Universidade Federal de Juiz de Fora
Programa de Pós-Graduação em
Engenharia Elétrica

Wolmar Araujo Neto

Construção e Sintetização de Modelos de Estado para o Controle de Processos

Dissertação de Mestrado

Juiz de Fora
2014

Wolmar Araujo Neto

Construção e Sintetização de Modelos de Estado para o Controle de Processos

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, área de concentração: Sistemas de Energia, da Faculdade de Engenharia da Universidade Federal de Juiz de Fora como requisito para obtenção do grau de Mestre.

Orientador: Prof. Dr. Leonardo de Mello Honório

Juiz de Fora
2014

Wolmar Araujo Neto

Construção e Sintetização de Modelos de Estado para o Controle de Processos

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, área de concentração: Sistemas de Energia, da Faculdade de Engenharia da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do grau de Mestre.

Aprovada em 25 de Agosto de 2014.

BANCA EXAMINADORA:

Prof. Dr. Leonardo de Mello Honório - Orientador
Universidade Federal de Juiz de Fora, UFJF

Prof. Dr. Luiz Edival de Souza
Universidade Federal de Itajubá, UNIFEI

Prof. Dr. Pedro Gomes Barbosa
Universidade Federal de Juiz de Fora, UFJF

Dedico esta dissertação aos meus avós, em especial a Vó Giselda, pelos quais possuo profunda admiração e amor.

AGRADECIMENTOS

Agradeço, primeiramente a Deus e Nossa Senhora, por terem me dado saúde, força de vontade, e esperança. Aos meus pais pelo amor incondicional, carinho e compreensão, aos meus amigos e amigas por sempre acreditarem em mim e nunca me deixarem desistir dos meus sonhos. Por fim, meu muito obrigado, aos amigos do GRIn, que foram minha família durante esses anos em Juiz de Fora.

"Devemos dividir nossos problemas no maior número possível de partes, para melhor resolvê-los."

René Descartes

RESUMO

A presente dissertação tem como objetivo principal desenvolver uma metodologia capaz de auxiliar no projeto e implementação de controles para sistemas a eventos discretos (SED) de forma a otimizar a modelagem, documentação e implantação da planta estudada. Para tanto a definição tradicional de autômato foi alterado para incluir um novo subconjunto de estados obrigatórios, isto é, estados que o sistema deve passar antes de chegar a um estado marcado. Seguindo a modelagem tradicional, cada subsistema é modelado através desta nova abordagem. Em uma segunda etapa, uma nova iteração de análise de cada subsistema é realizada para restringir situações indesejadas do sistema. Com esta fase finalizada os subsistemas são mesclados através de uma técnica tradicional de autômatos em paralelo. O sistema final resultante é apresentado a um otimizador, baseado em PROLOG, que busca a melhor sequência de eventos obedecendo a todas as restrições modeladas e termine no estado marcado informado pelo projeto. Este resultado fornece a Rede de Petri ideal do sistema e sua programação em GRAFCET. Tais métodos serão testados em componentes de um sistema de manufatura flexível real.

Palavras chave: Sistemas a Eventos Discretos, Autômato, Redes de Petri, CLP, Otimização, PROLOG.

ABSTRACT

The present master thesis has as its main goal to develop a methodology able to assist in design and implementation of discrete event system (DES) controllers in order to optimize the modeling, documentation and deployment of plant in study. To do so the automaton traditional concept was changed to include a new subset of mandatory states, i.e. states that the system should pass before reach a marked state. Following the traditional modeling, each subsystem is modeled using this new approach. In a second step, a new iteration of analysis of each subsystem is performed to restrain system's unwanted situations. At the end of this step the subsystems are composed by a traditional parallel automatons technique. The resulting system is presented to an optimizer, based in PROLOG, which searches the best event sequence obeying to all modeled restrictions and ends in the marked state defined in design. This result provides the ideal Petri Net of the system and its GRAFCET programming. Such methods will be tested in a flexible real manufacturing system.

Keywords: Discrete Event Systems, Automaton, Petri Nets, PLC, Optimization, PROLOG.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Com o processo de industrialização sofrido pela nossa sociedade ao longo dos tempos desde da 1ª revolução industrial, criou-se a necessidade de medir e conhecer diversas grandezas físicas e posteriormente controlá-las. Com o aumento das tecnologias desenvolvidas e quanto mais competitivas se tornaram as economias, os processos industriais aumentavam sua complexidade e rigor (THOMAZINI; ALBUQUERQUE, 2005).

A automação surgiu a partir do momento que o homem passou a buscar um caminho para a redução dos seus trabalhos manuais em determinados processos. A criação do moinho hidráulico, por exemplo, durante o século X, tinha como intuito aumentar a produção de farinha (moinhos eram capazes de substituir o trabalho de 10 a 20 homens) (GOEKING, 2010).

Pode-se dizer que a Automação Industrial teve inicio por volta de 1970, posteriormente vindo ocorrer no Brasil por volta de 1975 (MATA, 2010). A palavra *automation* foi inventada na década de 1960, é um neologismo que indica a participação do computador no controle automático industrial. A automação resulta na busca um nível maior de qualidade nos processos, apresentando maior variedade de modelos para o mercado, segurança, diminuição das perdas materiais e energéticas, e mais qualidade da informação sobre o processo.

Uma linha de produção industrial é formado de recursos (matéria-prima, atuadores, computadores etc.) que, por meio de eventos de transformação dos insumos básicos (entradas), seguem uma determinada ordem (regras), para o desenvolvimento de um produto final. Com o intuito de se tornar mais competitivo, minimizando desperdícios e custos de modo a maximizar o lucro, diversos métodos de otimização são aplicados. Um enfoque eficiente é a modelagem do sistema produtivo na forma de um problema de fluxo em rede com restrições adicionais (CARVALHO; FILHO; FERNANDES, 1998).

No âmbito da administração, pode-se associar os métodos de otimização às hipóteses formalizadas, na década de 1980, pelo físico israelense Eliyahu Goldratt, conhe-

cida como “Teoria das Restrições” (do inglês, *Theory of Constraints - TOC*), a qual é definida como uma abordagem de gestão centrada na melhoria dos processos que restringem o fluxo da produção. Nesta teoria, podem existir dois tipos de restrições, as consideradas físicas (máquinas, material, pedidos, recurso), e não físicas (normas, procedimentos) ou como são nomeadas, restrições políticas (BACK, 2013).

Em problemas que envolvem automação de processos, pode-se associar os métodos de otimização e modelagem à “Teoria dos Autômatos” (CASSANDRAS, 2008), cujo foco principal compreende os estudos das denominadas “máquinas de estado”. Neste tipo de enfoque, autômatos formam a classe mais básica dos modelos de sistema de eventos discretos. Por meio da Teoria dos Autômatos, é possível modelar todos os estados de uma determinada linha de produção (ITO YUJI KOBAYASHI, 2010), e até mesmo dividir seus componentes em células, módulos, e pequenos subprocessos e representá-los por autômatos simples, desenvolvendo uma modelagem do sistema mais intuitiva, e menos complexa em relação ao todo.

Visando a comunicação entre Autômatos, Carl A. Petri, formalizou um método de estudo dos sistemas dinâmicos a eventos em sua tese de doutorado, conhecida como Redes de Petri (RP) (PETRI, 1962). Mais tarde, em 1970, os trabalhos de Holt e Commoner (COMMONER et al., 1971), vieram a contribuir para as definições das RPs, conhecidas até os dias de hoje, que ficaram praticamente padronizadas (MORAIS, 2013). Como descrito em (CASSANDRAS, 2008), todo autômato pode ser representado através de uma RP, entretanto nem toda Redes de Petri (RP) pode ser representada por uma autômato. Desta forma Redes de Petri representam uma classe ainda maior do que as das linguagens formais marcadas pelos autômatos. Uma das diferenças é que em uma Rede de Petri pode-se particionar um estado em suas variáveis e, com poucas marcações, representar um sistema muito complexo.

Como visto até o momento, enquanto a modelagem já tinha um certo “padrão” surgindo no início dos anos 70, sua implementação, isto é, a programação nos controladores industriais ainda não estava estabelecida. No intuito de desenvolver um padrão para programação para estes Controladores Lógicos Programáveis (CLPs), em 1979 foi criado um grupo de trabalho na Comissão Internacional Eletrotécnica (IEC). Com o objetivo de analisar por completo o projeto de um CLP, tais como hardware, instalação, teste, documentação, programação e comunicações, a norma IEC 1131, publicada em 1992, estabeleceu padrões para CLP, recebendo mais tarde o número 6, passando a ser conhecido assim como IEC 61131.

A seção da norma que estabelece o padrão global para programação de controladores lógicos programáveis foi publicada no ano de 1993. O IEC 61131-3 (JOHN; TIEGELKAMP, 2001) consiste na definição da linguagem para estruturar a organização interna do programa, sendo que a estrutura utilizada neste trabalho para a programação dos CLP's será a Função Gráfica de Sequenciamento (SFC), pois além de ser aceito pela maioria dos controladores lógicos programáveis modernos, tem uma estreita relação com Redes de Petri, por também ser composta de *passos*, *transições*, *arcos*; SFC possui ainda *ações qualificadas* e *expressões booleanas*, e graficamente é desenhada na vertical.

A norma também define outras quatro linguagens, sendo duas gráficas, Diagrama de Blocos de Funções (FBD) e Diagrama Ladder (LD), e duas textuais, Lista de Instrução (IL) e Texto Estruturado (ST).

Há uma necessidade de exprimir relações lógicas, ou, de causa e efeito, desencadeadas por eventos em instantes não predeterminados. Uma maneira de exprimir essas relações são por meio de Redes de Petri e diagramas de *ladder*, uma vez que esta abordagem apresenta maior poder descritivo de situações de concorrência de eventos e maior capacidade para prever conflitos, privilegiando seu estudo em conexão com a automação industrial (MORAIS, 2013).

No entanto, transformar um cenário desejado em linhas de programação não é uma tarefa simples para o programador, é preciso ter um conhecimento prévio a respeito do processo a ser modelado e programado. Pode-se dizer que o desafio inicial é modelar o sistema a ser controlado da forma mais fiel possível ao sistema real, com o intuito de impedir possíveis conflitos de decisões, garantindo a segurança dos usuários e elementos dos sistema, além de otimizar o processo no sentido de diminuir o tempo de execução e aumentar a eficiência.

Logo, a probabilidade de possíveis erros de lógica e erros de programação tendem a crescer na medida que o sistema a ser modelado apresenta uma quantidade significativa de elementos, além do mais, o tempo gasto para a realização e conclusão da modelagem e programação de sistemas mais complexos, tende a ser bastante elevado.

Um fato curioso e determinante para o avanço da automação até os dias de hoje, foi a necessidade da General Motors (meados de 1968) substituir seu controle baseado em relés por outro que não consumisse dias ou até semanas para sofrer alterações, que fosse capaz de reduzir custos, tendo a flexibilidade de um computador, e que fosse capaz de suportar as intempéries do ambiente industrial (MORAIS, 2013). O

Controlador Lógico Programável, tornou-se então de extraordinária importância para a industria, garantindo uma robustez adequada aos ambientes industriais, linguagem amigável para os projetistas, automatizando uma grande quantidade de ações com precisão, confiabilidade, e rapidez.

No final de 1980, ocorre a necessidade de modelagem, análise, controle e simulação de complexos sistemas de manufatura, especialmente integrados por computador. Levando pesquisadores e engenheiros a buscarem melhores metodologias e ferramentas; essas deveriam ser capazes de lidar com características do sistema como eventos assíncronos, sequências, concorrência, sincronização, exclusão mútua, impasses, e escolhas (BOGDAN FRANK L. LEWIS; JR., 2006).

Teoria como autômatos e máquinas de estado ganharam força no estudo dos sistemas de eventos discretos, por serem um instrumento capaz de representar em sua linguagem regras muito bem definidas (CASSANDRAS, 2008). Entretanto, foram logo revelando-se inadequadas para alguns cenários. Uma vez que os problemas de explosão de estado seria atingido no início do projeto do sistema, e quaisquer falhas de projeto ou imperfeições poderiam invalidar todo o projeto do sistema.

Por outro lado, as Redes de Petri, começaram a ser aplicadas ao controle de sistema de fabricação automatizada (efetuando-se inicialmente na França) sob a forma um pouco modificada da norma *Grafcet*, para a programação de autômatos programáveis industriais (CARDOSO, 1997). Bem equipadas com os recursos necessários para lidar com as características de fundamentais de controle nas industrias, foi ganhando uma certa popularidade entre os pesquisadores de sistemas de eventos discretos e aplicações industriais em automação da manufatura, por exemplo, a proposta do Dr. Robert Al-Jaar para o uso de redes de Petri estocásticas na modelagem e análise de linhas de produção (BOGDAN FRANK L. LEWIS; JR., 2006).

Entretanto, uma linguagem não sobressai totalmente a outra. Prova disso é uma pesquisa realizada em 2008 mostrando dois procedimentos de diagnóstico para sistemas a eventos discretos, baseados respectivamente em autômatos e Redes de Petri (LAI DAVIDE NESSI, 2008). Apresentando como resultado, que o procedimento de autômatos são mais gerais, contudo, a abordagem de Rede de Petri apresenta vantagens significativas em termos de complexidade computacional.

Ramadge Wonham propõem uma teoria de controle baseada na supervisão de SEDs (RAMADGE; WONHAM, 1987) , sendo o sistema que se deseja controlar deve satisfazer certas restrições qualitativas desejadas. Muitas extensões do problema básico de con-

trole de supervisão são estudadas, tais como controle com observações parciais, controle descentralizado, controle modular, controle de sistemas não-determinísticos e controle de comportamentos infinitos (KUMAR; GARG, 1995).

São encontrados na literatura diversas metodologias e softwares com a função de auxiliar a modelagem de sistemas, como *Aghata* (CHORLEY; BENCH-CAPON, 2004), que utiliza a ferramenta *carver* (LABBÉ; LAPITRE, 2006) para simplificar o automato que representa o sistema. A plataforma TART possui interface gráfica amigável para entrada de dados, construída em java, também analisando o sistema em linguagem autômata, que possui finalidade acadêmica. Outros exemplos baseam sua modelagem nas Redes de Petri para representar sistemas, como proposto em (MARTINEZ; MURO; SILVA, 1987) e (SILVA; MACIEL et al., 2004).

Nenhuma destas alternativas auxilia na construção do código para PLC. Em (ZYU-BIN, 2007) é proposta uma alternativa baseada em autômatos que auxilia o usuário na construção do código de controle.

Em 1972, na cidade de Marselha, surgia uma ferramenta com a proposta de implementar um sistema de comunicação homem-máquina. Philippe Roussel escolheu o nome PROLOG como uma abreviação para "*Programação en logique*", fruto de um casamento bem sucedido entre o processamento em linguagem natural e provas de teoremas automatizadas (COLMERAUER; ROUSSEL, 1996). Por se tratar de uma linguagem de programação centrada em torno de um pequeno conjunto de mecanismos básicos, incluindo correspondência de padrões, baseados em árvore de dados estruturados (BRATKO, 2001), se torna uma importante ferramenta aliada as Redes de Petri.

As RP's podem ser analisadas na forma de uma Sistema a Base de Regras e o PROLOG é uma linguagem de programação usada para resolver problemas relacionados a objetos e relações entre objetos (CLOCKSIN; MELLISH; CLOCKSIN, 1987). "A programação lógica faz sentido na medida em que traz elementos da lógica matemática para a programação de computadores. Parte do apelo à programação lógica vem do fato de que muitos problemas são expressos de maneira natural como teorias lógicas" (SILVA,2007,p.27).

Este trabalho refere à união destas linguagens, buscando absorver o melhor de suas qualidades, afim de auxiliar e otimizar o processo de programação dos CLP's responsáveis por controlar o sistema que se deseja modelar, sem necessidade do desenvolvimento de um supervisor no controle dos processos. Uma comparação de desempenho entre autômatos e Redes de Petri na modelagem de um sistema complexo pode ser vista em

(LAI et al., 2008).

Além de desenvolver uma nova ferramenta capaz de trabalhar um estilo mais descritivo de programação, onde o programador precisa saber quais relações existem entre diversas entidades, enquanto em outras formas de programação, o programador precisa dizer exatamente o que o computador precisa fazer (SILVA, 2007). É necessário buscar uma resposta ótima do resultado apresentado pelo mecanismo de inferência (PROLOG), como proposto em (LU; LIU, 2013).

Uma abordagem modular para controle de supervisão foi formulada e aplicada a quatro problemas de controle de supervisão em (WONHAM; RAMADGE, 1988). Em geral, as condições para a existência de uma solução modular são mais fortes do que para uma solução não-modulares. No entanto, quando aplicável, uma síntese modular oferece as vantagens importantes de redução da complexidade computacional e design modular estruturado.

2 MÓDULOS - FESTO

2.1 INTRODUÇÃO

Neste capítulo apresenta-se o Sistema Modular de Produção ("*Modular Production System*" MPS®) da empresa **FESTO** (Fig. 1) o qual tem como intuito a simulação de uma linha de produção industrial, voltado para o meio acadêmico.

O "Sistema de Aprendizagem" para automação proposto pela empresa inclui todos os temas atuais de automação em sua gama de produtos: eletro-pneumática, pneumática, hidráulica, eletro-hidráulica, eletrônica, sensores, robótica, tecnologia CNC, PLC, *Fieldbus*, tecnologia de fabricação e engenharia de processo, e também mecatrônica (DIDACTIC, 2013).

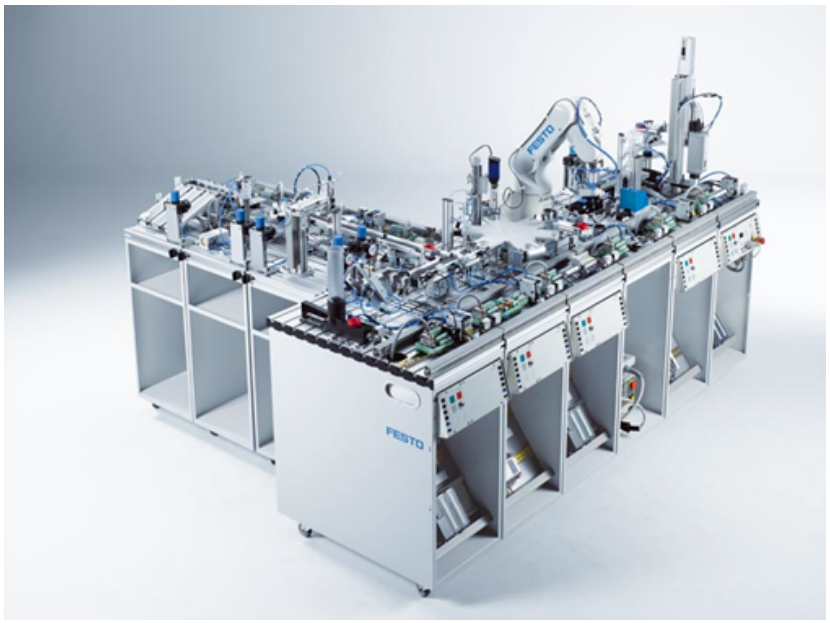


Figura 1: Sistema Modular de Produção - MPS
(POLA, 2013)

2.1.1 INTRODUÇÃO AO SISTEMA MODULAR DE PRODUÇÃO - MPS

Ao analisar o funcionamento da linha de produção de diversas fábricas, conclui-se que a mesma pode ser constituída por células de produção individuais. Cada célula tem uma função específica no processo (distribuição, análise, processamento, manuseio, montagem, armazenamento).

Combinando de forma eficiente estações individuais, é possível montar um sistema de produção capaz de atender diversas necessidades produtivas (POLA, 2013).

Por ser um sistema didático, a filosofia dos módulos é trabalhar diferentes assuntos, dentro dos quatro tópicos da Figura 2 abaixo:

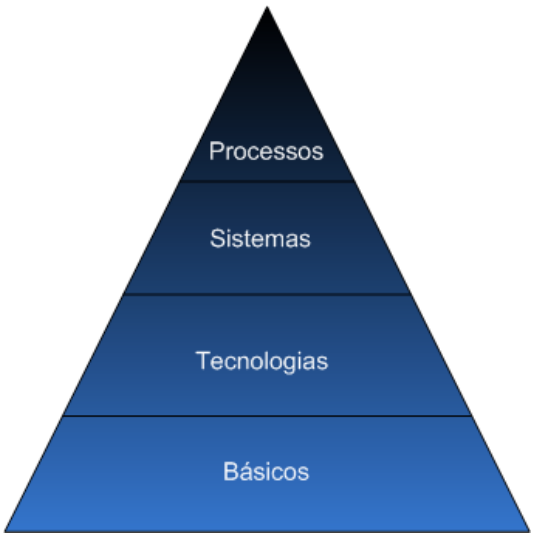


Figura 2: Pirâmide de Conhecimento

Sendo o tópico Básicos responsável por temas como Matemática, Física, Eletricidade, e Eletrônica por exemplo. O tópico Tecnologias abrange temas como Pneumáticas, Hidráulica, Sensores, PLC, CNC, e Robótica. Já o tópico de Sistemas trabalha com Comissionamento, Detecção de Falhas, Planejamento, Organização e Trabalho em Equipe. E no topo da Pirâmide, no tópico de Processos, o usuário pode trabalhar temas como Otimização, Manutenção Produtiva Total (TPM), Kanban, Automato, Programação Lógica (PROLOG), e etc.

PRODUTOS

Os produtos que o **MPS** irá trabalhar são peças do tipo camisa de cilindro, peças para corpo de medidor (Fig. 3), relógio, termômetro, e higrômetro (Fig. 4). Sendo o objetivo final, produzir pequenos cilindros pneumáticos.

