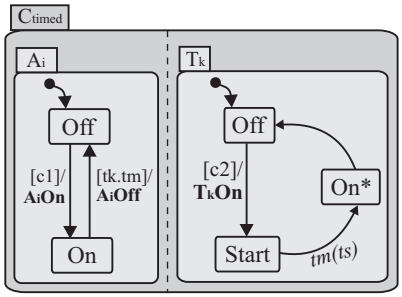

Figura 4: Modelo de controle: operação temporal

3 Software de Controle: Validação

A abordagem para modelagem do software de controle apresentada na seção 2 mantém os aspectos de descrição e especificação embutidos no modelo *Statechart*. Transições, condições de guarda e ações implícitas são usadas para descrever as restrições do sistema. Dessa forma, a abordagem permite analisar algumas propriedades do controlador através do uso da árvore de alcançabilidade. Além disso, ambientes simulados podem ser usados para validar o modelo do controle em conjunto com o modelo da planta.

A árvore de alcançabilidade de um determinado modelo permite analisar propriedades, como: i) **reiniciabilidade** - para cada configuração de estados SC_i alcançada a partir da configuração inicial SC_0 é possível retornar à configuração SC_0 através de uma seqüência de eventos? ii) **vivacidade** - o controlador age em todos os componentes do modelo? iii) **deadlock** - existe uma configuração SC_i na qual não é possível evoluir porque nenhuma transição é capaz de ser disparada?

Na literatura, existe um algoritmo para criação da árvore de alcançabilidade para *Statecharts* proposto por Masiero et al., em (Masiero et al., 1994). Aqui, discute-se brevemente a adaptação de tal algoritmo para a análise das propriedades estruturais citadas anteriormente. O algoritmo foi implementado em conjunto com o ambiente de execução SCXML, com as seguintes modificações:

- O conjunto que contém as possíveis transições para uma dada configuração inclui somente as transições com eventos controlados através de um agente externo e com eventos temporais disparados automaticamente pelos componentes para simular restrições físicas dos dispositivos;
- Para obter uma nova configuração do modelo através do disparo de transição, os eventos que representam dependências entre componentes são implicitamente forçados e disparados. Tais características diminuem o número de estados na árvore de alcançabilidade produzida;
- A parte do algoritmo que descreve aspectos de história foi completamente excluída, uma vez que os *"Basic" Statecharts* não incluem essa característica.

Com isto é possível fazer uma análise formal do comportamento do sistema (controle + planta). Nota-se que no caso existe um modelo da planta, representado em algum formalismo que permita verificação e validação, como por exemplo autômato, Rede de Petri ou *Statecharts*. Em (Moura et al., 2008), nós propomos um procedimento sistemático para modelagem de plantas complexas via *Statecharts* e discutimos alguns aspectos da modelagem do controlador. No entanto, foi apresentado apenas uma visão descritiva de nossas idéias.

Neste artigo, optou-se, sem perda de generalidade, por modelar o comportamento da planta em *Statecharts*. Então, o sistema (controle + planta) pode ser descrito como ilustrado na Figura 5, onde o sistema é uma composição em paralelo entre o controlador e a planta. A principal vantagem desta abordagem é que os eventos dos sensores e atuadores tornam-se eventos internos do sistema. Assim, as propriedades intrínsecas do sistema, com alcançabilidade, deadlock e re-iniciabilidade,

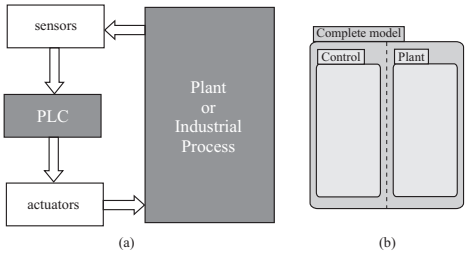

Figura 5: Modelo completo: planta + controle

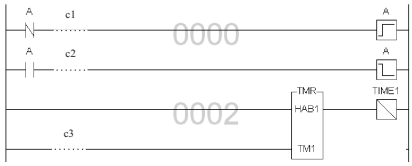

Figura 6: Atuadores: diagrama Ladder

tornam-se propriedades intrínsecas e extrínsecas do controlador.