(a) (167021) (b) (534619)

Figura 3: (a) Peças tipo camisa de cilindro e (b) para corpo de medidor (POLA, 2013)

Figura 4: Relógio (534621), Termômetro (534622), Higrômetro (534623) (POLA, 2013)

ESTRUTURAS DOS MÓDULOS

Cada módulo pode ser dividido em quatro partes, sendo elas: Planta, Painel de Controle, Gabinete Móvel, e Controlador. Onde a Planta é responsável por efetuar a ação desejada indicada pelo Painel de Controle, e o Controlador responsável por analisar o estado do módulo e indicar as próximas tarefas que devem ser efetuadas. A estrutura das estações é possível ser observada na figura abaixo (Fig. 5).

Figura 5: Estrutura da Estação MPS - (1) Planta, (2) Painel de Controle, (3) Gabinete Móvel, (4) Controlador

(POLA, 2013)

2.2 ESTAÇÕES - MPS

Nesta sessão, vamos apresentar a função de cada estação, suas peculiaridades, e seus principais componentes. Os *designers* das estações são propícios para a simulações de linhas de produção modulares, proporcionando assim uma flexibilidade na elaboração de uma variedade de cenários. A Figura 6 mostra o cenário trabalhado, em laboratório, na Universidade Federal de Juiz de Fora.

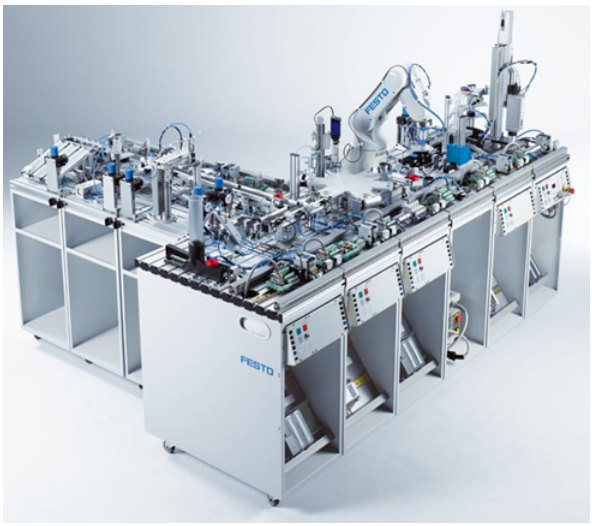


Figura 6: Estações - MPS
(POLA, 2013)

As estações modeladas para simulação neste presente trabalho serão apresentadas conforme sua sequência de atuação no cenário de produção proposto (Figura 7). A produção inicia com a distribuição de peças para a linha de produção, passando por uma análise de cor e tamanho da peça, manipulação de peças para montagem de medidores em cilindros, utiliza-se um músculo pneumático para trabalhar como uma prensa pneumática e finalizar o processo de produção, e por fim, o armazenamento das peças conforme sua cor.

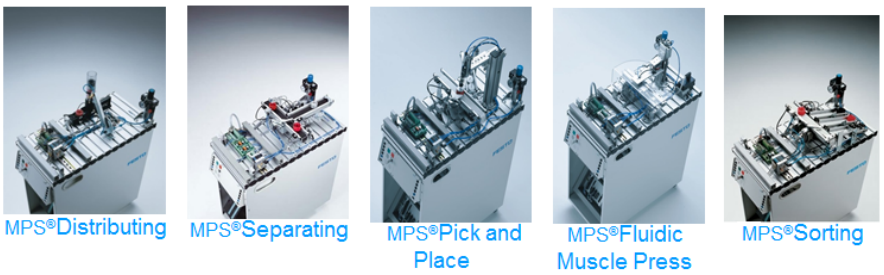


Figura 7: Estações utilizadas para simulação
(POLA, 2013)

2.2.1 ESTAÇÃO DISTRIBUTING

Esta Estação é responsável por fornecer peças de trabalho (Figura 3) à linha de produção. Dentro de seu *magazine* pode ser armazenadas até nove peças. Este módulo ainda conta com um cilindro pneumático de dupla ação, para o posicionamento das peças em um local determinado, de modo que um atuador giratório possa pegar as peças, por meio de uma ventosa para a sucção de peças, e posicionar a mesma na próxima estação.

PRINCIPAIS COMPONENTES:

A estação *Distributing* é formada principalmente por um módulo magazine, sensor óptico de barreira, e módulo atuador giratório. A Figura 8 apresenta um modelo gráfico da estação apontando os principais componentes.

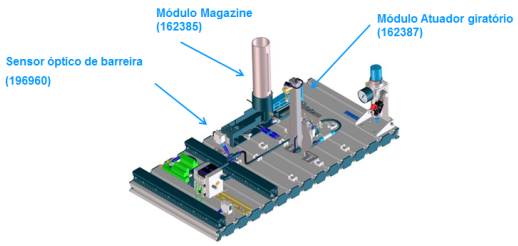


Figura 8: Estação *Distributing* - MPS
(POLA, 2013)

O módulo magazine (Figura 9) é responsável por disponibilizar as peças de trabalho, camisa de cilindro e corpo de medidor, ao sistema. Ele possui um cilindro de dupla ação instalado na parte inferior, responsável por empurrar as peças para que sejam distribuídas. Dois sensores magnéticos instalados no magazine detectam o fim de curso de atuação do cilindro, e um sensor de barreira é utilizado para a detecção de presença de peças.

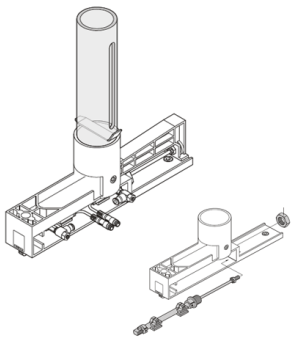


Figura 9: Módulo Magazine
(POLA, 2013)

O módulo atuador giratório (Figura 10) é responsável pelo transporte de peças de trabalho para a estação posterior, sendo as peças apanhadas por meio de sistema a vácuo e transferidas por meio de um atuador semi-rotativo pneumático. O qual possui range de giro de 0 a 180, limitado por um fim de curso mecânico, e possui duas chaves eletromecânicas de fim de curso para a detecção das posições extremas de atuação.

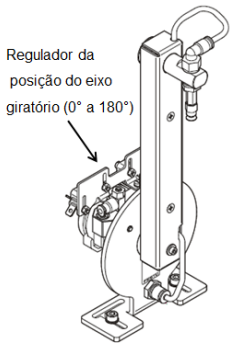


Figura 10: Módulo Atuador Giratório
(POLA, 2013)

O sensor óptico de barreira (Figura 11) é utilizado para detecção de presença de peça quando o feixe de luz é interrompido.

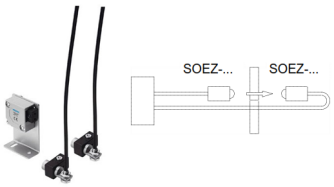


Figura 11: Sensor Óptico de Barreira
(POLA, 2013)

Uma das peculiaridades deste módulo é impedir determinadas situações, deve-se tomar cuidado para que o atuador linear não avance com uma peça enquanto o atuador giratório esteja na posição magazine. Caso ocorra tal cenário os atuadores poderiam se danificar, logo, é importante que seja bloqueada a ação de avançar o atuador.

O caminho ideal desejado para que a peça realize o processo neste módulo é: ao entrar peça no *buffer*, o atuador linear deve avançar (desde que o at. giratório esteja na pos. de próx. estação) levando a peça para uma posição intermediário de transporte, logo após, o atuador giratório volta para estação e é ligado o sopro para fixar a peça no mesmo, então o at. giratório deve ir para próxima estação e liberar a peça desligando o vácuo e ligando o sopro.

2.2.2 ESTAÇÃO SORTING

A estação *Sorting* realiza a classificação das peças (separando por tipo ou por cor). Um sensor óptico indutivo é responsável pela classificação das peças. Logo após a classificação, as peças são separadas em três diferentes rampas, de acordo com os critérios estabelecidos na programação.

PRINCIPAIS COMPONENTES:

A estação é formada principalmente por um Módulo atuador de parada, sensor óptico difuso, sensor indutivo, sensor retro reflexivo, esteira transportadora, e módulo desviador. A Figura 12 apresenta um modelo gráfico da estação apontando os principais componentes.

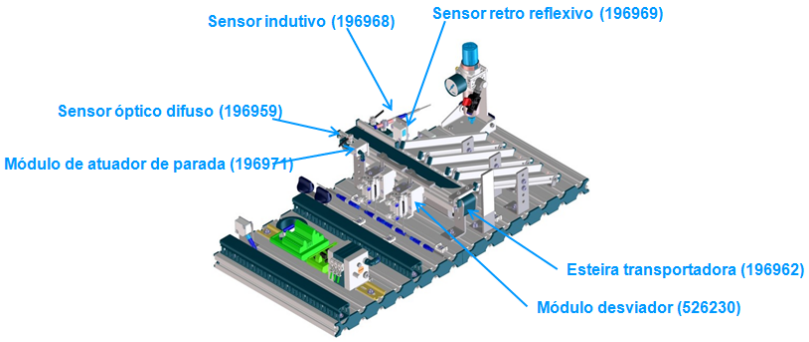
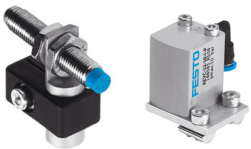


Figura 12: Estação *Sorting* - MPS
(POLA, 2013)

Sendo o sensor indutivo utilizado para detectar presença de peças metálicas, e o módulo atuador de parada utilizado para realizar a parada e liberação de peças na esteira (Figura 13).



(a) S.I. (b) M.A.P.

Figura 13: (a) Sensor Indutivo e (b) Módulo Atuador de Parada
(POLA, 2013)

Comandado por atuador linear pneumático, o Módulo Desviador (Figura 14) é utilizado para realizar o desvio de peças para as rampas. Este dispositivo é específico para liberação de apenas uma peça por vez.

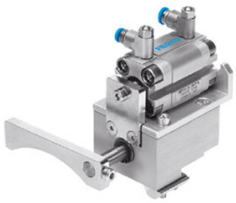


Figura 14: Módulo Desviador
(POLA, 2013)

Este módulo não apresenta situação de risco para seus atuadores, ele deve classificar e separar as peças em diferentes rampas. O ideal é que esta estação consiga otimizar seu tempo de classificação sem comprometer o armazenamento das peças em seus respectivos locais. Logo, ao entrar determinada peça na estação, a esteira deve ser ligada para que a peça se dirija até a posição de classificação e que após identificada seja liberada para sua respectiva rampa.

2.2.3 ESTAÇÃO SEPARATING

A ideia deste módulo é realizar a separação de peças referente a profundidade do orifício, ou na altura da peça. Dependendo das características da peça, será decidido se a mesma irá seguir pela esteira transportadora 1 ou pela 2, através de uma configuração prévia no sistema.

PRINCIPAIS COMPONENTES:

A estação *Separating* é formada principalmente por duas esteiras transportadoras, um sensor óptico Difuso, sensor óptico analógico, dois sensores ópticos de barreira, e um módulo desviador. A Figura 15 apresenta um modelo gráfico da estação apontando os principais componentes.

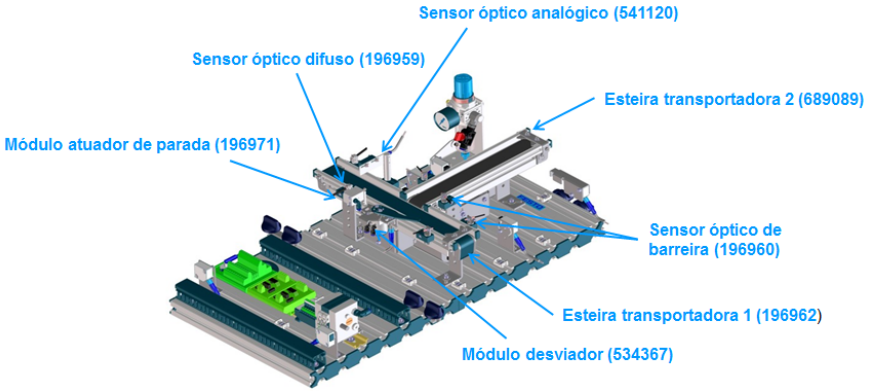


Figura 15: Estação *Separating* - MPS
(POLA, 2013)

Inicialmente a peça é movida até um atuador de parada, ao chegar na posição de medição, o sensor óptico analógico (Figura 16) é utilizado para a medição de profundidade do orifício e altura da peça.

Figura 16: Sensor Óptico Analógico
(POLA, 2013)

Determinado as características da peça, o atuador de parada libera a passagem para que o módulo desviador possa desviar, caso necessário, o fluxo de peças de uma esteira para outra. O grande desafio para a programação deste módulo é otimizar o processo de separação, garantindo que as esteiras não estejam ligadas sem necessidade e que a classificação das peças possa ser em fluxo contínuo.

2.2.4 ESTAÇÃO PICK AND PLACE

A tarefa desta estação é manipular peças do tipo medidores (higrômetro, termômetro e relógio). Quando o corpo de medidor esta na posição do atuador de parada, o manipulador *Pick&Place* realiza a manipulação do medidor por meio de um sistema de vácuo, e realiza a montagem.

PRINCIPAIS COMPONENTES:

A estação *Pick and Place* é formada principalmente por um módulo *Pick&Place*, sensor óptico de barreira, módulo de parada elétrico, dois sensores ópticos difuso, e uma esteira transportadora. A Figura 17 apresenta um modelo gráfico da estação apontando os principais componentes.

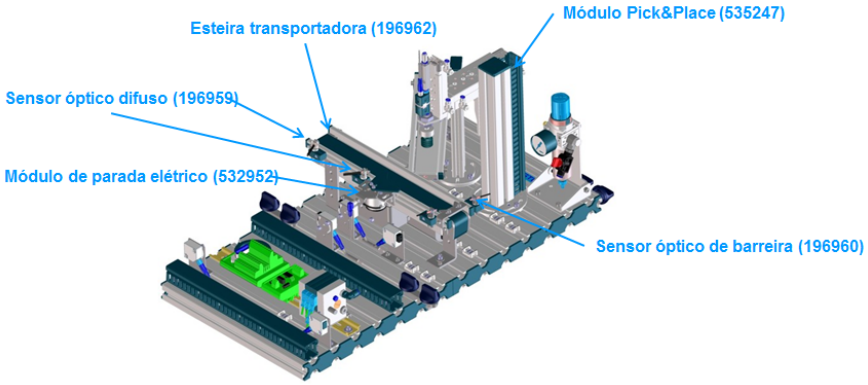


Figura 17: Estação *Pick and Place*- MPS
(POLA, 2013)

Utilizado para realizar a parada e a liberação de peças na esteira, o módulo de parada elétrico (Figura 18) conta com um atuador elétrico com giro de até 90º. Sendo um dispositivo específico para a liberação de apenas uma peça por vez.

Figura 18: Módulo de Parada Elétrico
(POLA, 2013)

Outro importante componente nesta estação, é o módulo *Pick&Place* (Figura 19). Ele é utilizado para manipular os medidores da rampa através de sistema de vácuo e transportá-los até a "peça corpo" posicionada na esteira. Sendo seus principais componentes: vacuostato, ventosa, geradora de vácuo, atuador pneumático no eixo Z e X.

Figura 19: Módulo *Pick&Place*
(POLA, 2013)

Esta estação é mais uma que possui situações indesejadas as quais o programador deve impedir que ocorram, garantindo que os atuadores não se danifiquem. O módulo *Pick&Place* não pode avançar ou recuar caso esteja na posição inferior por exemplo

(cenário de risco), o vácuo não deve ser desligado enquanto o módulo esteja na posição superior (risco de perder a peça do tipo medidor no transporte).

O processo desejado contém os seguintes passos: ao entrar uma peça do tipo corpo medidor na estação, a esteira deve ser ligada até a peça se encontrar na posição de receber um medidor, é necessário que o módulo *Pick&Place* desça e busque um medidor no *buffer*, para realizar o transporte então é necessário ligar vácuo, elevar o módulo, avançar o mesmo, e desce-lo para ir de encontra a peça corpo de medidor para então desligar o vácuo e liberar a peça (recuando o atuador de parada e elevando *Pick&Place*).

2.2.5 ESTAÇÃO FLUIDIC MUSCLE PRESS

O principal fundamento desta estação, é a utilização da tecnologia de músculo pneumático. A estação *Fluidic Muscle Press* é utilizada como prensa pneumática, sendo possível realizar o controle de pressão exercido pelo músculo pneumático.

PRINCIPAIS COMPONENTES:

A estação é formada principalmente por um módulo *Fluidic Muscle Press*, sensor difuso, módulo linear rotativo, reguladora de pressão proporcional. A Figura 20 apresenta um modelo gráfico da estação apontando os principais componentes.

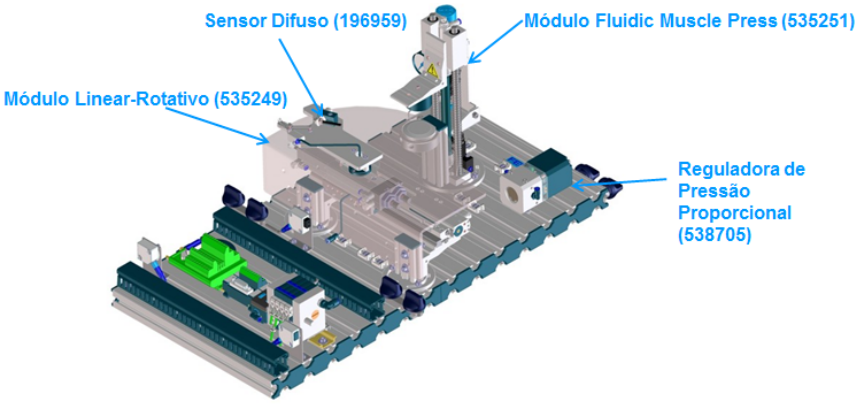


Figura 20: Estação *Fluidic Muscle Press*- MPS
(POLA, 2013)

Módulo *Fluidic Muscle Press* (Figura 21) responsável por prensar a peça "tipo medidor" na peça "corpo", realiza esta função através de um atuador pneumático do tipo músculo. Sendo seus principais componentes: válvula reguladora de fluxo, válvula reguladora de pressão, e o atuador pneumático do tipo músculo.

Figura 21: Módulo *Fluidic Muscle Press*
(POLA, 2013)

A Reguladora de Pressão Proporcional (Figura 22) é responsável por controlar a pressão do módulo prensa. Suas principais características são: Range da pressão de operação (0,15 a 6 bar), e setpoint (0-10V).

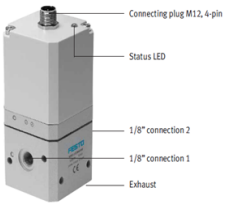


Figura 22: Reguladora de Pressão Proporcional
(POLA, 2013)

O transporte da peça para a posição de prensa, e/ou próxima estação, fica a cargo do Módulo Linear-Rotativo (Figura 23). Este módulo é composto por: Um módulo garra com sensor óptico difuso acoplado, módulo linear com curso de 100 mm - 900 mm e sensores magnético de fim de curso, e um módulo rotativo com sensores magnéticos para identificação de posicionamento.

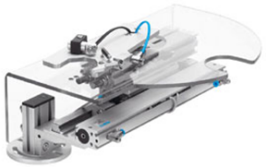


Figura 23: Módulo Linear-Rotativo
(POLA, 2013)

Deve-se tomar cuidado para que o módulo linear rotativo não altere seu estado enquanto o módulo *fluidic muscle press* esteja na posição inferior. Caso ocorra tal

cenário os atuadores poderiam se danificar, logo, é importante que seja bloqueada qualquer ação de rotação do atuador.

O caminho ideal desejado para que a peça realize o processo neste módulo é: ao entrar peça na estação, o atuador linear rotativo deve fechar a garra, levar a peça para a posição de prensa, logo após, o atuador *fluidic muscle press* é ligado para fixar as peças, então o módulo *fluidic* deve ser elevado para que o atuador linear rotativo leve a peça para posição de próxima estação e então liberar a peça avançando o atuador e abrindo a garra.

3 SISTEMAS A EVENTOS DISCRETOS - AUTOMATA

Na teoria tradicional de controle, existem muitos métodos bem-sucedidos de modelagem que são adequadas para sistemas de tempo contínuos (CTS), e sistemas de tempo discreto (DTS). No entanto, geralmente, estas metodologias não são utilizáveis para resolver problemas relacionados com a modelagem e controle de sistemas dinâmicos de eventos discretos, ou *DEDS - discrete event of dynamic systems* cujo comportamento dinâmico depende da ocorrência de uma sequência de ações. DEDS são sistemas assíncronos, com muitas situações de conflito e com alto paralelismo entre as atividades dos diversos subsistemas (CASSANDRAS, 2008). Um processo típico de um DEDS variável x é dada na Figura 24 onde pode-se observar que não existe uma marcação temporal no eixo horizontal, mas apenas a sequência dos eventos $\{e_0,e_1,e_2,e_3,e_4,e_5\}$ com a sequência correspondente dos valores das variáveis discretas $\{x_0,x_1,x_2,x_3,x_4,x_5\}$ no eixo vertical representando respostas sobre os eventos acontecidos. Outro ponto importante no gráfico é que a combinação dos valores das variáveis geram os estados do sistema. Em e_0 o estado é $E_1 = \{1,0,0,1,0,0\}$, na ocorrência do evento e_3 o estado é $E_3 = \{0,1,1,1,0,1\}$.