Outra vantagem dessa abordagem deve-se ao fato de se manter explicitamente separadas as funcionalidades do controlador e da planta. Com isto, diferentemente de outras abordagens, como R & W (controle supervísório), a síntese do controlador leva a modelos bem mais compactos. Na próxima seção é apresentado um procedimento, com respectivo algoritmo, para transformar o modelo do controlador descrito em *Statecharts* para uma linguagem implementável em PLCs. No caso, optou-se pela *linguagem Ladder*.

4 Software de Controle: Implementação

Como o modelo do controle não representa o próprio controlador, a tradução deste modelo para uma linguagem de programação aceita por PLCs deve ser realizada. No caso, optou-se pela tradução para *linguagem Ladder*. A tradução é feita sistematicamente através de um método que analisa um componente de cada vez, de acordo com o seu tipo: **atuador** ou **temporizador**.

Os estados OFF e ON dos dispositivos atuadores são representados em *Ladder* através de contatos auxiliares do tipo *flip-flop Reset* e *flip-flop Set*, repectivamente. Cada transição do modelo gera uma linha *Ladder* da seguinte forma: o estado origem deve ser adicionado à condição de guarda e o estado destino representa a ação a ser executada pela transição. Considere, por exemplo, o atuador genérico mostrado na Figura 1-a, as transições "[c1]/aOn" e "[c2]/aOff" geram as linhas 3 e 4 do *diagrama Ladder* apresentado na Figura 6.

Os temporizadores foram traduzidos da seguinte forma: a transição do estado OFF para START que habilita o temporizador é usada para iniciar a contagem; o evento para indicar o fim da temporização programada é definida através de um relé auxiliar que pode ser usado em outros trechos do código *Ladder*, de acordo com a aplicação. O temporizador genérico mostrado na Figura 1-b, é traduzido para as linhas 5 e 6 do *diagrama Ladder* (ver Figura 6).

As variáveis auxiliares que representam os sensores e chaveadores são usadas livremente nas condições de guarda e ações do código *Ladder*, de acordo com as transições do modelo. Como as condições de guarda das transições (em cada linha *Ladder*) devem ser garantidas por uma varredura completa do PLC (PLC-scan cycle), todas as condições devem ser avaliadas e armazenadas em variáveis auxiliares no início do ciclo (ver linhas 0, 1 e 2 na figura).

É importate notar que para evitar não-determinismo no sistema, as condições de guarda para um mesmo estado fonte devem ser mutuamente exclusivas. Essas restrições podem ser verificadas em tempo de construção do modelo e podem ser reportadas para o usuário através de mensagens de advertência. No entanto, como elas são mutuamente exclusivas apenas para um mesmo estado do atuador, as linhas *Ladder* não podem ser geradas em qualquer ordem, pois inconsistências do tipo ligar/desligar um atuador podem ocorrer em uma mesma varredura. Para evitar tais inconsistências, o estado temporário dos atuadores deve ser armazenado em variáveis auxiliares e, no final do código, essas variáveis devem ser atualizadas para as saídas correspondentes (ver linhas 7 e 8 na figura).

O algoritmo completo utilizado para traduzir o software de controle para código *Ladder* é apresentado a seguir:

5 Estudo de Caso

Para ilustrar melhor a metodologia apresentada no artigo, um exemplo típico da área de manufatura é analisado como estudo de caso.

5.1 Examplo: Célula de Manufatura

Considere a célula de manufatura apresentada na Figura 7, cujos dispositivos podem ser executados simultaneamente. O problema com estas execuções é a necessidade de pontos de sincronização entre trechos/blocos paralelos.

Considere também o fluxo de execução com uma possível operação dos dispositivos para este sistema como mostrado na Figura 8. Observa-se que os quatro dispositivos podem funcionar de maneira simultânea e que a mesa deve funcionar somente após o fim da operação dos mesmos. Assim, é necessária a criação de um ponto de sincronização para os dispositivos e, então, permitir a rotação da mesa.

Algoritmo 1. Tradução do Modelo do controle para diagrama Ladder

read input variables {sensors: True / False}

compute $C_i(i = 1, 2, \ldots)$ {guard conditions}
{For each timer} {k = number of timers}
for $j = 1$ to k **do**
  if $T_j.enabled$ and $T_j.timeout$ **then**
    $T_j.tm := true$
  else
    $T_j.tm := false$
  end if
end for
...
{For each actuator} {n = number of actuators}

for $j = 1$ to n **do**
  $A_j.temp := A_j$ and C_i
end for
...
{Update outputs from temporary variables}
for $j = 1$ to n **do**
  $A_j := A_j.temp$
end for
write output variables {actuators: On / Off}

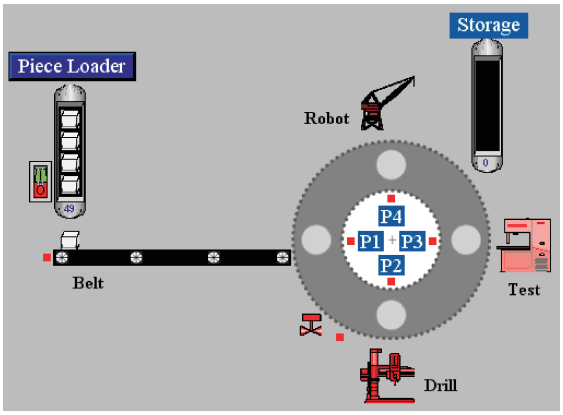

Figura 7: Simulação

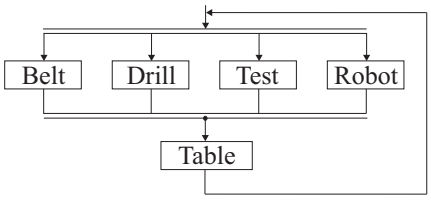

Figura 8: Fluxo de execução

Considere ainda os cenários de funcionamento dos dispositivos descritos a seguir:

BELT : Se existir peça no buffer de entrada (posição inicial da esteira) e não existir peça na posição P1, a esteira deve ser ligada; depois, quando a peça chegar na posição P1 a esteira deve ser desligada. As cláusulas **se ... então** desta especificação são:

- **Se** inputbuffer & ¬P1 **então** BeltOn;
- **Se** P1 **então** BeltOff;

DRILL : Se existir peça na posição P2, a furadeira e um temporizador timerT1 devem ser ligados; ao final da temporização, a furadeira deve ser desligada. As cláusulas **se ... então** desta especificação são:

- **Se** P2 **então** DrillOn & tm1On;
- **Se** tm1.tm **então** DrillOff;

TEST : Se existir peça na posição P3, o testador e um temporizador timerT2 devem ser ligados; ao final da temporização, o testador deve ser desligado. As cláusulas **se ... então** desta especificação são:

- **Se** P3 **então** TestOn & tm2On;
- **Se** tm2.tm **então** TestOff;

ROBOT : O robô retira uma peça da posição P4 e a coloca em um depósito. Se existir peça na posição P4, o robô e um temporizador timerT3 devem ser ligados; ao final da temporização, o robô deve ser desligado. As cláusulas **se ... então** desta especificação são:

- **Se** P4 **então** RobotOn & tm3On;
- **Se** tm3.tm **então** RobotOff;

TABLE : A rotação da mesa é controlada por um cilindro de ação simples. O avanço total do braço do cilindro provoca o giro de 90°. Assim, após a operação dos quatro dispositivos, o cilindro deve ser acionado para obter uma nova configuração do ambiente. O retorno do cilindro deverá ocorrer quando o sensor que detecta o avanço total do braço for observado. As cláusulas **se ... então** para esta especificação são:

- **Se** BeltEnd & DrillEnd & TestEnd & RobotEnd **então** ValveOn;
- **Se** SensorOn **então** ValveOff;

Considere, finalmente, que as outras restrições impostas ao modelo são:

1. Cada dispositivo deve executar somente uma vez, antes de uma rotação da mesa;

2. Se em uma dada configuração não existir peça no buffer ou nas posições P2, P3 e P4 então a esteira, a furadeira, o testador e o robô não devem ser acionados.

3. A mesa só deve girar se existir pelo menos uma peça nas posições P1, P2 ou P3;

A inclusão dessas restrições no modelo do controlador é realizada através da definição de novas transições entre os estados e/ou alterações nas condições de guarda. Para a criação do software de controle do exemplo em questão, inicialmente, deve-se criar variáveis auxiliares para sincronizar a execução entre os dispositivos e a rotação da mesa. No caso, ev1, ev2, ev3 e ev4 significam fim de operação da esteira, da furadeira, do testador e do robô, respectivamente. Essas variáveis devem ter seus valores atribuídos para "true" no final da operação de cada um dos dispositivos. De acordo com o funcionamento normal da célula, a mesa só deve girar quando todos os dispositivos tiverem concluído suas operações, ou seja, quando as variáveis $ev_i = true (i = 1, \ldots, 4)$. Ao final do processo de rotação, essas variáveis devem ter seus valores atribuídos para "false". As variáveis são usadas também nas transições para ligar os atuadores, por exemplo, a furadeira só deve ser ligada se a variável de controle ev2 for igual a "false".