DEDS são usados com muita frequência em sistemas de grande escala e / ou sistemas complexos. O controle neste tipo de sistema é definido como um conjunto de ações ou eventos que se fazem necessários para conduzir o sistema de um dado estado inicial em um prescrito, atendendo simultaneamente critérios como restrições, condições

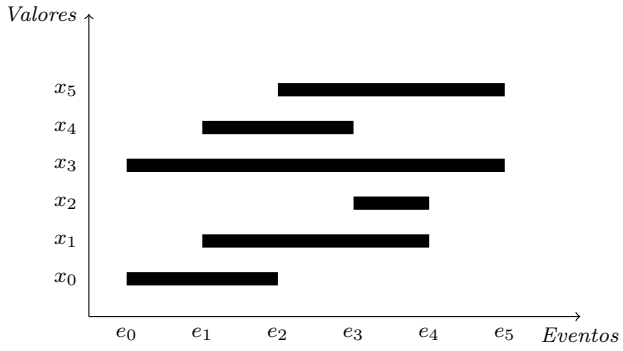


Figura 24: Comportamento dos Eventos e Variáveis em um SED

externas, etc impostas ao sistema em análise.

Exemplos típicos são sistemas de manufatura, transporte e de comunicação (incluindo os processos de computacionais). DEDS em geral serão estudados aqui com o objetivo de desenvolver modelos analíticos mais adequados para fins de controle, especialmente métodos baseados em autômata e Redes de Petri.

As técnicas neste texto utilizadas para cumprir estes objetivos são autômatos e redes de Petri, com aplicações focadas em sistemas flexíveis de manufatura.

Uma observação importante relacionado a este texto é que algumas modificações foram propostas nos conceitos originais para um melhor desempenho em sistemas de manufatura. Desta forma, tanto a modelagem tradicional, desenvolvida para ciência da computação, quanto a proposta serão indicadas para o leitor ter uma maior clareza do que é o descrito na literatura tradicional e o que é proposto aqui.

3.1 INTRODUÇÃO

Uma das ferramentas de modelagem mais populares para SED representação é autômato. No texto a seguir, damos uma descrição resumida das notações básicas da teoria dos autômatos.

Um *automato* A é definido por 6 elementos (CASSANDRAS, 2008)

$$A = \{X, E, f, \Gamma, x_0, X_m\} \tag{3.1}$$

onde

X	-	é o conjunto de estados presentes no sistema,
E	-	é o conjunto de eventos que geram transições nos estados,
$f : X \times E \to X$	-	é a *função de transição*
$\Gamma : X \to 2^E$	-	é o conjunto de eventos ativos, logo $\Gamma(x)$ é o conjunto de eventos ativos definidos em $f(x,e)$. Ou seja, todos os eventos que podem ser disparados a partir do estado x.
x_0	-	é o estado inicial e
X_m	-	é o conjunto de estados marcados.

Um estado pode ser definido individualmente como a situação de um dado elemento do sistema como mostra o item (3.2) onde é definido o estado do atuador linear como recuado, ou de forma coletiva onde todo o conjunto de variáveis que definem a posição do sistema como mostra o item (3.3) que mostra o estado inicial desejado da estação de distribuição.

$$X_{AL} = \{RC\} \tag{3.2}$$

$$X_{SISTEMA} = \{AL_{RC}, MG_{MG}, VC_{DS}, SP_{DS}, PC_{NOT}\} \tag{3.3}$$

Sendo:

- AL_{RC}: Atuador linear recuado
- MG_{MG}: Módulo giratório posição magazine
- VC_{DS}: Atuador de vácuo desligado
- SP_{DS}: Atuador de sopro desligado
- PC_{NOT}: Sensor de peça indicando que não há peça no magazine

Em muitos casos (especialmente quando se trata de sua aplicação prática) E e X apresentam um número finito de elementos. A função de transição f descreve o mapeamento entre estes dois conjuntos da seguinte maneira: se existir um evento e que gera transição do estado x para o estado y, então $f(x, e) = y$. Se, após a ocorrência do evento e o estado do sistema x não muda escrevemos $f(x, e) = x$. Quando $f(x, a) = y$ e $f(y, b) = z$ temos

$$f(y, b) = f(f(x, a), b) = f(x, ab) = z \tag{3.4}$$

que graficamente é mostrado de acordo com a figura 25

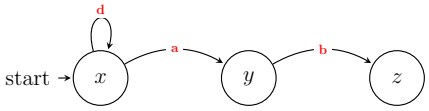


Figura 25: Grafo representando $f(y, b) = f(f(x, a), b) = f(x, ab) = z$ e $f(x,d) = x$

isto é, a definição de função de transição é estendido geralmente para o conjunto de sequências, denotado E^*. Dado que o conjunto de eventos que fazem com que as transições de estado x é geralmente um subconjunto de E, $\Gamma(x) \subset E$, é evidente que a função de transição f existe apenas em parte do seu domínio ($\Gamma(x)$ é chamada de função de eventos ativos e é uma parte da definição do autômato). Assim, $f(x, e)$ não deve ser definido para cada caso e em cada estado x. O conjunto de estados marcados, X_m, é um subconjunto de X e, em geral, é capaz de apontar que alguns estados que têm um significado especial. Um estado marcado pode ser associado a um estado final ou a algum estado obrigatório que o sistema deva passar antes de finalizar sua tarefa. Por exemplo, em um sistema de engarrafamento, após o vasilhame chegar na posição correta, este deve ser fechado através de um acionamento, dando origem a um estado e_i onde a garrafa está no ponto k_i do processo e já devidamente lacrada. Pular este estado gera um produto incompleto, desta forma e_i deve ser um estado marcado.

Considerando o item 3.1, pode-se modelar o sistema do atuador linear apresentado em 2.2.1 como sendo

$$A_{al} = \{E_{al}, X_{al}, f_{al}, x_{0al}, X_{mal}\} \tag{3.5}$$

onde os elementos são definidos por:

$$E_{al} = \{Avancar, Recuar\} = \{AV, RC\} \tag{3.6}$$
$$X_{al} = \{ALAvancado, ALRecuado\} = \{A, R\} \tag{3.7}$$
$$f_{al}(A, AV) = A \tag{3.8}$$
$$f_{al}(R, AV) = A \tag{3.9}$$
$$f_{al}(A, RC) = R \tag{3.10}$$
$$f_{al}(R, RC) = R \tag{3.11}$$
$$x_{0al} = R \tag{3.12}$$
$$X_{mal} = R \tag{3.13}$$

Inicialmente pode-se perceber que a função de transição f_{AL} foi definida em todo o domínio, desta forma está relacionada com cada estado de X_{al}. O estado marcado $X_{mal} = A$ define que o sistema tem que passar por este estado em particular. Esta condição está associada com o fato de que, para empurrar uma peça do magazine para

o buffer intermediário é necessário que o atuador linear seja acionado. Pode-se perceber que esta representação é complexa e que, para sistemas de grande porte, sua visualização fica comprometida. Desta forma, a exemplo da Figura 25, sua representação em diagrama de transição de estado é mostrada na Figura 26;

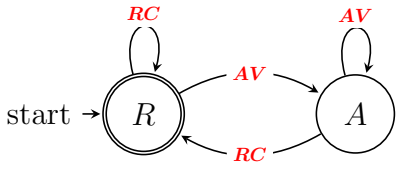


Figura 26: Automato Completo do Atuador Linear Festo Distribuiting.

Apesar de correto, muitas vezes adicionar transições que não geram mudança de estado podem complicar a representatividade, desta forma uma forma otimizada de representar este sistema seria basicamente o diagrama da Figura 27;

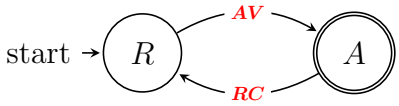


Figura 27: Automato Resumido do Atuador Linear Festo Distribuiting.

Algumas observações sobre o conteúdo já apresentado.

- Um autômato é comumente denominado de máquina de estado ou gerador.
- Se X é um conjunto finito, chamamos A um autômato de estados finitos determinístico.
- As funções f e Γ estão completamente descritas no diagrama de transição de estado da autômato.
- O autômato é dito ser determinístico, pois $f(x,e) \rightarrow x_1$ possuiu uma saída única, ou seja, não pode haver duas transições com o mesmo rótulo de evento saindo de um estado. Em contraste, a estrutura de transição de um autômato é dita como não determinista se existe $f(x,e) \rightarrow x_1$ e $f(x,e) \rightarrow x_2$, ou seja, para o mesmo estado um evento pode levar a dois estados diferentes.
- A seleção dos estados marcados é uma questão de modelagem que depende do problema de interesse. Ao designar certos estados como marcados, define-se , por exemplo, que o sistema completou alguma operação ou tarefa. Em sistemas

complexos o autômato deve passar por todos os estados marcados antes de determinar o fim da execução de um ciclo. Esta definição difere do conceito original de autômato proposto em ciência da computação. Para uma representação mais próxima do original, poderia definir o autômato como sendo formado por 7 elementos; os seis descritos anteriormente e um novo X_t como sendo o conjunto de estados que o sistema deve passar antes de ir para o estado marcado, que indica a finalização da execução do autômato.

- uso de transições que não geram mudanças de estado é fundamental para o sincronismo e integração de múltiplos autômatos.

3.2 BLOQUEIO E SEGURANÇA EM AUTÔMATOS

Existem dois tipos de situações muito indesejadas em sistemas flexíveis de automação; os bloqueios mortal e vivo. Nestas situações o sistema fica preso em estados que não têm caminho possível para alcançar o estado marcado, ou seja, ele inicia uma tarefa e não finaliza e nem é capaz de voltar para o estado inicial. De forma prática, o sistema fica inoperante e necessita ser reiniciado manualmente. Situações como esta são consequência de um erro do projeto físico ou de má especificação na forma de se trabalhar os recursos existentes. É muito comum em processos concorrentes que uma parte da linha esteja utilizando um recurso A e esperando um recurso B para finalizar a tarefa corrente. Caso exista outro processo que esteja em situação oposta, ou seja, segurando o recurso B e esperando o A para finalizar sua tarefa, o sistema fica bloqueado.

A literatura apresenta diversos outros exemplos de casos como os sitados, talvez um dos mais conhecidos é o dos filósofos glutões que será discutido futuramente. Entretanto, como uma forma mais simples de apresentar o problema, considere o autômato A mostrado na figura 28. Caso a linguagem $\lambda(A)$ apresente a *substring* {3}, então o autômato fica necessariamente em um bloqueio,que pode ser vivo ou mortal. Caso a linguagem apresente a *substring* {4}, o bloqueio é mortal pois o autômato fica preso no estado {E}. Caso esta *substring* não esteja presente na linguagem, o autômato pode ficar alternando entre os estados {D} e {F} indefinidamente desde que os eventos {5} e {6} fiquem sendo disparados.

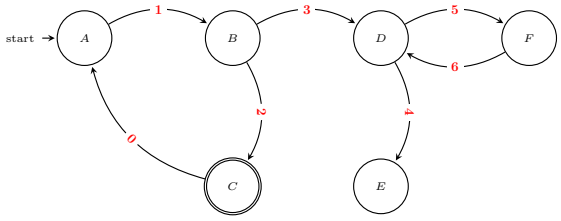


Figura 28: Automato com presença de bloqueio mortal e bloqueio vivo

DEFINIÇÃO DE BLOQUEIO

Pode-se perceber pelo exemplo anterior que a linguagem marcada $\lambda_m(A)$ é formada pela combinação apenas das *strings* {1,2,3}. Qualquer outro subconjunto que apareça na linguagem, esta deixa de ser marcada e passa a ser $\lambda(A)$.

Desta forma, pode-se definir matematicamente que, caso o sistema apresente um bloqueio, então tem-se

$$\lambda_m(A) \subset \lambda(A) \tag{3.14}$$

e, caso o sistema seja livre de bloqueio, então tem-se

$$\lambda_m(A) = \lambda(A) \tag{3.15}$$

Outro tipo de preocupação em sistemas de manufatura é a segurança. Este fator é ainda mais crítico quando recursos são compartilhados entre diversas linhas de produção.

DEFINIÇÃO DE SEGURANÇA

De uma forma mais formal, a segurança é definida como o estudo de propriedades que estão que levam o sistema a ter acessibilidade à certos estados indesejáveis no autômato. A presença de determinadas sequências de comandos gerada pelo autômato pode levar em uma situação de risco à manufatura. A identificação desta sequência de comandos, representada por uma *substring* dentro da linguagem definida pelo autômato, não é trivial por dois motivos; primeiro, normalmente a construção de um sistema complexo envolve na modelagem de sistemas menores e sua composição paralela, o que gera o segundo motivo que é a existência de estados matematicamente e até fisicamente plausíveis, mais que geram contingências na planta. Como exemplo, considere a planta

de distribuição; acionar o atuador linear com o módulo giratório na posição magazine é uma possibilidade tanto matemática quanto física, entretanto não é uma possibilidade sistêmica uma vez que isso danificaria a garra do módulo giratório. Entretanto o estado que esta ação leva, atuador avançado e módulo na posição magazine, é um estado válido, pois caso o atuador já esteja avançado é possível ir com o módulo para a posição desejada. Então neste caso o estado é valido, apenas como é feita a transição de um estado para outro que deve ser monitorada. Como o conjunto de transições parte de um estado inicial, o que deve ser feito é monitorar a linguagem gerada pelo autômato, retirar a parte crítica e reconstruir o autômato. Como exemplo de análise, partindo de x_0 a string $s = \{Av\}$ não pode ser permitida como preâmbulo de nenhuma linguagem.

Desta forma, para proceder com a segurança de um autômato, os seguintes passos deves ser realizados;

- Para determinar se um estado y é alcançável a partir de x, realiza-se a o operação acessível *Ac* utilizando apenas uma string, desta forma declara-se x estado inicial e verifica-se se em alguma parte do autômato y é o estado posterior. Caso o resultado for positivo, o caminho que leva a esta sequência de estados não é válida. Para facilitar a visualização, considere, para o exemplo anterior onde a sequência de eventos procurada é $x = \{R,M\}$ e $y = \{A,M\}$.

- Para determinar se uma determinada substring é possível no autômato, simula-se a substring de todos os estados acessíveis na autômato; isso é feito facilmente quando o diagrama de transição de estado é representado como uma lista encadeada.

3.3 COMPOSIÇÃO PARALELA DE AUTÔMATOS

De uma forma geral, a maneira de se construir autômatos de sistemas complexos é através da modelagem e união de seus susbsistemas. Esta união é realizada por uma operação denominada *Composição* de autômatos (CASSANDRAS, 2008). Basicamente, cria um novo espaço de estados através da combinação linear dos estados de cada subsistema e busca sincronizar os eventos de forma que exista coesão entre os novos estados criados.

De uma forma mais matemática, as equações 3.16 e 3.17 mostram como é feita a composição de dois autômatos.

$$A_1 \parallel A_2 = A_c \left(X_1 \times X_2, E_1 \cup E_2, f((x_1x_2),e),\Gamma_{1||2},(x_{01}x_{02}), X_{m1} \times X_{m2}\right) \tag{3.16}$$

$$f((x_1x_2),e) = \begin{cases} (f_1(x_1,e)f_2(x_2,e)) & if\ e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1,e)x_2) & if\ e \in \Gamma_1(x_1) \setminus E_2 \\ (x_1f_2(x_2,e)) & if\ e \in \Gamma_2(x_2) \setminus E_1 \end{cases} \tag{3.17}$$

Em outras palavras, um evento que pertença a ambos autômatos pode ser executado somente quando o autômato conjunto chega no estado que é formado pelos estados que precedem o evento nos autômatos originais. Outros eventos podem ser executados sem qualquer restrição. Explicando ainda de outra forma, se um evento existir nos dois autômatos, ele só poderá ser disparado se ele suceder os estados correntes de ambos autômatos. Esta regra é de grande importância para impor restrições de atuação no sistema.

Como um exemplo sem as restrições, considere o autômato do *Atuador Linear* mostrado anteriormente na Figura 27 e repetido na figura 29.a por comodidade do leitor e o autômato do *Módulo Giratório* mostrado na Figura 29.b.

A composição destes dois autômatos começa com o produto dos estados de cada um, gerando uma nova lista de estados $X = \{(R,M),(R,E),(A,E),(A,M)\}$ onde os estados M, E significam que o módulo giratório está na posição *magazine* ou *estação*, respectivamente.

O conjunto de eventos é a soma dos eventos de cada autômato, desta forma $E = \{AV,RC,G_e,G_m\}$ onde os eventos G_m e G_e significam girar o módulo para o magazine ou para a estação respectivamente.

Para definição das funções de ativação f, a equação 3.17 deve ser utilizada. Desta forma, para o novo estado $\{R,M\}$ apenas eventos que originalmente eram disparados a partir de $\{R\}$ ou $\{M\}$ podem ser delegados. Desta forma, para este novo estado, são criadas duas funções de ativação $f_1(\{R,M\},AV) = \{A,M\}$ e $f_2(\{R,M\},G_e) = \{R,E\}$. Os outros estados seguem a mesma lógica. Uma visualização gráfica deste exemplo se apresenta na figura 30.

Verifica-se nesta figura que uma função de transição, nomeadamente $f_n(\{R,M\},AV) = \{A,M\}$ (marcada de vermelho) gera uma situação de contingência no sistema e não deveria estar presente no autômato.

(a) Autômato do Atuador Linear (b) Autômato do Módulo Giratório

Figura 29: Autômatos do Atuador Linear e Módulo Giratório

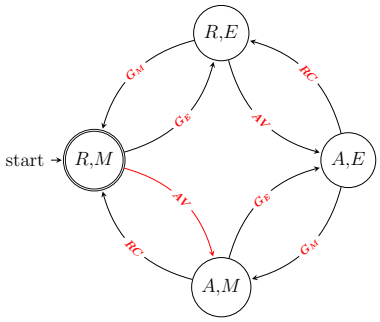


Figura 30: Automato do composto dos Automatos MG e AL da Festo Distribuiting

3.4 METODOLOGIA PARA O PROJETO DE AUTÔMATOS

Se dois subprocessos forem modelados independentemente, como no caso do atuador linear e do módulo giratório do exemplo anterior pode gerar situações indesejadas no autômato resultante. Desta forma, após uma primeira análise de cada módulo, seu comportamento no conjunto deve ser analisado e, caso necessário, seu *design* deve ser alterado.

Neste caso, não é possível ter o evento AV caso o atuador linear esteja recuado, entretanto não existe nenhum empecilho para que esta ação não seja executada se forem considerados os autômatos originais. Desta forma é necessário mudar o *design* original. Bloquear o evento em um estado significa que este evento deve estar no autômato analisado e não estar saindo do estado que irá gerar a contingência. Ou seja, o bloqueio é feito através da permissão do evento em outros estados do autômato.

Desta forma, adicionar o evento de avançar o atuador linear AV apenas no estado onde o módulo giratório se encontra na próxima estação (E), coloca uma restrição de quando este evento pode ser disparado após os autômatos serem interconectados. A Figura 31 mostra o grafo do autômato do módulo giratório com o evento AV sendo adicionado. Em um primeiro momento parece que tal adição é inócua, uma vez que sua ativação não causa mudança de estado no referido autômato. Entretanto, ao unir os dois autômatos em um único grafo como mostra a figura 30, a ativação do evento AV só poderá ocorrer quando o módulo giratório estiver na posição da próxima estação, impedindo de danificar o sistema. Outra observação importante é que, analisando os autômatos juntos ou separados, pode-se observar que o estado inicial já é o estado marcado, o que indica que estes autômatos não necessitariam fazer nenhuma ação para chegar no objetivo pré-estabelecido.

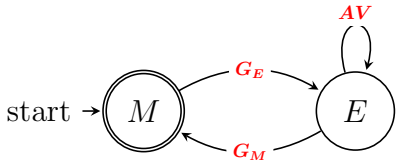


Figura 31: Automato do MG com adição de AV

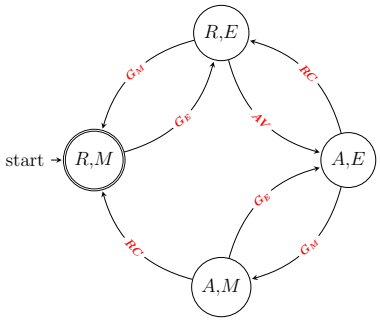


Figura 32: Automato do composto dos Automatos MG e AL da Festo Distribuiting

Considere agora como exemplo a figura 33 onde é desenhado o autômato de evolução das peças na célula. Onde os estados SP, PM e PE significam *sem peça*, *peça no magazine* e *peça na estação*, respectivamente. Os eventos Cr e DCr são, respectivamente carregar e descarregar peças do módulo. Neste caso, são eventos externos ao controle da planta em questão. O evento $AV \circ Tr$ é a composição do evento *Avançar* com o novo evento definido como *Transportar* que é a união de *ligar o vácuo, testar o vacuostato, mover o atuador giratório para o posição da próxima estação, desligar o vácuo, ligar o sopro, testar o vacuostato e desligar o sopro*. Neste ponto, duas considerações são importantes;

- a sequência grande de eventos contida em Tr é chamada de encapsulamento seletivo, ou seja, quando um conjunto de ações sempre será realizado da mesma forma e seguindo a mesma ordem, compõem-se todas em uma única ação. Isso simplifica o modelo e garante uma maior agilidade na modelagem. Neste caso, o evento AV poderia estar junto neste conjunto, mas por motivos explicados à frente, optou-se por deixá-lo aparte.
- O evento composto $\{AV \circ Tr\}$ significa que para que haja mudança de estado as duas ações AV e Tr devem ser executadas de forma sequencial, primeiro a AV e depois a Tr.

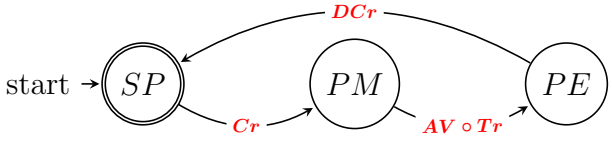


Figura 33: Automato da evolução de peças no módulo Festo Distribuiting

A execução do evento $\{AV \circ Tr\}$ gera apenas uma mudança de estado observável no sistema, entretanto existe um estado intermediário que não pode ser medido por

sensores, apenas infere que este existe pela consequência de uma data ação sobre um estado observável. Neste caso, não existe um sensor direto da presença da peça no fim do atuador linear, ou seja, após a peça sair do magazine, o próximo sensor seria o vacuostato após uma série de manobras. Entretanto pode-se inferir que, se existe peça no magazine e se o atuador linear empurrou esta peça para frente, a peça está na posição de ser capturada pelo atuador giratório. Ou seja, a peça saiu do magazine e fica armazenada em um buffer intermediário à espera da continuidade do processo.

Desmembrando este novo estado no autômato da evolução de peças mostrado na figura 33, gera um novo automato mostrado na figura 34. Outra modificação mostrada na figura já é em virtude do sincronismo com os outros autômatos, onde só é permitido o evento de girar o módulo giratório para a estação caso tenha peça no magazine.

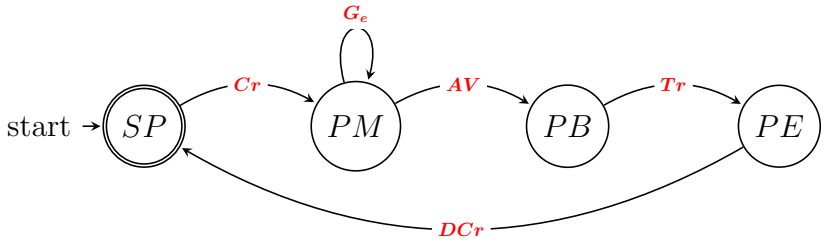


Figura 34: Automato da evolução de peças com estado extra

Como consequência da adição deste novo autômato com um novo conjunto de eventos, os autômatos do atuador linear e da mesa giratória foram adaptados segundo mostram as figuras 35 e 36. Para não correr o risco do atuador linear prender a peça caso este esteja avançado, o evento $\{Tr\}$ só pode ocorrer quando este estiver na posição $\{R\}$. Já no caso do módulo giratório, como o evento $\{G_E \in Tr\}$, $\{Tr\}$ também gera a mudança de estado da posição do magazine para a próxima estação. Neste caso $\{G_E$,$Tr\}$ significa que um ou outro evento podem ocorrer de forma independente, mas que ambos acarretam na mudança de estado indicada. Finalmente, por restrição da próxima estação, só é possível descarregar a peça caso o atuador giratório esteja na posição do magazine.

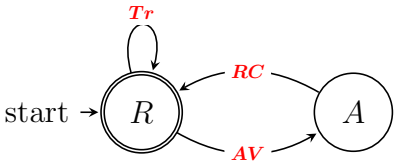


Figura 35: Automato do AL considerando transporte de peças

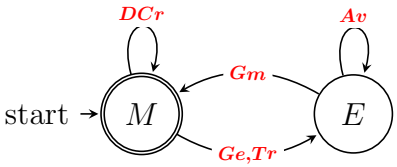


Figura 36: Automato do AG considerando transporte de peças

A composição dos autômatos da localização das peças, do atuador linear e do módulo giratório é mostrado na figura 37. Percebe-se que o sistema se encontra em repouso, uma vez que o estado marcado e o inicial são o mesmo. Uma vez que o evento $\{Cr\}$ é realizado (nota-se que este evento não é controlável, para este autômato seria um distúrbio), o sistema tenta entrar em repouso novamente.

No autômato resultante aparecem diversos estados que não pertencem ao caminho mínimo do sistema entre o estado de distúrbio após $\{Cr\}$ e o estado marcado $\{R,M,SP\}$. Um dos possíveis caminhos é mostrado na figura onde os estados estão marcados com a cor *verde claro* e os eventos sinalizados em *vermelho*. Nota-se que apenas 8 estados e 8 eventos foram necessários em relação aos 16 eventos e 28 eventos originais.

A minimização completa do diagrama de estados do autômato é mostrada na Figura 38, onde mostra o grafo de alcançabilidade sistêmica, ou seja, todos os estados que podem ser alcançados pelo sistema operando situação normal e sem contingência.

Nota-se que este grafo, apesar de mostrar de um funcionamento possível e correto da célula de manufatura, ainda não está otimizado por dois motivos;

- quando o sistema está no estado $\{R,E,PE\}$ o próximo passo segundo o grafo seria retornar com o módulo giratório para a posição do magazine. Entretanto em uma análise simples, pode-se perceber que, caso exista peça no magazine, seria mais vantajoso o sistema voltar para o estado $\{A,E,PB\}$ atuando o evento $\{AV\}$. Entretanto esta possibilidade não é factível; pela forma que o sistema foi modelado o evento $\{AV\}$ não altera o estado de $\{PE\}$ para $\{PB\}$, pois leva em consideração a existência de apenas uma ficha no processo. Modelar cada etapa da peça como uma autômato individual com estados indicando ou não a presença de ficha em cada posição da linha de montagem é uma alternativa viável, porém acarreta em um crescimento considerável do número de estados.

- Existe uma substring entre os estados $S_1 = \{A,E,PB\}$ e $S_2 = \{R,M,PB\}$ que é comutativa, ou seja, $\lambda(s_1) = \{R_r G_m\} = \lambda(s_2) = \{G_m R_r\}$. Em outras palavras,

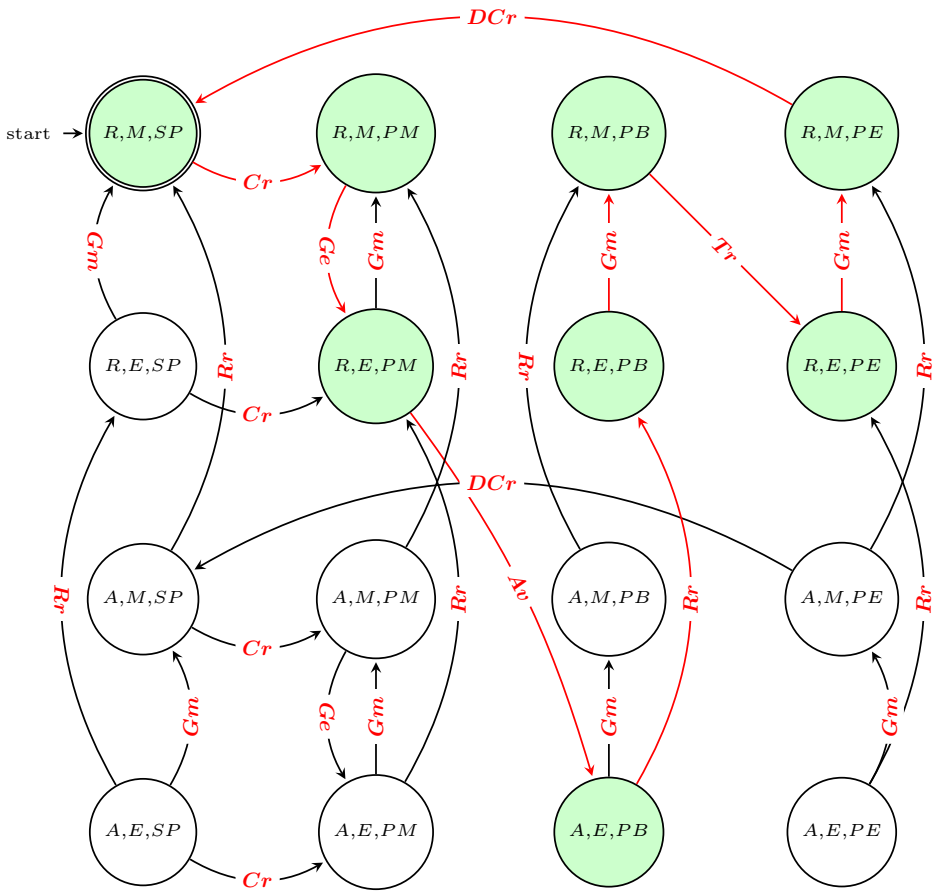


Figura 37: Automato composto PC,MG,AL

as linguagens s_1 e s_2 mapeiam a transição de estado $S_1 \rightarrow S_2$. Como os eventos não possuem restrições de ordem, poderiam ser executados em paralelo que acarretaria na mesma transição de estado, todavia em um tempo mais curto. Para tanto poderia definir um novo evento $Rr//G_m$ que indica que os eventos R_r e G_m podem ser executados em paralelo.

3.5 AUTOMATO MODIFICADO PARA SMF

Além da situações descritas como difíceis na etapa da modelagem de um sistema via autômatos (i.e. crescimento e paralelismo), existe uma outra dificuldade não apontada. Quando foi desenvolvida a teoria dos autômatos grande parte dos sistemas produtivos consideravam apenas linha de montagem, ou seja, um produto específico, tinha uma linha destinada para ele. Com o avanço tecnológico e a customização crescente dos produtos um cenário muito comum em empresas é o sistema flexível de manufatura onde uma linha de produção é responsável pela montagem de diversas peças.

Como exemplo considere a situação do sistema de separação descrito em 2.2.3, onde

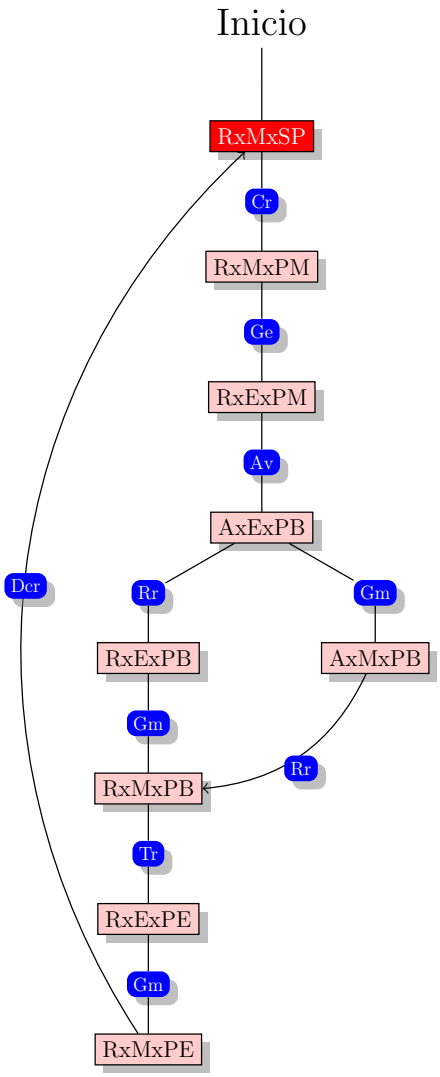


Figura 38: Grafo de alcançabilidade do Automato composto PC,MG,AL

uma peça é carregada sem saber se é corpo de medidor ou corpo de atuador pneumático. Após passar por um sensor de detecção e identificada o tipo, cada parte vai para o seu respectivo caminho na linha de produção.

Este modelo exige que exista mais de um caminho e, obviamente, mais de um estado final. Desta forma, existem passos que devem ser executados em uma linha e que não serão na outra ou vice-versa. Dentre todos os estados possíveis na execução de um caminho de um estado desejado e um final existem alguns estados que são obrigatórios dentro da linha de montagem. Fazer o caminho passar por um estado específico significa gerar um novo estado do tipo *Flag* que seria marcado após a passagem. Por exemplo, se fosse necessário realizar uma impressão em um produto, seria necessário ter a ação $E = \{Impr\}$ e um estado extra $X = \{Impresso\}$. Em um sistema produtivo real, com vários estados obrigatórios, a criação destas *Flags* aumentariam a complexidade do sistema, gerando uma explosão combinacional e, eventualmente, dificultado sua

modelagem.

Para manter a complexidade do sistema controlada, porém incorporando a capacidade de modelar sistemas flexíveis de manufatura com múltiplos caminhos com diversos estados obrigatórios a representação de autômato apresentada em 3.1 foi modificada.

Pela proposta, um *automaton* A é definido por 8 elementos

$$A = \{X,E,E_p,f,\Gamma,x_0,X_m,X_{ob}\} \tag{3.18}$$

onde

- E_p - é o conjunto de eventos que podem ser executados em paralelo. Os eventos originais se mantêm em E e uma nova etiqueta ($\{e_1//e_2\}$) é criada para representar o novo evento. A dimensão de cada elemento de E depende da quantidade de eventos que podem ocorrer em paralelo.
- X_{ob} - é um conjunto de estados obrigatórios que o autômato deve passar entre o estado inicial e o final.

e a definição de $\{X,E,f,\Gamma,x_0,X_m\}$ é dada em 3.1.

Onde, para a composição paralela entre dois autômatos, a formulação da equação 3.19 deve ser seguida:

$$A_1 \parallel A_2 = A_c \left(X_1 \times X_2, E_1 \cup E_2, g(E_{p1},E_{p2}), f((x_1x_2),e), \Gamma_{1||2}, (x_{01}x_{02}), X_{m1} \times X_{m2}, X_{ob1} \times X_{ob2}\right) \tag{3.19}$$

onde $g(E_{p1},E_{p2})$ representa uma função de composição, ou seja, analisa todas as possibilidades de paralelismo como mostra as equações 3.20 a 3.22

$$E_{cp1} = E_{p1}//E_2 \tag{3.20}$$

$$E_{cp2} = E_{p2}//E_1 \tag{3.21}$$

$$g(E_{p1},E_{p2}) = E_{cp1} \cup E_{cp2} \tag{3.22}$$

Aplicando esta definição no exemplo da célula de distribuição, gera algumas modificações; o grafo de alcançabilidade é mostrado na figura 39

primeiro o autômato mostrado na figura 34 é modificado para o que cada posição tenha seu próprio autômato;

segundo é incluida (de forma automática pelo sistema desenvolvido) uma nova transição $E_p = \{RrGm\}$;

terceiro neste exemplo, foram especificados dois estados obrigatórios $X_{ob} = \{\{R,M,SP,PB,PE\},\{R,M,SP,SB,PE\}\}$;

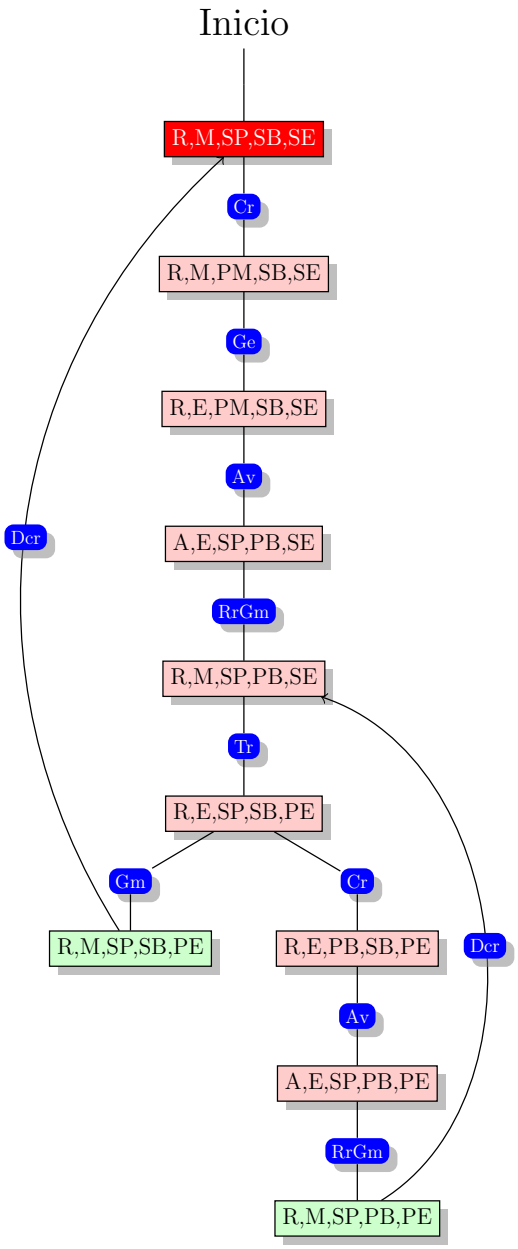


Figura 39: Grafo de alcançabilidade do Automato proposto

É possível observar na figura 39 que os estados obrigatórios adicionados ao sistema levam a uma bifurcação no estado $\{R,M,SP,PE\}$, onde para executar o evento Cr deve existir mais uma peça no magazine. Se não existir, o autômato têm que esperar a peça ser carregada antes de continuar o caminho, caso contrário ele iria retornar para o estado final antes de alcançar todos os estados.

Neste caso, a linguagem do autômato seria como mostra o item 3.23 onde as vírgulas foram colocadas para facilitar o entendimento do leitor.

$$\lambda(A) = \{Cr,Ge,Av,RrGm,Tr,Cr,Av,RrGm,Dcr,Tr,Gm,Dcr\} \tag{3.23}$$

4 REDES DE PETRI

4.1 INTRODUÇÃO

Quando fala-se sobre Redes de Petri, está se falando a respeito de uma ferramenta (gráfica e matemática) para estudos e simulação de sistemas dinâmicos a eventos discretos. No mundo contemporâneo e civilizado, existem diversos exemplos de sistemas a eventos discretos, responsáveis por manter uma organização para a nossa sociedade; Um bom exemplo onde ocorrem os SED são nas indústrias, nos serviços prestados ao público, nos processos burocráticos, nos softwares de tempo real e dos bancos de dados (MORAIS, 2013).

A Rede de Petri se adapta bem a um grande número de aplicações em que as noções de eventos e de evoluções simultâneas são importantes (CARDOSO, 1997). Onde uma análise da RP pode, com sorte, revelar informações importantes sobre a estruturação e comportamento dinâmico do sistema modelado (PETERSON, 1981).

O criador desta teoria foi Carl A. Petri, que em 1962 na Alemanha escreveu uma tese sobre *Comunicação entre Autômatos*. Entretanto, foi um grupo de pesquisadores da *Massachussetts Institute of Technology* - MIT, o responsável por criar entre 1968 e 1976, as bases da teoria que hoje conhecemos como Redes de Petri. Destaque para os pesquisadores F. Commoner e M. Hack (CARDOSO, 1997).

Um exemplo simples de SED é apresentado na Figura 40, onde se tem uma representação de unidade de atendimento de serviço ou de produção.

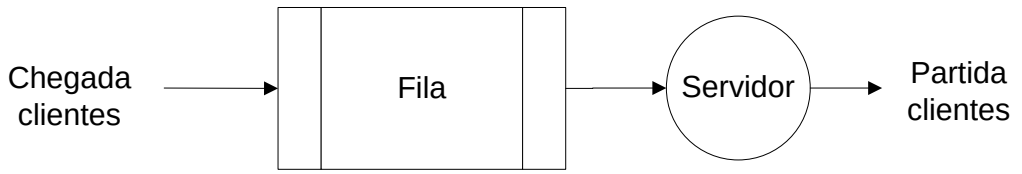


Figura 40: Unidade de atendimento

É importante observar que há sempre três elementos básicos em um SED:

- as entidades (pessoas ou peças) que chegam e esperam - *os clientes*;
- os recursos de atendimento ou produção, limitados, que geram espera - *os servidores*;
- o espaço onde ocorre a espera - *a fila.*

A complexidade dos SED, em particular no caso de sistemas de fabricação automatizada, leva a uma decomposição hierárquica com vários níveis de controle. Geralmente na literatura são utilizados cinco níveis (CARDOSO, 1997):

- planejamento;
- escalonamento;
- coordenação global;
- coordenação de sub-sistemas;
- controle direto (autômatos programáveis diretamente conectados aos sensores e aos atuadores).

Ou seja, a aplicação prática de Petri para a criação e a análise dos sistemas pode ser realizada de várias maneiras. Quaisquer problemas encontrados na análise apontam para falhas no projeto. O mesmo deve ser modificado para corrigir os defeitos. O sistema modificado pode então ser modelado e novamente analisado. Este ciclo é repetido até que a análise não revele mais problemas (PETERSON, 1981). Esta abordagem é apresentada na Figura 41.

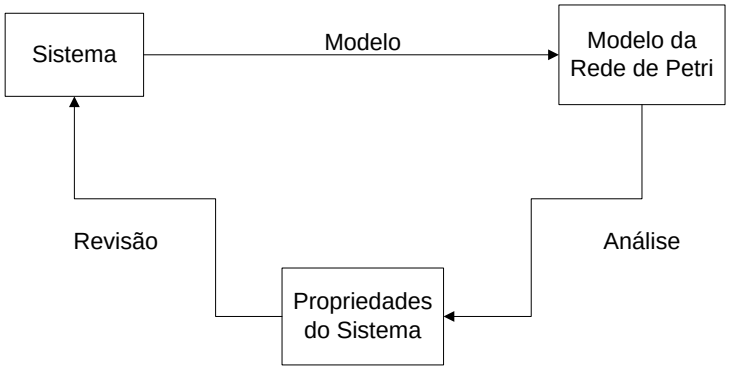


Figura 41: Modelagem de um Sistema utilizando Redes de Petri

4.2 CONCEITOS E FUNDAMENTOS

Algumas qualidades das Redes de Petri ajudam para que esta teoria se destaque na engenharia. Pois é capaz de capturar as relações de precedência e os vínculos estruturais dos sistemas reais, são graficamente expressivas, modelam conflitos e filas, têm um fundamento matemático e prático, e admitem várias especializações como RPs temporizadas, coloridas, estocásticas, etc.