Transições extras para garantir as restrições adicionais 2 e 3 devem ser incluídas no modelo, por exemplo, se não existir peça na posição P2, então a furadeira não deve ser ocionada, porém, a variável ev2 deve se tornar "true" para indicar o fim de operação da etapa. Essa idéia é aplicada aos outros dispositivos atuadores. No modelo da mesa, se não existir peças nas posições P1, P2 ou P3, então a mesa não deve girar (restrição 3), no entanto, as variáveis ev1, ev2, ev3 e ev4 devem se tornar "false" para permitir novas operações dos dispositivos. Assim, se não existir peças na célula de manufatura, o modelo irá alternar continuamente o valor das variáveis ev1, ..., ev4 entre "false" e "true". O modelo completo do controlador é apresentado na Figura 9. As condições de guarda c1, c2, ..., c15 são apresentadas na Tabela 1, onde a variável $IN = true$ indica ou não a presença de peça no buffer de entrada.

O exemplo em questão é composto de quatro dispositivos com três estados cada, e um cilindro que controla a mesa que possui também três estados. O modelo sem controle possui 243 estados, ou seja, 3 x 3 x 3 x 3 x 3 = 243 configurações distintas. A árvore de alcançabilidade do modelo controlado gerou 227 configurações distintas. Os 16 estados que foram reduzidos indicam as combinações de eventos referentes a peças nas posições *buffer de entrada*, P2, P3 e P4 que não podem ser geradas enquanto a mesa está girando. Assim, considerando que somente 16 cnfigurações foram evitadas, é possível dizer que essa abordagem produz um modelo pouco restritivo. Pela análise da árvore de alcançabilidade, verificou-se que não há deadlock e o sistema é reiniciável. Quanto a validação, o sistema segue o comportamento projetado.

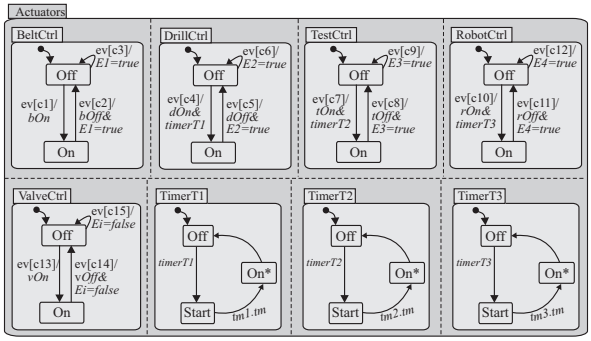

Figura 9: Modelo de controle

Tabela 1: Controlador: condições de guarda

c1	¬P1 & ¬ev1 & IN
c2	P1
c3	¬P1 & ¬ev1 & ¬IN
c4	P2 & ¬ev2
c5	tm1.tm
c6	¬P2 & ¬ev2
c7	P3 & ¬ev3
c8	tm2.tm
c9	¬P3 & ¬ev3
c10	P4 & ¬ev4
c11	tm3.tm
c12	¬P4 & ¬ev4
c13	ev1&ev2&ev3&ev4 & (P1 ‖ P2 ‖ P3)
c14	S1
c15	ev1&ev2&ev3&ev4 & ¬P1 & ¬P2 & ¬P3

A Figura 10 mostra um trecho do *código Ladder* para o exemplo da célula de manufatura, gerado a partir do algoritmo discutido na seção 4.

6 Conclusão

Este trabalho apresentou uma metodologia para sistematizar o processo de modelagem e validação do software de controe de aplicações industriais, em especial, aplicações da área de manufatura. A proposta utiliza o formalismo *Statechart* em conjunto com o ambiente de execução SCXML para representar, analisar e validar os modelos.