Em (CARDOSO, 1997) as vantagens da Rede de Petri podem ser resumidas pelas seguintes considerações:

1. pode-se descrever uma ordem parcial entre vários eventos, o que possibilita levar-se em conta a flexibilidade;
2. os estados, bem como os eventos, são representados explicitamente;
3. uma única família de ferramentas é utilizada através da especificação, da modelagem, da análise, da avaliação do desempenho e da implementação;
4. uma única família de ferramentas é utilizada nos diversos níveis da estrutura hierárquica do controle, o que facilita a integração destes níveis;
5. uma descrição precisa e formal das sincronizações torna-se possível, o que é essencial para alcançar-se a necessária segurança de funcionamento.

4.2.1 NOÇÕES BÁSICAS

4.2.1.1 CONCEITOS DE MODELAGEM

É importante estar familiarizado com os conceitos básicos utilizados na modelagem de um sistema baseado numa abordagem por eventos discretos. Quando se fala em **Eventos**, estão se referindo aos instantes de observação e de mudança de estado do sistema. Já **Atividades** são as caixas-pretas utilizadas para recuperar e esconder a evolução do sistema físico entre dois eventos (correspondendo em geral ao início e ao fim de uma determinada atividade). E por fim, os **Processos** são nada mais que sequências de eventos e de atividades interdependentes (CARDOSO, 1997).

Com isso pode-se concluir que um evento provoca uma atividade, que provoca um evento de fim de atividade, que pode provocar uma outra atividade e assim sucessivamente.

4.2.1.2 PARALELISMO, COOPERAÇÃO, COMPETIÇÃO

É importante salientar que os processos de um sistema podem ocorrer de forma simultânea, completamente ou relativamente independentes, ou não. Logo, certas atividades podem ser totalmente independentes entre si, e/ou possuírem eventos comuns (CARDOSO, 1997).

Os processos podem ter diferentes formas de interação, por exemplo, podem concorrer a um objetivo comum (**Cooperação**), ou devem ter acesso a um dado recurso para realizar a tarefa determinada (**Competição**), ou até mesmo os eventos podem ocorrer simultaneamente (**Paralelismo**).

Podendo definir as formas de interações como:

- **Cooperação**: Os processos devem ter independência antes de um ponto de sincronização determinado.
- **Competição**: A partir de um ponto de sincronização, existe uma exclusão entre os processos.
- **Pseudo-Paralelismo**: Os processos nunca ocorrem simultaneamente, é um paralelismo aparente, ordenados por exemplo, por um relógio comum.
- **Paralelismo Verdadeiro**: Os processos ocorrem simultaneamente, não sendo possível afirmar de forma precisa qual evento precedeu o outro.

4.2.1.3 ELEMENTOS BÁSICOS DAS REDES DE PETRI

A flexibilidade de definição das Redes de Petri é tamanha que é possível defini-la por meio de grafos, conjuntos e também funções; podendo-se combinar essas definições durante o estudo e análise. Em (MORAIS, 2013) o autor verifica as propriedades das redes por inspeção, por álgebra e por simulação, demostrando a diversidade de "ferramentas" e possibilidades em se trabalhar com RPs.

Se considerar a RP como um grafo orientado (Figura 42), pode-se perceber que a mesma é composta por dois tipos de nós (transições, posições), por arcos com "pesos" indicando partida e chegada no grafo, e *tokens*(fichas, marcas).

Segundo (CARDOSO, 1997) os elementos básicos que permitem a definição de uma RP, são em número de três elementos, os quais são polivalentes, na maioria das vezes podem ser interpretados livremente. Sendo os seguintes:

- **Lugar**(*Place*): São representados por um círculo, podem ser interpretado como uma condição, um estado inicial (duas circunferências concêntricas), um estado parcial, uma espera, um procedimento, um conjunto de recursos, um estoque, uma posição geográfica num sistema de transporte, um estado determinado de um sistema, etc.

- **Transição**(*Transition*): São representadas por barra ou retângulos, são associadas a um evento que ocorre no sistema, como por exemplo o evento/ação *ligar esteira*.

- **Ficha**(*Token*): São representadas por pontos, indicando que uma condição associada ao lugar foi verificada. Pode representar tanto um objeto (recurso) numa certa posição, ou um estado "verdadeiro" indicando a atual situação do sistema.

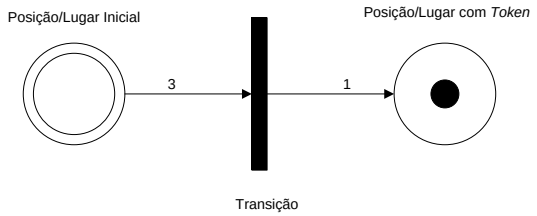


Figura 42: Representação gráfica de uma Rede de Petri

Na literatura, é comum encontrar pequenas diferenças nas definições de um autor para o outro referente as RPs. Em (PETERSON, 1981) o autor define que a Rede de Petri é uma quádrupla formada por conjuntos de *lugares*(E), *transições*(T), *funções de entrada*(I) e *funções de saída*(O) conforme (4.1). Já em (MORAIS, 2013) o autor afirma que a RP é uma quíntupla formada por conjuntos de *lugares*(P), *transições*(T), *arcos*(A), *funções de pesos*(W), e *vetor de marcas*(m) conforme (4.2).

$$RP = (E, T, I, O) \tag{4.1}$$

$$RP = (E, T, A, W, m_0) \tag{4.2}$$

Tais definições variam conforme o nível de detalhamento e complexidade de representação do sistema, cabendo ao programador decidir qual a definição mais indicada para trabalhar com seu problema. Por exemplo, em (TSINARAKIS; VALAVANIS, 2005) o

autor trabalha com uma definição de RP ordinária diferente das duas definições apresentadas, contudo, com elementos que estão presentes em ambas definições. Veja a equação (4.3).

$$RP = (E,T,I,O,m_0) \tag{4.3}$$

4.3 DEFINIÇÕES

Nesta seção, será apresentado a Rede de Petri enquanto um modelo formal e seus conceitos básicos. Conceitos estes, usados em todo o estudo de Redes de Petri e por isso é fundamental para um correto entendimento a respeito de RP.

As definições que serão apresentadas, são semelhantes com as definições da teoria de autômatos. Na verdade, eles definem uma nova classe de máquinas, a *Rede de Petri Autômato*(PETERSON, 1981). Como veremos mais tarde, este ponto de vista pode levar a alguns resultados interessantes na teoria da linguagem formal e teoria dos autômatos.

Conforme a seção 4.2, é possível explicar Redes de Petri através de diferentes visões. Neste presente trabalho, daremos atenção para dois (Grafo Orientado, Sistema de Regras) dos três tipos de representação:

- Grafo Orientado
- Conjunto de Matrizes
- Sistema de regras baseado numa representação do conhecimento (*Condição-Ação*)

Tanto a visão gráfica como a matricial permitem verificar se as transições são paralelas ou se existe um conflito, permite evoluir a rede do sistema; Oferecendo graficamente uma visão mais dinâmica e global dos processos para o projetista, e a representação matricial sendo utilizada pelo computador na verificação automática da rede.

"Já a representação sob um sistema de regras tem o objetivo de compatibilizar a representação da rede de Petri com técnicas de Inteligência Artificial" (CARDOSO, 1997). Um sistema complexo (Figura 43) que utilize uma hierarquia de controle em vários níveis, pode ter a RP como ferramenta na modelagem dos subsistemas por exemplo.

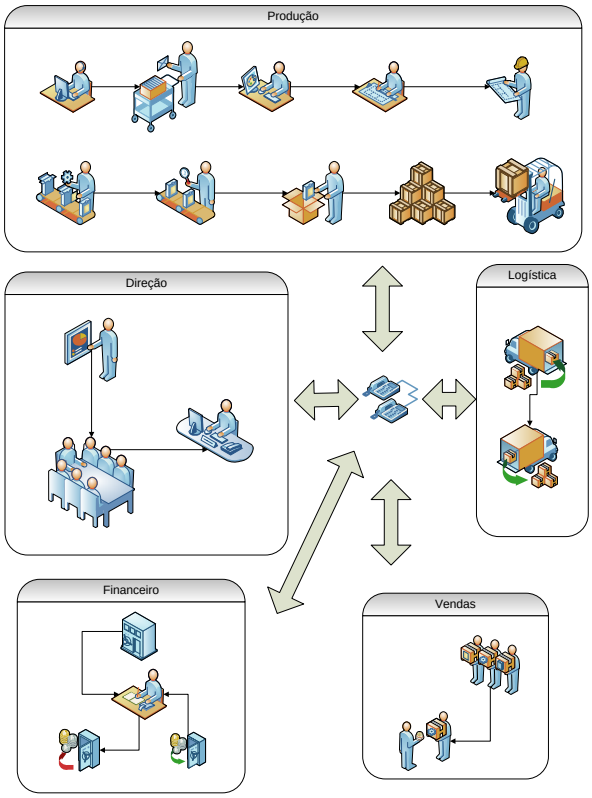


Figura 43: Sistema Complexo dividido em Subsistemas

4.3.1 CONCEITOS

Adotando a RP como uma quádrupla, temos: um conjunto de lugares/estados (E), de transições (T), e de funções de entrada (I) e saídas (O), também conhecidas como *pré-sets* e *pós-sets* respectivamente.

Logo, a estrutura de uma Rede de Petri é definida por estados, transições, funções de entrada e funções de saída conforme 4.1. Onde $E = e_1,e_2,...,e_n$ é o conjunto finito de estados, $n \geq 0$, $T = t_1,t_2,...,t_m$ é o conjunto finito de transições, $m \geq 0$; sendo a interseção desdes dois conjuntos um conjunto vazio.

Os conceitos de Pré-sets (I) e Pós-sets (O) são fundamentais no estudo das RPs. Onde o pré-set de t (Equação 4.4) é o conjunto dos estados em E a partir das quais existe arco para transição t, e o pós-set de t (dado em 4.5) sendo o conjunto das posições em E para as quais existe arco oriundo da transição t.

$$t := \{\forall e_i \in E/(e_i,t) \in I\} \tag{4.4}$$

$$t := \{\forall e_i \in E/(t,e_i) \in O\} \tag{4.5}$$

Analogamente pode-se definir Pré-set e Pós-set também de E, a Figura 44 mostra um exemplo gráfico destas definições.

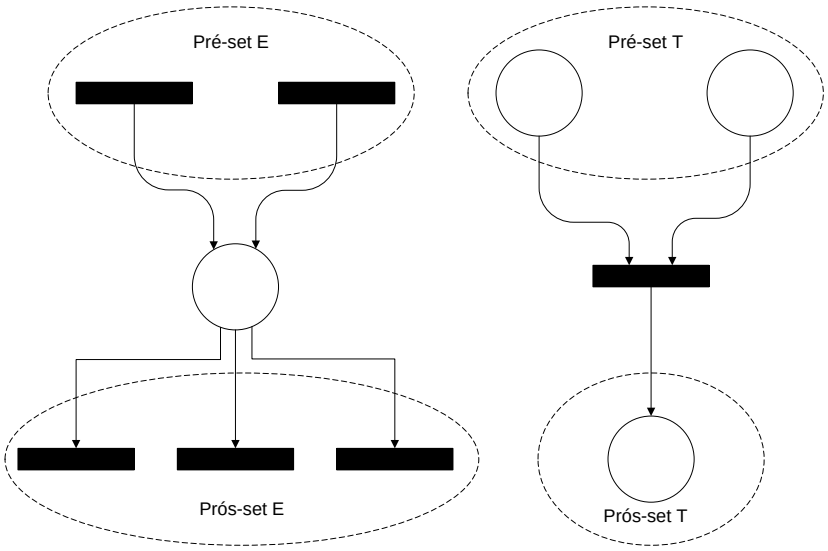


Figura 44: Pré-sets e pós-sets

As funções de entrada e de saída podem ser utilizadas para mapear os arcos que ligam transições a estados, e mapear arcos que ligam estados a transições. É definido uma transição t_j para ser uma entrada de um determinado estado e_i, se e_i é uma saída de t_j (Equação 4.6). Uma transição t_j é uma saída do lugar e_i se e_i é uma entrada de t_j (Equação 4.7).

$$(t_j,I(e_i)) = (e_i,O(t_j)) \tag{4.6}$$

$$(t_j,O(e_i)) = (e_i,I(t_j)) \tag{4.7}$$

Uma estrutura de uma RP representada por uma quádrupla pode ser descrita conforme a Figura 45, contendo os conjuntos de estados (E), transições (T), funções de entrada (I), e funções de saída (O).

Outra forma de analisar uma determinada RP, como dito anteriormente, é através da representação gráfica; sendo o modelo mais usual para ilustrar os conceitos da teoria de Redes de Petri. Por exemplo, um gráfico, G, de Rede de Petri é um multigrafo bipartido dirigido, $G = (V, A)$, onde $V = (v_1, v_2, ..., v_s)$ é um conjunto de vértices e $A = (a_1, a_2, ..., a_r)$ é um conjunto de arcos direcionados, sendo $ai = (v_j, v_k)$ com v_j,$v_k \in V$.

O conjunto V pode ser particionado em dois conjuntos disjuntos E e T tal que

$$C = (E, T, I, O)$$
$$E = \{e_1, e_2, e_3, e_4, e_5\}$$
$$T = \{t_1, t_2, t_3, t_4\}$$

$$I(t_1) = \{e_1\} \qquad O(t_1) = \{e_5\}$$
$$I(t_2) = \{e_2, e_3, e_5\} \qquad O(t_2) = \{e_5\}$$
$$I(t_3) = \{e_3\} \qquad O(t_3) = \{e_4\}$$
$$I(t_4) = \{e_4\} \qquad O(t_4) = \{e_2, e_3\}$$

Figura 45: Estrutura de uma Rede de Petri representada por uma quádrupla

$V = E \cup T$, $E \cap T = \emptyset$, e para cada arco dirigido, $a_i \in A$, se um $a_i = (v_j, v_k)$, tem-se então que $v_j \in E$ e $v_k \in T$ ou $v_j \in T$ e $v_k \in E$.

A figura 46 é um modelo gráfico da Rede de Petri equivalente a estrutura abordada na figura 45. Assumindo a estrutura da RP como uma quádrupla $C = (E,T,I,O)$, define-se o conjunto de vértices V como sendo $V = E \cup T$, e A o conjunto de arcos direcionados para todo $e_i \in E$ e $t_j \in T$.

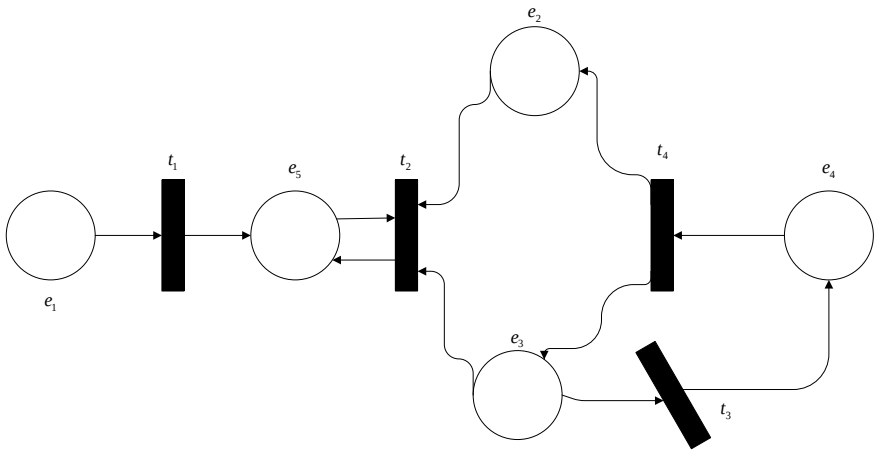


Figura 46: Gráfico de uma Rede de Petri equivalente da estrutura apresentada na Fig. 45

O módulo de distribuição pode ser representado de forma simplificada conforme a Figura 47, sendo o atuador linear representado pelos estados e_1 (recuado) e e_2 (avançado), o atuador giratório por e_3 (pos. magazine) e e_4 (pos. próx. estação), sensor vácuo pelos estados e_5 (desligado) e e_6 (ligado), e as posições das peças pelos estados e_7, e_8, e_9, e e_{10}. Onde e_7 representa a peça no *buffer* do magazine do atuador linear, e_8 indica que a peça se encontra na posição intermediária de processo, e_9 que a peça esta sendo transportada, e e_{10} indica que não há mais peça no processo (foi solta na próx. estação).

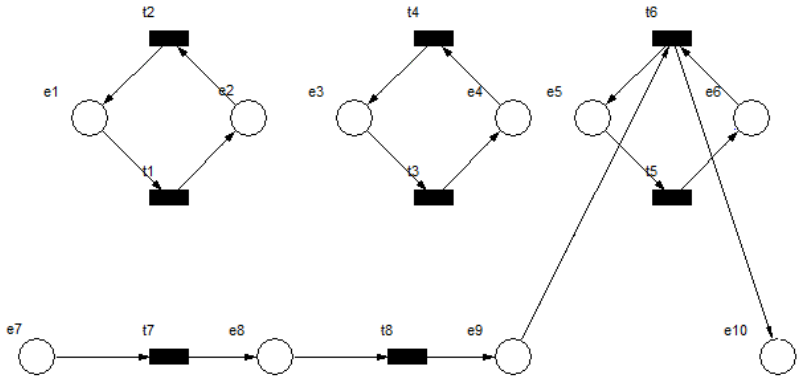


Figura 47: Gráfico de uma Rede de Petri equivalente da estrutura simplificada do módulo *distributing*

4.3.2 REDES DE PETRI MARCADA

Uma *Redes de Petri Marcada* pode ser representada como uma dupla, $M = (C,\mu)$, formada por uma estrutura de RP simples e um conjunto de marcações μ. Podendo ser escrita também na forma de $M = (E,T,I,O,\mu)$.

A marcação μ indica a quantidade de *tokens*(marcas,fichas, *jetons* em francês) contida no estado e de uma RP marcada. O número e a posição dos *tokens* podem mudar durante a execução da Rede de Petri.

O conjunto μ é definido como um vetor de n posições, $\mu = (\mu_1,\mu_2,...,\mu_n)$, onde n é igual a número de estados presentes no sistema. Sendo cada peso, μ_i, relacionado a um determinado estado, e_i, conforme a função $\mu(e_i) = \mu_i$.

A notação em forma de função é geralmente o mais comum de ser visto, já em gráficos de Redes de Petri é representado as marcações por meio de pequenas marcas circulares dentro dos círculos que representam os estados na RP. A Figura 48 é um exemplo de uma representação gráfica de RP Marcada.

No módulo *distributing* a configuração inicial com uma peça no *buffer* (e_7) é composta por: atuador linear recuado (e_1), atuador giratório na posição magazine (e_3), vácuo desligado (e_5) conforme a Figura 49.

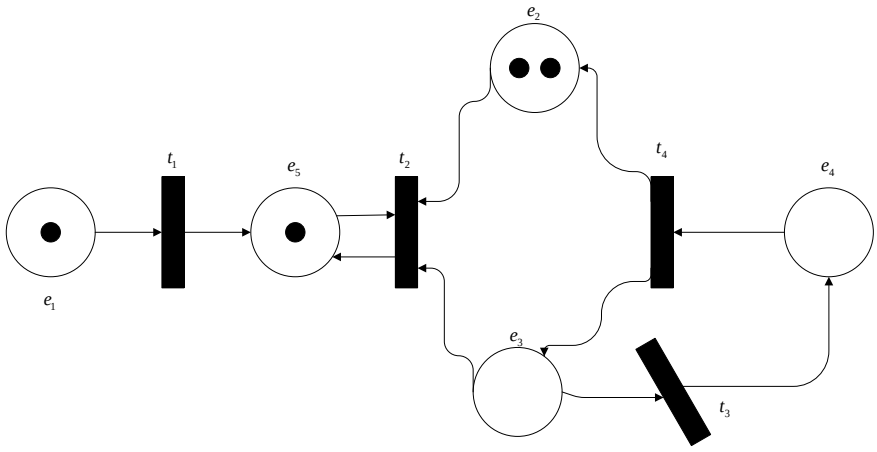


Figura 48: Gráfico de uma Rede de Petri Marcada equivalente da estrutura apresentada na Fig. 45

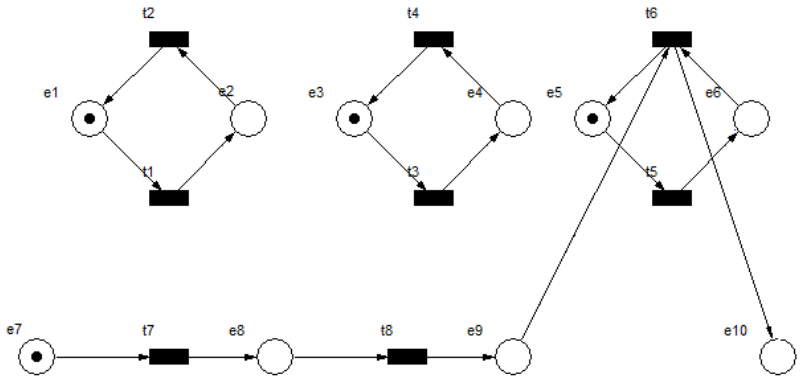


Figura 49: Gráfico de uma Rede de Petri equivalente da estrutura simplificada do módulo *distributing* com fichas

4.3.3 REGRAS DE EXECUÇÃO PARA REDES DE PETRI

A variação de valor dos *tokens* pela rede de acordo com certas regras é chamada de *execução da Rede de Petri*. Esta, é controlada pela quantidade e distribuição dos *tokens* em toda a RP.

Todo este processo é regido por duas fases, habilitação e disparo de transição, responsáveis por remover *tokens* de estados que estão enviando um "sinal" de entrada para a transição ativa, adicionar novos *tokens* conforme o "sinal" de saída indicado na distribuição da rede.

Tem-se uma transição t_j habilitada, se e somente se $t_j \in T$, para todo $e_i \in I$, com a condição de que a Eq. 4.8 seja verdadeira.

$$\mu(e_i) \geq (e_i, I(t_j)) \tag{4.8}$$

Isto é, ela é ativada se o número de *tokens* em cada um dos lugares de entrada for maior (ou igual) que o peso do arco que liga este lugar à transição (CARDOSO, 1997).

Ao habilitar a transição, ela é disparada por meio de duas operações:

- retirada dos *tokens* das posições do pré-set (número referente ao valor do peso do arco correspondente),
- adição em cada um dos estados do pós-set (número referente ao valor do peso do arco correspondente).

Pode-se então definir a nova marcação, μ', pela Eq. 4.9:

$$\mu'(e_i) = \mu(e_i) - (e_i, I(t_j)) + (e_i, O(t_j)) \tag{4.9}$$

Para uma melhor compreensão, a Figura 50 ilustra como a marcação se desenvolve de acordo com a habilitação e disparo. Cada estado pode ou não receber um *token* ou enviar, de acordo com o estado das transições.

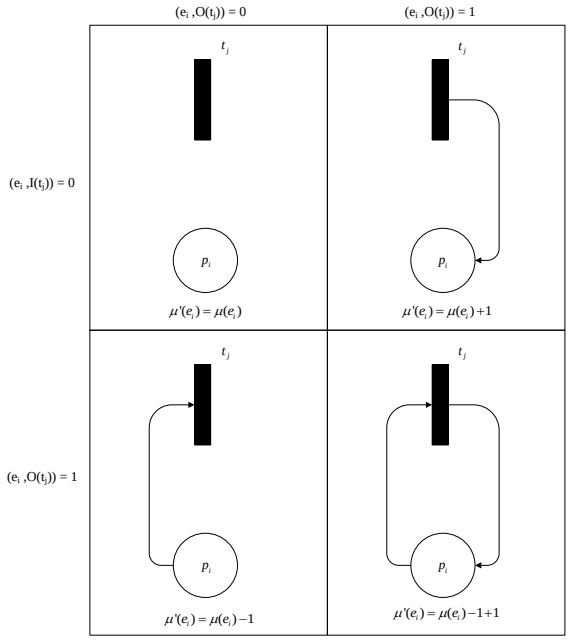


Figura 50: Ilustração de como a marcação de um lugar muda quando uma transição dispara

(PETERSON, 1981)

Logo, as regras de execução para o módulo de distribuição podem ser representadas conforme a Figura 51. A proposta da modelagem é levar a peça da posição *buffer* do magazine (e_7) para outra estação (e_{10}) sem colocar em risco os atuadores do módulo e garantir que o processo seja realizado de forma ótima, ou seja, sem que ocorra eventos desnecessários para a obtenção do objetivo final.

Na representação do grafo, as linhas pontilhadas são chamadas de *arcos de teste* pois tem a função de apenas indicar para a transição que determinado estado contém ficha. Já os arcos contendo um círculo na ponta indicam para a transição que só será ativada se não houver fichas naquele estado.

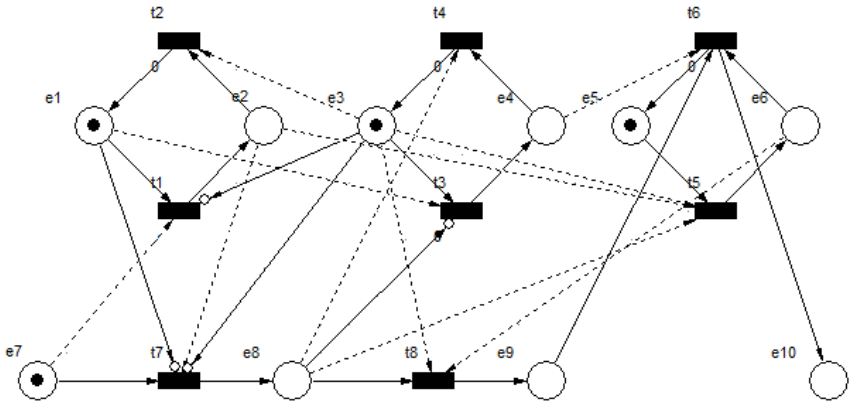


Figura 51: Gráfico da RP equivalente do módulo *distributing* com regras de execução

É importante observar que a transição t_1 (responsável por avançar o atuador linear) só é ativada caso o atuador linear esteja recuado, o at. giratório não pode estar na pos. de magazine e deve haver peça no *buffer*, garantindo assim a integridade dos atuadores. Outra transição com regra de execução que é necessária para a obtenção do objetivo final do processo é t_5 (ligar vácuo), a qual só é ativada se o at. linear estiver avançado, atuador giratório na posição do magazine e a peça na posição intermediária.

As outras transições como por exemplo recuar at. linear (t_2), mover at. giratório para próx. estação (t_3), mover at. gir. para magazine (t_4), e desligar vácuo (t_6) também possuem regras que garantem uma sequência segura e desejada para a obtenção do objetivo final do processo. Sendo t_7 e t_8 transições que representam a mudança de posição da peça após os eventos ocorridos.

4.3.4 CONFLITO E PARALELISMO

Na subseção 4.2.1.2 apresentou-se de forma resumida os conceitos a respeito de conflitos e paralelismo em um sistema. Nesta subseção será explanado tais situações na aplicação das Redes de Petri, afim de tornar mais simples a tarefa de modelagem de um sistema. Em (CARDOSO, 1997), o autor comenta a importância de identificar a interação entre as atividades de um processo: "*Uma vez feita a identificação da interação entre as atividades, basta fazer a tradução para o modelo*". Entende-se por conflito, um dado estado do sistema que possibilite duas ou mais transições, excludentes, ocorrerem e atrapalharem a evolução do mesmo. Obrigando que a execução de tais tarefas seja feita uma por uma.

A ordem de execução em que se escolhe executar as transições altera o resultado final, gerando grandes problemas em uma linha de produção automotiva por exemplo. No sistema da direita na Figura 52, duas transições estão aptas para serem ativadas, porém, ao aplicar a operação de remoção dos *tokens* para uma das transições, imediatamente desabilita-se a outra transição, ocasionando um tipo de conflito chamado **conflito-confusão**.

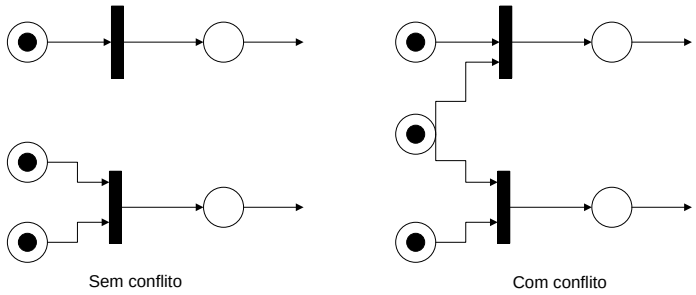


Figura 52: Redes de Petri Sem conflito(esq.) e Com conflito(dir.)
(MORAIS, 2013)

A estrutura de um determinado modelo, pode conter algum tipo de conflito e/ou paralelismo que não possa ser efetivado; a ocorrência dos mesmo depende necessariamente que a marcação possibilite tal fato. Tem-se então duas possibilidades de ocorrência para conflito e paralelismo dentro de uma RP. Eles podem ser estruturais (fazem parte do modelo do sistema mas não interferem no funcionamento da rede), ou efetivos (as marcações no sistema permitem sua ocorrência, podendo ser prejudicial ou não para o funcionamento da RP).

Em (CARDOSO, 1997), o autor define estas 4 possibilidades da seguinte maneira:

Conflito estrutural: *"duas transições t_1 e t_2 estão em conflito estrutural (Eq.4.10)*

se e somente se elas têm ao menos um lugar de entrada em comum:"

$$\exists e \in E, I(e,t_1)I(e,t_2) \neq 0 \tag{4.10}$$

Conflito efetivo:*"duas transições t_1 e t_2 estão em conflito efetivo (Eq.4.11) para uma marcação M se e somante se ambas estão em conflito estrutural e estão sensibilizadas:"*

$$M \geq I(.,t_1) e M \geq I(.,t_2) \tag{4.11}$$

Paralelismo estrutural:*"duas transições t_1 e t_2 são paralelas estruturalmente (Eq.4.12) se não possuem nenhum lugar de entrada em comum:"*

$$\forall e \in E, I(e,t_1)I(e,t_2) = 0 \tag{4.12}$$

Paralelismo efetivo:*"duas transições t_1 e t_2 são paralelas (Eq.4.13) para uma marcação dado M se e somente se são paralelas estruturalmente e:"*

$$M \geq I(.,t_1) e M \geq I(.,t_2) \tag{4.13}$$

Logo, caso a análise de um modelo via Redes de Petri indique que exista um conflito, tal modelo deve ser aperfeiçoado: revendo a modelagem do sistema físico, ou modificando o sistema real e seu modelo (adição de controladores, mais recursos nos estoques, etc.)

4.4 SISTEMA DE REGRAS

Um sistema de produção (sistema de regras) é um algoritmo, uma forma de inteligência artificial, que basicamente relaciona um conjunto de regras sobre determinados comportamentos ou ações a serem tomadas. Uma ferramente útil para o planejamento automatizado, sistemas especialistas e seleção de ação. Tendo como objetivo, fornecer mecanismos necessário para executar produções, a fim de atingir alguma meta para o sistema.

A Rede de Petri pode ser considerada como um sistema de regras, dada a sua característica de fazer evoluir o estado a partir de regras representadas por transições,

baseado em uma representação da forma:

$$se <condição> \ então <ação>$$

Composto por três elementos; um conjunto que represente o conhecimento disponível sobre o sistema (base de fatos), um conjunto capaz de deduzir novos fatos (base de regras), e um mecanismo de inferência capaz de realizar novas deduções, aplicando as regras aos fatos (CARDOSO, 1997).

O mecanismo de inferência consiste em uma "ferramenta" capaz de organizar o conjunto de regras em forma de lista, e percorrê-la sequencialmente. Onde a regra é aplicada ou disparada se a parte *condição* da regra for verdadeira, caso contrário (nenhuma regra aplicável) o mecanismo de inferência para.

Assim como na RP, o Sistema de Regras pode ter um cenário onde várias regras podem ser aplicadas, interferindo no resultado final da dedução conforme a regra escolhida. Logo, em um dado contexto onde diversas regras podem ser aplicáveis, existe a possibilidade de ocorrer um conflito.

Fazendo uma analogia entre as duas técnicas, a base de regras de um Sistema de Regras corresponde ao conjunto de transições da rede, com suas condições e ações; à base de fatos inicial correspondendo a marcação inicial da RP.

O Sistema de Regras utilizado em conjunto com a Rede de Petri, permite não só representar sistemas baseados em conhecimento e obter resultados quando não existe um algoritmo conhecido, mas também permite uma análise mais detalhada sobre seu modelo.

4.5 ANÁLISE DAS REDES DE PETRI

O objetivo principal deste trabalho é aplicar as Redes de Petri em automação industrial e sistemas de manufatura. Logo, apesar de existir na literatura muitas classes de Redes de Petri, iremos apresentar apenas duas classes (Grafos de Eventos e Máquinas de Estado).

4.5.1 CLASSES

4.5.1.1 DEFINIÇÕES

Para um melhor entendimento a respeito das classes das Redes de Petri, é importante ter conhecimento de algumas definições básicas necessárias:

Rede de Petri Pura: Não possui posição comum ao pré-set e ao pós-set em suas transições (Eq. 4.14); sendo possível transformar qualquer Rede de Petri em uma rede pura adicionando-se posições e transições artificiais.

$$\forall t | \{^\bullet t\} \cap \{t^\bullet\} = \emptyset \tag{4.14}$$

Rede de Petri Ordinária: Todos os arcos da rede têm peso igual a 1 (Eq. 4.15); são redes em que se predomina ações puramente lógicas(0,1), onde não envolvem quantidades variáveis.

$$\forall m \in M | m = 1 \tag{4.15}$$

Circuito direcionado: Caminho de arcos sucessivos, de um nó até ele mesmo.

Rede de Petri Fortemente Conexa: Conjunto de circuitos direcionados que formam uma única rede (Figura 53).

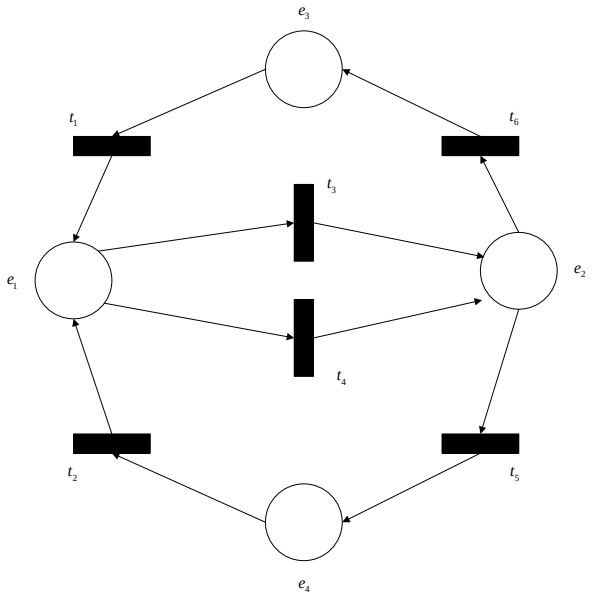


Figura 53: Rede de Petri fortemente conexa, com circuito elementar, representando um sistema com conflito confusão.

Circuito Elementar: Classificação dada a um circuito direcionado cujo o único nó repetido é o inicial. Na Figura 53, existem quatro circuitos elementares:

$$(t_1, t_3, t_6),(t_1, t_4, t_6),(t_2, t_3, t_5),(t_2, t_4, t_5)$$

4.5.1.2 GRAFO DE EVENTOS

Também chamado de Grafo Marcado, é uma Rede de Petri ordinária, onde cada estado tem exatamente uma transição de entrada e uma transição de saída (Eq. 4.16). Na Figura 54 tem-se o exemplo de duas redes representando um modelo de Grafo de Eventos e outro não.

$$\forall e \in E, |^\bullet e| = |e^\bullet| = 1 \tag{4.16}$$

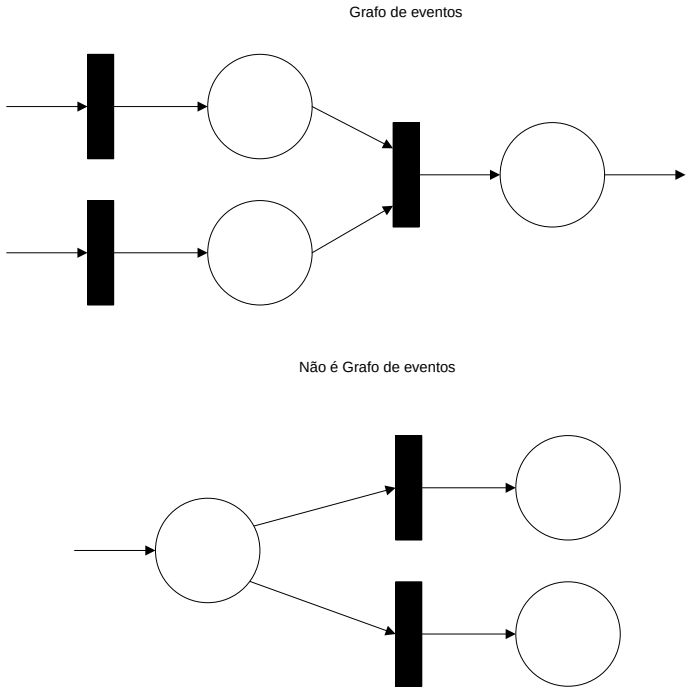


Figura 54: Duas redes representando um exemplo de Grafo de Evento e outro não.

4.5.1.3 MÁQUINA DE ESTADO

Outro tipo de Rede de Petri ordinária em que cada transição tem exatamente uma posição de entrada e uma posição de saída (Eq. 4.17). São redes capazes de modelar sistemas com conflito do tipo confusão, e um exemplo de Máquina de Estado pode ser observado na Figura 53.

$$\forall t \in T, |^{\bullet}t| = |t^{\bullet}| = 1 \tag{4.17}$$

Conclui-se então que, Grafos de Eventos e Máquinas de Estado são Redes de Petri ordinárias duais uma da outra.

4.5.2 PROPRIEDADES

Neste item serão abordados as propriedades referentes ao modelo e ao desempenho das Redes de Petri. É de fundamental importância que um sistema automático atenda às necessidades dos usuários.

Um exemplo relevante de um modelo de RP, é a Rede marcada reiniciável; que tem como característica encontrar uma sequência de disparo, a partir de qualquer marcação acessível, que leve a rede de volta à marcação inicial. Entretanto, quando não é possível voltar à marcação inicial, porém existe uma sequencia de disparo que leve a uma marcação que sensibilize cada uma das transições da rede, classifica-se esta rede como uma rede viva.

4.5.2.1 PROPRIEDADES ESTRUTURAIS

- Limitação (Rede k-limitada)

Em uma simulação é possível modelar um sistema que venha a ter a característica de ter em algum de seus estados um número de marcações (*tokens*) que tendem ao infinito(Fig. 55), afim de avaliar o desempenho de um sistema independentemente dos limites de seus elementos de armazenamento. Entretanto, é um modelo inviável para representar um sistema real, visto que não é possível garantir um estoque de matéria prima infinito para a produção de determinado produto por exemplo.

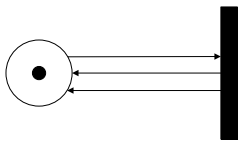


Figura 55: RP não limitada

Um estado/posição é considerado k-limitado se e somente se para um determinado estado(e), o número de *tokens*(M') for menor ou igual a k, para todos os *tokens* subsequentes ao estado inicial (CARDOSO, 1997). Ou seja, deve-se obedecer a Eq. 4.18:

$$\{\forall M' \in A(R,M) | M'(e) \leq k\} \tag{4.18}$$

Defini-se então como uma rede k-limitada(*bounded*), uma rede em que todos os seus estados são k-limitados. Podendo ser classificada como binária ou segura, se for k-limitada com $k = 1$.

Em uma rede binária, os lugares representam condições lógicas, ajudando a encontrar uma incoerência em determinado modelo caso mais de um *token* esteja presente em um de seus estados.

• Conservação

Uma rede conservativa é um tipo de RP em que a soma total dos *tokens* permanece constante durante toda a sua execução (MORAIS, 2013).

• Vivacidade (Rede marcada viva)

Considera-se uma Rede de Petri viva, se dado um estado inicial, todas as suas transições são vivas (habilitadas a partir de algum estado consequente do estado inicial).

A característica da RP viva é garantir que nenhum bloqueio pode ser provocado pela estrutura da rede, vale ressaltar no entanto, que ela não é capaz de garantir a ausência de eventuais bloqueios provocados por uma má interação entre a rede e o seu ambiante externo (CARDOSO, 1997).

• Conflito mortal

"O principal problema operacional em sistemas a eventos discretos é o conflito mortal (*deadlock*)" (MORAIS, 2013). Este tipo de conflito ocorre quando há uma parada total das habilitações subsequentes a um determinado estado, este conflito impossibilita a vivacidade de uma forma particular no modelo.

Em alguns casos, para eliminar este tipo de conflito, é necessário modificar não só a RP mas também o sistema real, adicionando mais transições ou modificando alguns estados. A Figura 56 representa um tipo de conflito mortal, onde o sistema para pois não é possível receber nenhum *token* no final do modelo, visto que ambas as linhas dependem de marcas uma das outras para ativar as transições.

Podendo-se definir que para um determinado estado esteja em *deadlock*, todas as suas transições pertencem tanto ao pré-set, como também pertencem ao pós-set de um determinado e_d (estado com conflito mortal).

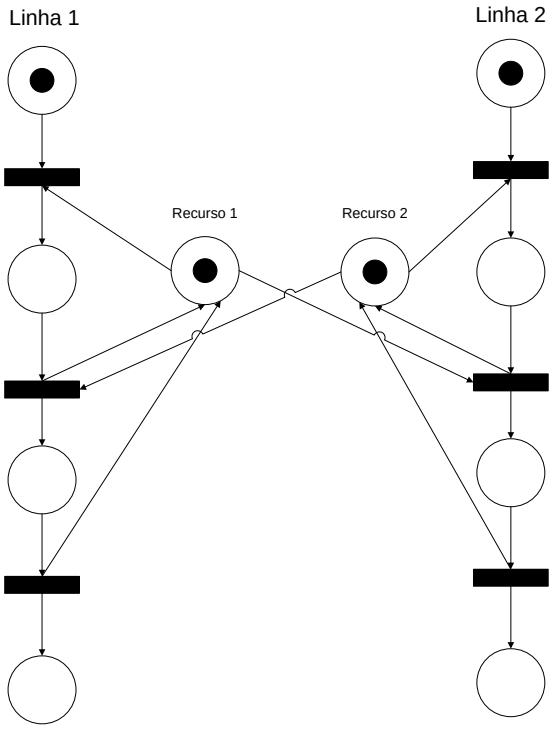


Figura 56: Rede de Petri com ocorrência de *deadlock*

• Persistência

Quando uma rede tem transições capazes de serem habilitadas simultaneamente, e a execução de uma não desabilita a outra, a RP é considerada como persistente. Logo, uma rede que apresente um conflito-confusão não pode ser considerada persistente.