Um algoritmo para criar a árvore de alcançabilidade do modelo controlado (planta + controle) foi implementado em linguagem Java para analisar as propriedades de *reiniciabilidade*, *vivacidade* e

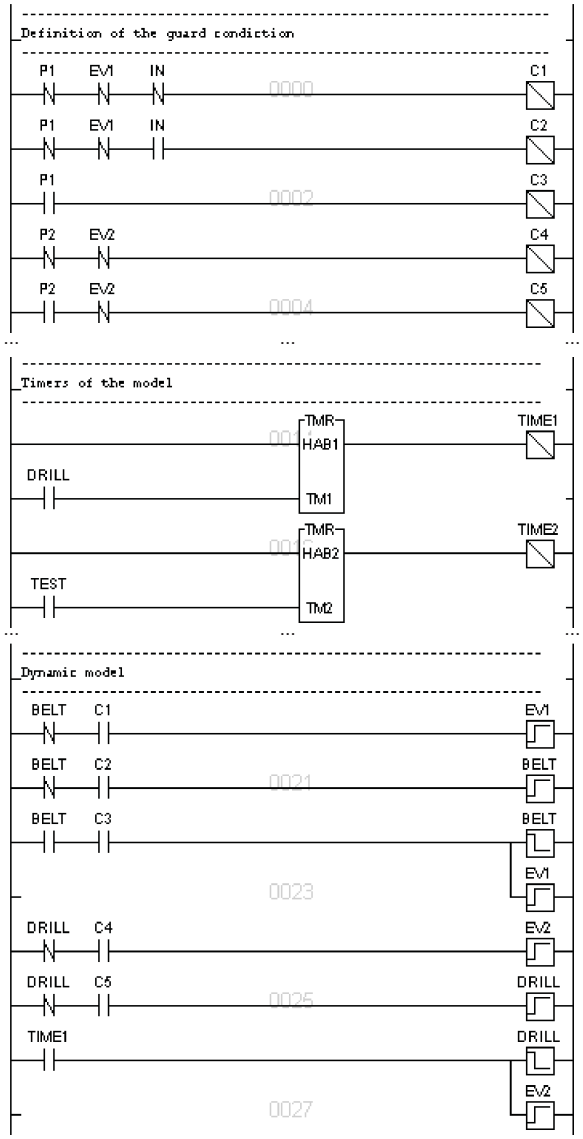

Figura 10: Diagrama Ladder parcial

deadlock de nossos modelos. O controle reduz o espaço de estados do modelo da planta e garante a execução apenas das operações desejadas. Um procedimento para traduzir o modelo do controlador para *código Ladder* também foi discutido com o objetivo de realizar testes e analisar os resultados em aplicações reais. Um exemplo típico de aplicação da área de manufatura foi discutido como estudo de caso para ilustrar melhor nossas idéias.

Como trabalhos futuros, pretende-se formalizar os métodos e procedimentos apresentados no artigo. Tal formalização visa a verificação formal dos nossos modelos, além de permitir comparações com estudos de casos desenvolvidos através da teoria clássica de controle supervisório, como a abordagem R&W, baseada em autômatos finitos e abordagens baseadas em Rede de Petri.

Referências

Bani Younis, M. and Frey, G. (2006). UML-based approach for the re-engineering of PLC programs, *32nd Annual Conference of the IEEE Industrial Electronics Society (IECON'06)*, pp. 3691–3696.

Gourcuff, V., Smet, O. D. and Faure, J.-M. (2006). Efficient representation for formal verification of plc programs, *8th International Workshop on Discrete Event Systems (WODES 2006)*, Ann Arbor, Michigan, USA.

Harel, D. (1987). Statecharts: A visual formalism for complex systems, *Science of Computer Programming* **8**(3): 231–274.

IEC (1993). International eletrotechnical commission. programmable controllers part 3, programming languages, IEC61131-3.

Masiero, P., Maldonado, J. and Boaventura, I. (1994). A reachability tree for statecharts and analysis of some properties, *Information and Software Technology* pp. 615–624.

Moura, R., Couto, F. and Guedes, L. (2008). Control and plant modeling for manufacturing systems using statecharts, *IEEE International Symposium on Industrial Electronics (ISIE 2008)*.

Moura, R. and Guedes, L. (2007). Modelagem de aplicações de automação industrial usando diagramas uml e reuso de componentes, *VIII Simpósio Brasileiro de Automação Inteligente (SBAI 2007)*.

Ramadge, P. and Wonham, W. (1989). The control of discrete event systems, *Proceedings of the IEEE*, Vol. 77(1), pp. 81–98.

SCXML (2006). The jakarta project commons SCXML, http://jakarta.apache.org/commons/scxml/.

Skoldstam, M., Akesson, K. and Fabian, M. (2008). Supervisory control applied to automata extended with variables - revised, *Technical report*, Goteborg: Chalmers University of Technology.