• Reversibilidade

Quando uma rede é classificada como reversível, entende-se que seu sistema seja capaz de retornar a um *estado específico*, caso ocorra uma interrupção pelo operador ou por algum dispositivo de segurança. É uma propriedade importante, pois visa a recuperação dos sistemas diante de eventos perturbadores do funcionamento.

A RP aplicada neste trabalho é uma Rede de Petri ordinária formalmente definida na equação 4.19. Esta representação inclui elementos presentes nas definições do Capítulo 4:

$$RP = \{P,T,I,O,m_0\} \tag{4.19}$$

onde

• $P = \{p_1,p_2,...,p_n\}$ - Conjunto finito de estados (*Places*)

- $T = \{t_1,t_2,...,t_n\}$ - Conjunto finito de transições
- $I : (T) \rightarrow P$ - Funções de entrada
- $O : (T) \rightarrow P$ - Funções de saída
- m_0 - Distribuição inicial dos *tokens* nos estados.

O conceito de pré-set e pós-set é crucial no estudo e aplicação da metodologia proposta, sendo importante para o entendimento do funcionamento do programa (com suas entradas e saídas de dados).

5 MODELAGEM DO MÉTODO PROPOSTO

5.1 INTRODUÇÃO

Esse capítulo tem o objetivo de apresentar a metodologia de forma mais clara e detalhada para o entendimento da ferramenta proposta. Inicialmente, serão apresentadas as representações de sistemas modelados na linguagem de Autômatos que serão utilizados na entrada de dados. Em sequência, serão descritos os sistemas modelados na linguagem de Rede de Petri. Finalmente, na segunda metade deste capítulo serão apresentados a modelagem trabalhada em linguagem C# e Prolog, de modo a permitir o completo entendimento do funcionamento e resultado da metodologia proposta.

A metodologia desenvolvida para a construção dos modelos representativos de sistemas complexos em linguagem autômata tem como objetivo aplicar as regras de René Descartes (DESCARTES; WEISSMAN; BLUHM, 1996) na modelagem de sistemas.Devido a complexidade de alguns cenários (sistema), é necessário dividi-lo no maior número de partes possível, de modo que cada atuador/sensor será modelado como um pequeno autômato; Onde a linguagem autômata tem a finalidade de modelar de forma mais clara e intuitiva o sistema.

Uma releitura das regras foi realizada e adaptada no presente trabalho:

- A linguagem autômata entra com a finalidade de modelar intuitivamente, com ideias claras, o sistema que se deseja controlar (Regra da evidência). De acordo com a complexidade do mesmo, se torna necessário dividi-lo no maior número possível de partes, sendo cada atuador/sensor um autômato independente (Regra da análise).

- Tal modelagem deve respeitar a ordenação das partes, quanto a um sistema único, seguindo o critério de relação constante entre elas. É de extrema importância garantir a sequência de ações que devem ocorrer no desenvolvimento do processo a ser controlado, garantindo a integridade dos componentes do sistema (Regra

da síntese).

- Respeitadas as relações dos componentes do sistema, a plataforma irá compor os autômatos simples em um autômato global. O próximo passo da análise requer que o sistema seja simplificado, selecionando exclusivamente o que for necessário e suficiente para o funcionamento do sistema.

- Finalmente, este autômato global simplificado é transformado em uma RP, e trabalhado como sistema de regras por um mecanismo de inferência. Este mecanismo é encarregado de encontrar uma solução ótima no universo de possibilidades, que será transformada em programação do CLP. O diagrama apresentado pela Figura 57 representa o diagrama de funcionamento proposto pela plataforma.

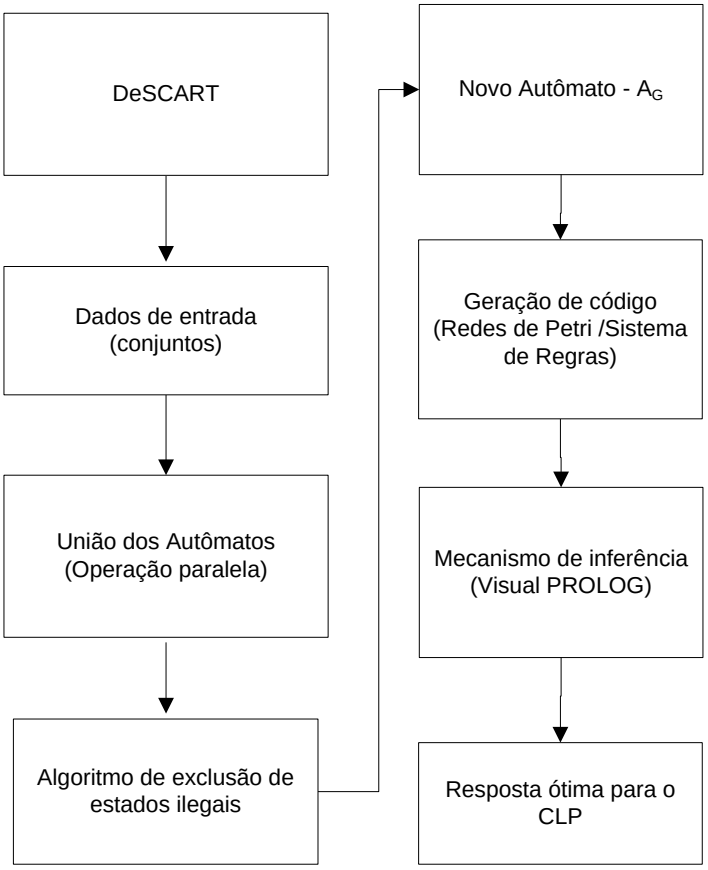


Figura 57: Diagrama de estratégia de ação

Uma abordagem modular para controle de supervisão proposta neste trabalho é aplicada aos módulos didáticos da FESTO. Em geral, as condições para a existência de uma solução modular são mais fortes do que para uma solução não-modular (WONHAM; RAMADGE, 1988). No entanto, quando aplicável, uma síntese modular oferece

as vantagens importantes de redução da complexidade computacional e design modular estruturado.

No restante desta seção serão detalhados todos os passos descritos pelo algoritmo do diagrama mostrado na Figura 57

5.2 SUPERVISOR

Neste trabalho tem-se como objetivo controlar uma planta que simula um processo industrial, considerando-a como um sistema a eventos discretos e trabalhando seus estados e eventos em linguagem automata. Não é obrigatório modelar o sistema como um autômato, entretanto, o método proposto trabalha com uma representação mais flexível e intuitiva do sistema a modelar. Para obter o desempenho desejado nas modelagens é necessário determinar algumas especificações, as quais visam trabalhar em uma condição ótima, utilizando o mínimo de restrição possível. O problema de controle é considerado totalmente resolvido quando existe um controlador capaz de garantir que as especificações são atendidas.

O conjunto de possibilidades de eventos em um determinado sistema pode ser infinito, muitas vezes algumas sequências de eventos são consideradas indesejadas (por colocar a integridade do elementos do sistema em risco, entrar em um ciclo vicioso, ou até mesmo ir para um estado que trave todo o desenvolvimento da planta). Em (RAMADGE; WONHAM, 1989) o autor defende que para controlar um SED é necessário postular que certos eventos do sistema podem ser desativados (impedidos de ocorrer). Isso permite influenciar a evolução do sistema, proibindo a ocorrência de eventos importantes em determinados momentos, criando-se assim o conceito de supervisor.

Uma abordagem modular para o controle de uma planta didática na forma de um sistemas de controle de eventos discretos é apresentada, e ilustrada com exemplos no decorrer deste trabalho. Sistemas a eventos discretos são modelados por autômatos, juntamente com um mecanismo para ativar e desativar um subconjunto de transições de estado. O problema básico é garantir uma supervisão adequada que o comportamento de malha fechada do sistema encontra-se dentro de um determinado comportamento legal (WONHAM; RAMADGE, 1988). Assumindo que este comportamento pode ser decomposto em um cruzamento de restrições de componentes, que determinam as condições em que é possível sintetizar o controle apropriado de forma modular.

A maioria dos problemas que envolvem o SED é composto por vários subsistemas

concorrentes. Neste contexto, o principal objetivo do controle de supervisão é a coordenação de tais subsistemas de modo que o sistema global obedeça a uma série de especificações individuais e conjuntas (QUEIROZ; CURY, 2002). Considere um sistema modelado composto por L e L_m, onde L é o conjunto de todas as possibilidades de combinações que o SED pode gerar e $L_m \subseteq L$ é a linguagem dos *strings* marcados que é usado para representar a conclusão de algumas operações ou tarefa, onde a definição de Lm é uma questão de modelagem.

L e L_m são definidos ao longo do conjunto de eventos E. L é sempre o mesmo, ou seja, $L = L$, enquanto L_m não precisa ser necessariamente. Sem perda de generalidade e por uma questão de conveniência, suponha que L e L_m são as linguagens geradas e marcadas pelo autômato A:

$$A = (X,E,f,\Gamma,x_0,X_m)$$

Essas linguagens, $\mathcal{L}_m(A) = L_m$ e $\mathcal{L}(A) = L$, são provenientes do conjunto de funções ativas, $\Gamma(X_i,E_i)$, sendo E dividido em dois subconjuntos distintos ($E = E_c \cup E_{inc}$), onde E_c é o conjunto de eventos controláveis os quais podem ser impedidos de ocorrer ou desabilitados por um dado supervisor e E_{inc} o conjunto de eventos incontroláveis.

Os módulos da FESTO, assim como a maioria dos problemas que envolvem sistemas a eventos discretos, é composto por vários subsistemas concorrentes. Neste contexto, o principal objetivo do controle de supervisão é a coordenação de tais subsistemas de modo que o sistema global obedeça a uma série de especificações individuais e conjuntas. Portanto, se faz necessário modelar estes subsistemas em paralelo (identificando suas singularidades e limitações), ou seja, cada elemento ou conjunto de elementos que podem ser considerado "uma parte do todo".

Ao analisar o módulo *Distributing* apresentado neste trabalho, pode-se considerar como subsistemas os atuadores linear e giratório. Apesar de fazerem parte do mesmo módulo, são elementos que trabalham de forma independente em um mesmo sistema. Identificar a sequência desejada, ou a linguagem que não pode estar presente no universo de possibilidades do sistema se torna necessário para garantir a eficiência e segurança do processo como um todo.

Logo, o controle de SED proposto na literatura defende a modelagem de um supervisor consiste em alternar a entrada de controle através de uma sequência de elementos (RAMADGE; WONHAM, 1989). Entretanto, construir um supervisor não é trivial, é

preciso conhecer a fundo o sistema que se deseja controlar. Os atuadores linear e giratório por exemplo, não podem variar seu estado de uma forma totalmente aleatória, é preciso uma determinada sequência para que a peça que se deseja transportar para o módulo seguinte alcance seu objetivo e não ponha em risco a integridade dos atuadores (o atuador linear não pode avançar caso o giratório esteja não posição magazine).

Em pequenos sistemas o supervisório é de fácil obtenção, todavia, o aumento de complexidade de modelagem é significativo e torna-se necessário que o programador seja um perito no processo que se deseja controlar, entendendo a fundo as limitações dos atuadores e o processo em si; determinando os estados obrigatórios ao modelar de forma precisa e funcional o supervisor.

O controle supervisório para eventos discretos proposto por Shengbing e Kumar em (JIANG; KUMAR, 2006) defende o controle de uma planta através da composição síncrona obtida com a interseção de uma dada planta P e um supervisor S, denotado por $P||S$, o qual é definido como $P||S = (Z, \sum, \delta_{P||S}, z_0, AP, L_{P||s})$, em que $Z = X_P \times X_S$ é o conjunto dos estados e $\delta_{P||S} : Z \times \Sigma \rightarrow 2^Z$ é a função de transição de estados para $P||S$. Um modelagem baseada neste proposta para o módulo *Distributing* utilizando a linguagem dos autômatos poderia ser feita conforme a Figura 58:

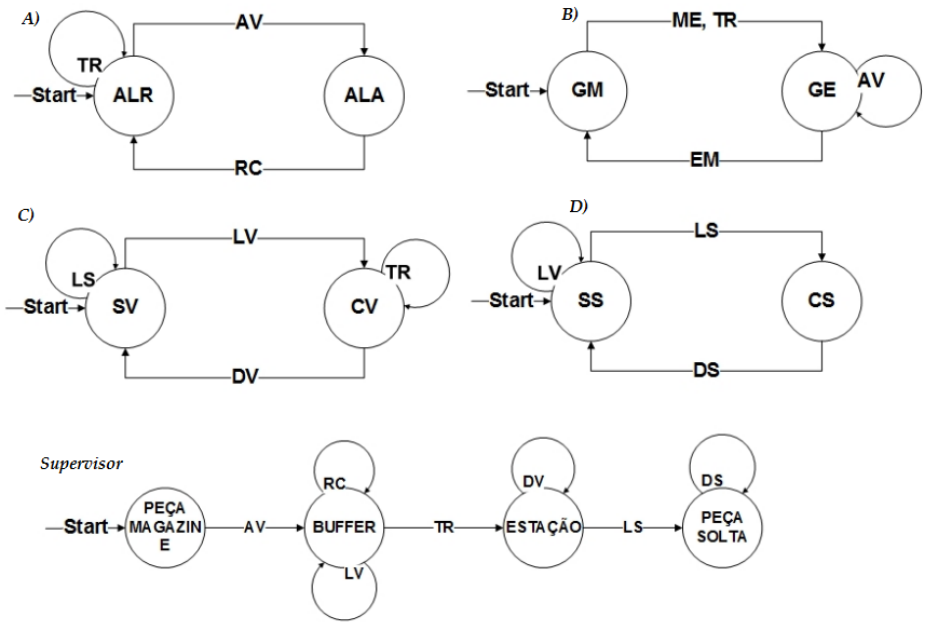


Figura 58: Autômatos (A,B,C,D) e supervisor S do módulo *Distributing*

Como passo final antes da implementação física, a minimização dos supervisores é levado em conta, para a redução do número de estados de um supervisor pode representar economia de memória e esclarecer a lógica de controle (QUEIROZ; CURY, 2002). Em

(KOSMATOPOULOS; CHRISTODOULOU, 1995) o autor defende que uma vez modelado o sistema de produção, é de interesse primário a construção de uma política de controle que otimiza o desempenho do sistema de manufatura. Normalmente, o objetivo da política de controle é o de produzir em um tempo mínimo um certo número de objetos.

Este trabalho visa garantir o controle do sistema que se deseja modelar sem a utilização do supervisor "clássico", o qual é de difícil obtenção para sistemas extremamente completos e para programadores que não detém o conhecimento de um especialista no sistema que se deseja programar. O aspecto mais difícil de síntese modular é assegurar que o supervisor modular resultante seja do tipo sem bloqueio (WONHAM; RAMADGE, 1988).

Os estados obrigatórios são determinados a medida que os arcos de *self-loop* são criados em cada autômato separado para cada atuador, determinando que aquele evento é permitido de ocorrer no estado marcado sem o alterar. Em contra partida, os eventos que colocariam o sistema em uma situação critica ou de bloqueio são ignorados nos *self-loops*, indicando na metodologia que aquele evento não pode ocorrer no estado marcado atual.

O método proposto garante com isso uma programação mais intuitiva do usuário, possibilitando definir de forma clara a sequência não desejada do sistema (as linguagens que não podem estar presente) ao fazer a composição dos autômatos que trabalham em paralelo. Tal composição ainda permite que os "novos estado" gerados que não possuem eventos que determinem se o mesmo é um pré-set ou pós-set sejam descartados por serem considerados inalcançáveis ou bloqueios mortais. Gerando uma nova árvore de alcançabilidade, a qual garante integridade física do elementos do sistema e a obtenção do objetivo.

Para alcançar um desempenho ótimo do sistema que se deseja modelar, foi utilizado uma ferramenta baseada na linguagem PROLOG, contendo um sistema de regras, para indicar ao programador a sequência ótima necessária para realizar a tarefa dos módulos que se desejada modelar. Onde o método de busca implementado, A^*, foi utilizado para resolver problemas de fluxos simples.

5.3 REPRESENTAÇÃO METODOLÓGICA E GRÁFICA DOS AUTÔMATOS

A metodologia de representação gráfica desenvolvida tem como tarefa a decomposição de sistemas em componentes simples na linguagem autômata, foi denominada como

DeSCART (*Decomposition in Simple Components by Automata Restriction Task*). Responsável por descentralizar a informação ao passo que o sistema modelado é decomposto em pequenos subsistemas, adotando que cada componente do sistema (sensores e atuadores) sejam representados como pequenos autômatos.

Além da técnica de decomposição, DeSCART é uma abordagem simples de representação, definindo todas as possíveis funções de transição para cada estado (funções de ativação) de cada autômato no universo de eventos possíveis (conjuntos de eventos).

Seja um autômato simples $A_i = \{E_i,X_i,f_i,x_0^i,x_{obj}^i\}$ a descrição de um componente de um sistema e $A_j = \{E_j,X_j,f_j,x_0^j,x_{obj}^j\}$ a descrição de outro componente. O conjunto dos eventos que podem ocorrer em ambos autômatos, $E_{ij} = E_i \bigcap E_j$, consiste nos eventos possíveis para operação do sistema como um todo (Figura 59).

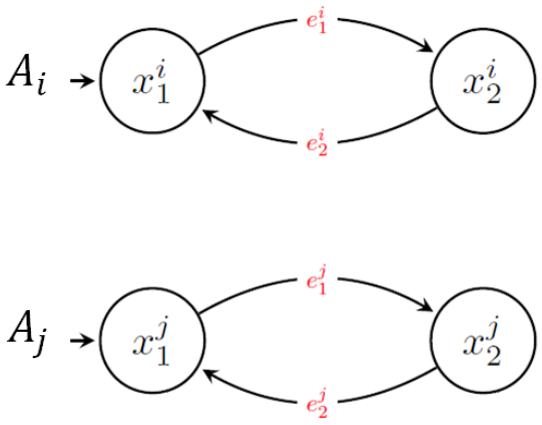


Figura 59: Autômatos simples A_i e A_j, como parte de um sistema.

Existe um subconjunto de eventos presentes em A_j que são permitidos para um estado determinado de A_i, cuja ocorrência não o altera. Seja determinado estado $x_1^i \in X_i$. Um subconjunto $E_j'(x_1^i) \in E_{ij}$ é um grupo de eventos permitidos no estado x_1^i, para os quais é válida a relação

$$f(x_1^i,e \in E_j') = x_1^i. \tag{5.1}$$

Para cada estado de A_i e A_j deve-se determinar quais eventos de E_j e E_i são permitidos, de modo a incluir na modelagem dos autômatos as funções de transição para esses eventos, fazendo com que os conjuntos de transições f_i e f_j se tornem

$$f_i = f_i + \{f(x \in X_i, e \in E'_j(x)) = x\} \quad (5.2)$$

$$f_j = f_j + \{f(x \in X_j, e \in E'_i(x)) = x\}, \quad (5.3)$$

transformando o modelo da Figura 59 no modelo da Figura 60, que será usado como entrada do sistema.

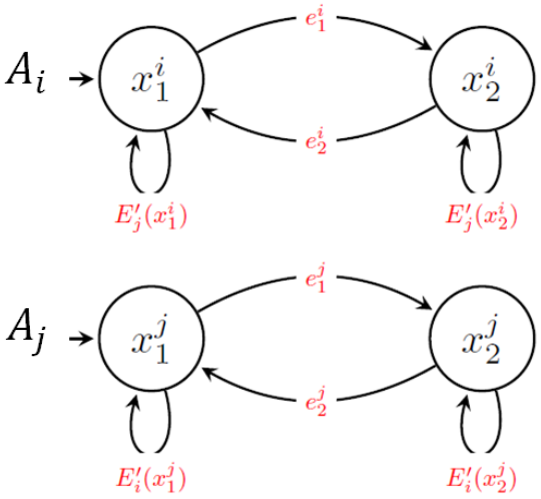


Figura 60: Autômatos simples A_i e A_j, utilizando *DeSCART*

Esta técnica de restrição é de extrema importância para o funcionamento do método proposto. As atribuições impostas, sob forma de funções de transição, garantem a segurança dos componentes e pode ser considerada com um recurso de sincronia dos eventos que devem ocorrer no sistema como um todo. DeSCART garante que na união dos autômatos, realizada pela plataforma, os estados indesejados ou que não foram previstos na modelagem se tornem inalcançáveis. Esta propriedade é muito desejável, visto que não é necessário a criação de um conjunto arbitrário de estados bloqueados. Onde os grafos gerados pela metodologia são utilizados como dados de entrada na plataforma, descritos na forma de conjuntos.

5.4 PLATAFORMA PARA INSERÇÃO DE AUTÔMATOS

Foi desenvolvida uma plataforma com interface gráfica usando a linguagem C#, em ambiente Windows. O C# é uma poderosa linguagem de programação orientada a objetos, que possui grande importância no *framework* ".NET"(*dot net*), que será utilizado na construção da interface gráfica. Para programação desta aplicação foi utilizado o Visual Studio 2012 com ambiente de programação para construção de executáveis para

plataforma Windows. Contudo, a linguagem C# e o *framework* ".NET"possuem portabilidade para sistemas UNIX (como linux e mac), tanto 32 e 64 bits, tornando a aplicação a mais genérica possível.

A plataforma foi desenvolvida para receber autômatos representados na forma de conjuntos (eventos, estados, funções de transição), capaz de unir autômatos (dois a dois) utilizando à composição paralela. Considerando o diagrama de estados da Figura 61, as especificações para este autômato (dados de entrada da plataforma) são dadas por:

$$A_{fini} = \{X,E,f,x_0,x_{obj}\} \tag{5.4}$$

$$X = \{q_0,q_1,q_2,q_3\} \tag{5.5}$$

$$E = \{0,1\} \tag{5.6}$$

$$x_0 = \{q_0\} \tag{5.7}$$

$$x_{obj} = \{q_2\} \tag{5.8}$$

$$f = \{[q_0,0,q_2],[q_1,0,q_3],[q_2,0,q_0],[q_3,0,q_1],[q_0,1,q_1], [q_1,1,q_0],[q_2,1,q_3],[q_3,1,q_2]\} \tag{5.9}$$

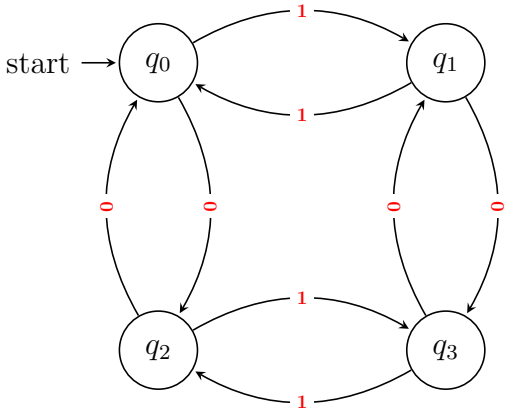


Figura 61: Diagrama de estados do autômato finito A_{fini}

Durante o cadastro dos autômatos, a plataforma adota algumas medidas preventivas indicando erro na entrada dos dados (caso necessário). O usuário fica impedido de tentar cadastrar autômatos com nomes iguais e conflitos dentro do mesmo autômato, por exemplo; estados, funções e eventos iguais, além de funções que levem o modelo a ser não-determinístico.

A plataforma permite que o usuário adicione a quantidade de estados, eventos, e

funções que desejar, indicando nos campos referentes a cada conjunto quais elementos foram cadastrados até o momento (Figura 62).

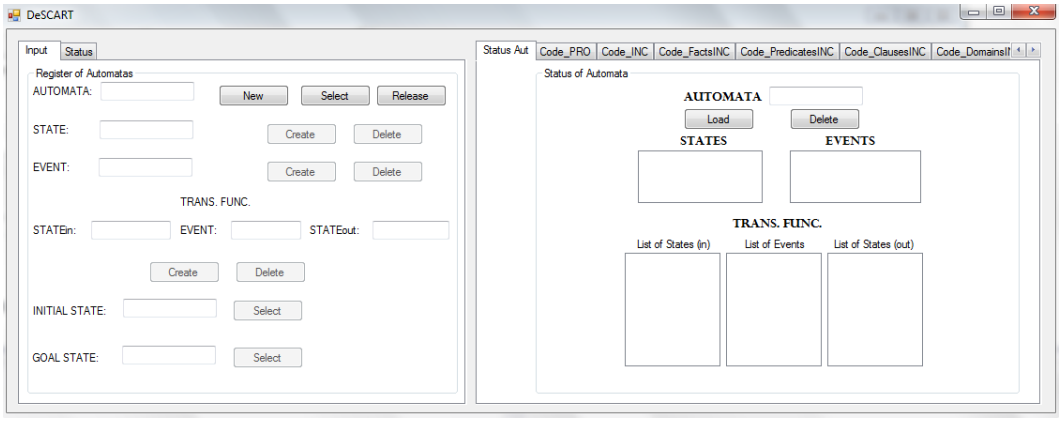


Figura 62: Entrada de dados da plataforma

Outra funcionalidade da ferramenta se encontra na possibilidade de que após entrar com os dados obtidos pelo método de modelagem proposto, o usuário pode escolher gerar um código PROLOG e editá-lo, para qualquer sistema modelado desejado sem a necessidade de unir os sub-sistemas cadastrados (Figura 63).

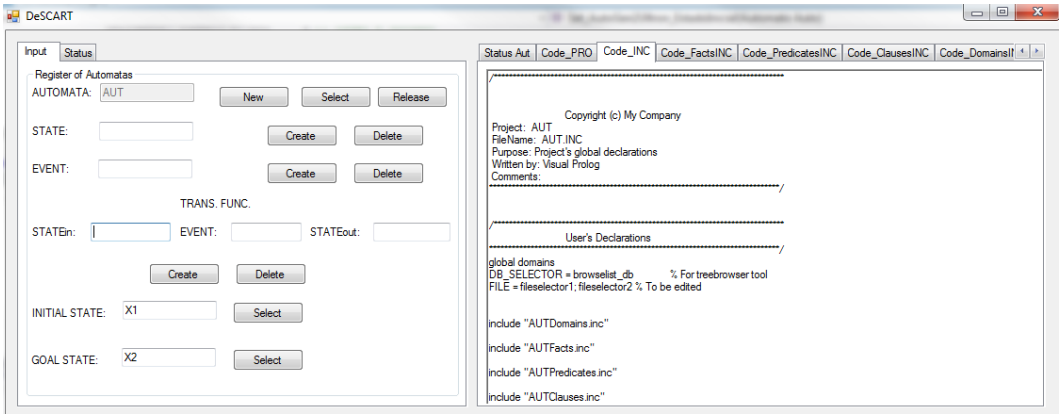


Figura 63: Edição de código gerado pela plataforma

Onde cada aba representa um arquivo gerado necessário para executar a RP por meio do mecanismo de inferência. Apresentadas essas definições, o próximo passo é a representação das Redes de Petri geradas com o programa proposto.

5.5 CONSTRUÇÃO DO AUTÔMATO GLOBAL

Após o cadastrado de todos os autômatos que representam os componentes do sistema, algumas operações devem ser aplicadas para a obtenção de A_G, o autômato

que representa o sistema como um todo. As operações são responsáveis por modelar a forma de comportamento conjunto de um dado número de autômatos que operam simultaneamente. Buscando modelar uma composição síncrona (CASSANDRAS, 2008), a composição paralela (Figura 64) é a operação escolhida para realizar a união dos autômatos e impor restrições de atuação no sistema, definida como

$$A_1 \parallel A_2 = Ac(X_1 \times X_2, E_1 \cup E_2, f((x_1 x_2), e), \Gamma_{1||2}, x_{01} x_{02}, x^1_{obj} x^2_{obj}). \tag{5.10}$$

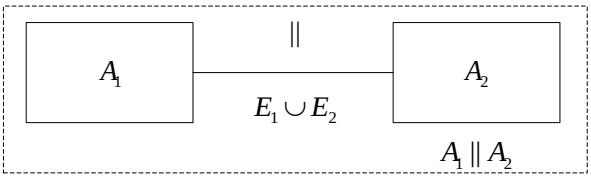


Figura 64: Paralelismo de Autômatos

Na Equação 5.10, tem-se dois autômatos A_1 e A_2, onde Ac é chamada de *operação acessível* (operação que elimina todos os Estados que não são acessíveis a partir do estado inicial) (BOGDAN FRANK L. LEWIS; JR., 2006), obtendo um conjunto de estados resultante que contém todas as combinações feitas pelos estados em A_1 e A_2.

O novo conjunto de eventos, obtido pela composição paralela, é resultado da união de eventos em A_1 e A_2. Logo a *função de transição* do autômato em questão é definido como:

$$f((x_1 x_2), e) = \begin{cases} (f_1(x_1, e), f_2(x_2, e)) & if \quad e \in \Gamma_1(x_1) \cap \Gamma_1(x_2) \\ (f_1(x_1, e), x_2) & if \quad e \in \Gamma_1(x_1) \backslash E_2 \\ (x_1, f_2(x_2, e)) & if \quad e \in \Gamma_2(x_2) \backslash E_1 \end{cases} \tag{5.11}$$

A representação do sistema requer questões a serem resolvidas, por exemplo, identificar estados ilegais (*deadlocks e estados inacessíveis*). Isso pode ser difícil, especialmente se o sistema for grande e tiver centenas de estados.

Em alguns exemplos os *deadlocks* e estados inacessíveis são causados por uma propriedade estrutural do sistema e eles podem ser reconhecidos quando o modelo autômato do sistema é construído. Um estado inacessível é formalmente definido por (5.12), e um estado *deadlock* por (5.13).

$$\overline{x}_{ac} = x \in (X - \{x_0\}) | f(x_{pre},e) \neq x, \quad \forall e \in E, x_{pre} \in X \tag{5.12}$$

$$x_{dl} = x \in (X - \{x_{obj}\}) | f(x,e) = \varnothing, \forall e \in E \tag{5.13}$$

Por uma determinação errada dos estados inicial e objetivo pode ocorrer que $\overline{x}_{ac} = x_{obj}$ ou que $x_{dl} = x_0$, isto é, o estado objetivo é inalcançável ou o estado inicial é um *deadlock*. A plataforma prevê este tipo de situação, apresentando uma *flag de erro* ao usuário.

A plataforma lida com esses estados ilegais como estados a serem excluídos do sistema, fazendo uma busca por todo o modelo e excluindo x_{dl} e $\overline{x}_{ac}$. Também são eliminadas as funções de transição $f(\overline{x}_{ac}, \forall e \in E)$ e $f(\forall x \in X, \forall e \in E) = x_{dl}$.

5.6 CONSTRUÇÃO DO SISTEMA DE REGRAS

Após a eliminação dos estados ilegais, o autômato A_G está pronto para ser transformado em uma Rede de Petri, representada por um sistema de regras. Nesta seção a construção será exemplificada através do módulo *Distributing*.

Seja RP_G a Rede de Petri análoga ao autômato A_G. A transformação dos estados X de A_G para o conjunto P de RP_G não é tão simples, por ser possível representar os *places* de diferentes maneiras. A representação adotada, considera que a quantidade de estados presentes no conjunto X pode ser representada como uma função exponencial, $f(n) = 2^n$, onde n indica a quantidade de *places* contidos em P, e o resultado da função a quantidade de estados na linguagem autômata. Logo, os atuadores e sensores que representam os estados do A_G para *Distributing* podem ser combinados em pares e transformados em um único *place* para cada atuador/sensor como apresentado nas equações (5.14) a (5.18);

$$p_1 = x_{ALR} + x_{ALA} \tag{5.14}$$

$$p_2 = x_{GM} + x_{GE} \tag{5.15}$$

$$p_3 = x_{SV} + x_{CV} \tag{5.16}$$

$$p_4 = x_{SS} + x_{CS} \tag{5.17}$$

$$p_5 = x_{CP} + x_{SP} \tag{5.18}$$

O módulo passa a ser representado por 5 *places*, ao invés de possíveis 32 estados, que podem ou não conter *tokens*. Sendo a transformação de eventos E de A_G para transições T de RP_G direta, assim como o estado inicial x_0 para m_0, como apresentado nas equações (5.19) e (5.20); É importante observar que como esta modelagem é na forma de sistema de regras (os quais podem ser representados na forma de Redes de Petri) as transições T apresentadas são referentes as regras que existem em todo o sistema.

$$T = E \tag{5.19}$$

$$m_0 = x_0 \tag{5.20}$$

A construção do conjunto de funções I e O pode ser feita através da análise da definição da função de transição f. Se,

$$f : X_{pre} \times E \rightarrow X_{pos}, \tag{5.21}$$

então,

$$I : \ T \rightarrow X_{pre} \tag{5.22}$$

$$O : \ T \rightarrow X_{pos}. \tag{5.23}$$

As funções I e O serão transformadas em regras, para a composição do sistema de regras: os conjuntos P e T compõe a *base de fatos*, as funções I e O a *base de regras*, restando apenas definir o mecanismo de inferência. Durante a implementação da modelagem das Redes de Petri em Prolog, de modo a facilitar a implementação do algoritmo de busca, os *places* são analisados em conjunto. Logo, na linguagem de programação lógica, uma variável do predicado genérico, X_N, é definida como:

$$X_N = [p_1; p_2; p_3; p_4; p_5] \tag{5.24}$$

5.7 MODELAGEM DA REDE DE PETRI EM PROLOG

A Rede de Petri modelada a partir do autômato global será desenvolvida na forma de um *sistema de regras*, logo é necessário definir um mecanismo de inferência. Este mecanismo consiste em uma ferramenta capaz de organizar o conjunto de regras em forma de lista, e percorrê-la sequencialmente. A regra é aplicada ou disparada se a parte *condição* da regra for verdadeira, caso contrário (nenhuma regra aplicável) o mecanismo de inferência pára seu funcionamento.

Na plataforma proposta o mecanismo de inferência utilizado foi o **Visual Prolog**. A programação lógica foi escolhida por ser amplamente utilizada por programadores para aplicações de computação simbólica (CLOCKSIN; MELLISH; CLOCKSIN, 1987), incluindo: Bancos de dados relacionais, lógica matemática, resolução de problemas abstratos, automação de projetos, resolução de equação simbólica, e muitas áreas da inteligência artificial.

Tornando-se especialmente indicado para a solução de problemas que envolvem objetos e relações entre os objetos.

Em (JIANG; KUMAR, 2006) o autor utiliza uma técnica baseada na ramificação lógica de de tempo completo por causa de que a cada momento podem existir caminhos alternativos que representam diferentes possíveis futuros, utilizando também da técnica de autômatos em árvore.

Sendo $M = (Q, AP, R, L)$ um grafo de transição de estados, chamado de estrutura *Kripke*, onde Q é o conjunto de estados (finito ou infinito), AP é um conjunto finito de símbolos de proposição atômica, $R \subseteq Q \times Q$ é a relação total de transições, ou seja, para cada $s \in Q$ há um $s' \in Q$ tal que $R(s,s')$, e $L : Q \to 2^{AP}$ é uma função que rotula cada estado com um conjunto de proposições que são verdadeiras naquele estado.

Usando as proposições atômicas e conectivos booleanos como conjunção, disjunção e negação, o autor conclui em seu trabalho que pode construir expressões mais complexas que descrevem propriedades de estados. No entanto, também está interessado em descrever as propriedades de sequências (e mais geralmente de estruturas de árvore) de estados que o sistema pode visitar. A lógica temporal é um formalismo para descrever propriedades de sequências de estados, bem como de estruturas de árvore de estados. Tais propriedades são expressas usando operadores temporais e quantificadores de caminho da lógica temporal.

Já a lógica de predicados foi desenvolvida para transmitir facilmente ideias baseadas

em lógica em forma escrita. PROLOG aproveita esta sintaxe para desenvolver uma linguagem de programação baseada em lógica. Na lógica de predicados, primeiro você elimina todas as palavras desnecessárias de suas sentenças. Então, transforma-se a sentença, colocando o relacionamento em primeiro lugar e agrupando os objetos após a relação. Os objetos, em seguida, tornam-se argumentos de que o relacionamento age em cima. Por exemplo, as seguintes frases são transformados em sintaxe lógica de predicados:

Linguagem natural	**Lógica de predicado**
O carro é engraçado	$engracado(carro)$.
A rosa é vermelha	$vermelho(rosa)$.
Bill gosta de carros se e eles forem engraçados	$gosta(bill, Carro)$ if $engracado(Carro)$

Após o programador definir os objetos e relações, então é possível determinar as regras sobre essas relações quando forem verdadeiras. Por exemplo:

$$\begin{aligned} gosta \quad & (ellen, tennis). \\ gosta \quad & (john, football). \\ gosta \quad & (tom, baseball). \\ gosta \quad & (eric, nadar). \\ gosta \quad & (mark, tennis). \end{aligned}$$

$$gosta(bill, Atividade) :- gosta(tom, Atividade).$$

Note, a regra afirma que Bill só vai gostar de determinada atividade se Tom também gostar da atividade, logo, Bill gosta de *baseball*.

Com o intuito de um melhor intendimento, um exemplo didático pode ser aplicado em um caso de estudo (Módulo *Distributing*).

Após a plataforma determinar um autômato A_G^{DIS}, representado pela figura 65, tem-se os objetos na forma dos estados x_i e suas relações expressas na forma de arcos.

Como na representação deste modelo o intuito não é indicar relações de parentesco entre os nós do diagrama, e sim realizar uma busca por toda a árvore de possibilidades, adota-se um objeto único (5.25). Sendo os argumentos deste objeto os estados indicados

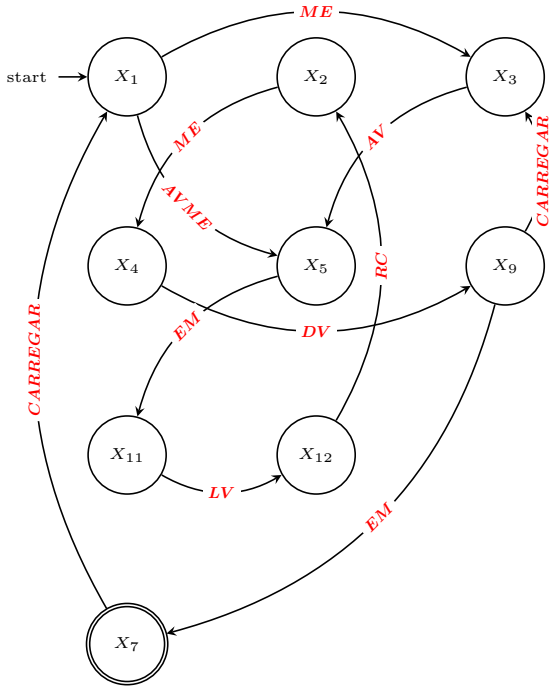


Figura 65: Diagrama de estados do autômato global - Distributing

pelo diagrama:

$$rm(x_1,x_2,x_3,...,x_{11},x_{12}) \tag{5.25}$$

Tal que:

$$x_1,x_2,x_3,...,x_{11},x_{12} = integer$$

O objeto rm representa os ramos da árvore de possibilidades, assumindo valores inteiros para receber ou enviar *tokens* durante sua evolução de estados. Note que também seria possível representar o diagrama como proposto na seção 5.6, contudo, por motivos implementação da técnica de busca (será tratada mais adiante) se adotou esta representação.

As relações expressas na forma de arcos, são chamadas de funções de evolução. Para cada arco existe uma regra indicando que é "verdadeiro" a evolução de determinado x_i para outro x_j. Por exemplo, se o programa proposto indica que $x_1 = 1$ (há um *token* naquela posição) as funções de evolução que existem, segundo o diagrama, saindo deste estado para outros são ME e $AVME$, podendo ser representadas de forma simples

pelas regras:

$$funcEvo(x_1,x_3) : -x_1 = 1.$$
$$funcEvo(x_1,x_5) : -x_1 = 1.$$

Para realizar a busca utilizando métodos de otimização, foi necessário modificar as regras de forma que auxiliassem na aplicação do algoritmo proposto. As regras são construídas de modo que "sub-regras", encapsuladas na função de evolução, retornem "verdadeiro" para então as regras principais serem aplicadas. Partindo da ideia que cada função de transição possui pré-sets e pós-sets, a plataforma faz um *loop* para cada conjunto, [*pré-set;evento;pós-set*], existente na lista de funções de transição possíveis.

As funções de transição (funções de evolução) contendo o evento ME, por exemplo, são representadas então na forma de:

```
% FUNÇÃO DE TRANS - ME
next_state(Estado, NovoEstado):-
retornavalores(Estado,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12),
X1= 1,
NovoEstado =     rm(0,0,1,0,0,0,0,0,0,0,0,0).

% FUNÇÃO DE TRANS - ME
next_state(Estado, NovoEstado):-
retornavalores(Estado,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12),
X2= 1,
NovoEstado =     rm(0,0,0,1,0,0,0,0,0,0,0,0).
```

Logo, a regra que representa uma determinada função de transição só irá retornar verdadeiro se; ao ser chamado o predicado *next_state* obter verdadeiro de todas as suas condições (o valor cujo *retornavalores* indicar deve ser obrigatoriamente igual ao valor XN de pré-set da função, e o valor do pós-set indicado pela variável *NovoEstado*).

Desejando obter um conjunto de operações que permitam levar o estado de x_0 a x_{obj}, a Rede de Petri apresentada é similar a um *grafo direcionado de transição de estado* devido a sua característica de ser finito, modelado sem equações diferenciais, e sem incerteza. Sendo assim, um planejador discreto pode ser utilizado para determinar

um plano; uma vez que é difícil fazer uma distinção clara entre a resolução de problemas e planejamento, nós podemos simplesmente nos referir a ambos como planejamento.

Existem definições diferentes, porém muito similares, de planejar e resolver um problema (LAVALLE, 2006). Como por exemplo, quando agentes de resolução de problema decidem o que fazer encontrando uma sequência de ações que o levam a um estado desejado. Sendo considerado planejamento a tarefa de surgir com uma sequência de ações que vão alcançar um objetivo. Podendo ser definido como:

- Espaço de estado (não vazio) - X, contendo um conjunto de estados finitos
- Para cada estado $x \in X$, tem-se um espaço de ações finito - $U(x)$
- Uma função de transição de estado f que produz um estado $f(x,u) \in X$ para todo $x \in X$ e $u \in U(x)$. Sendo $x' = f(x,u)$.
- Existe um estado inicial $x_1 \in X$
- Existe um estado objetivo $x_{obj} \in X$

Considerando o estado inicial e o estado objetivo como sendo vértices especiais de um determinado grafo, pode-se representar a definição acima como na Figura 66

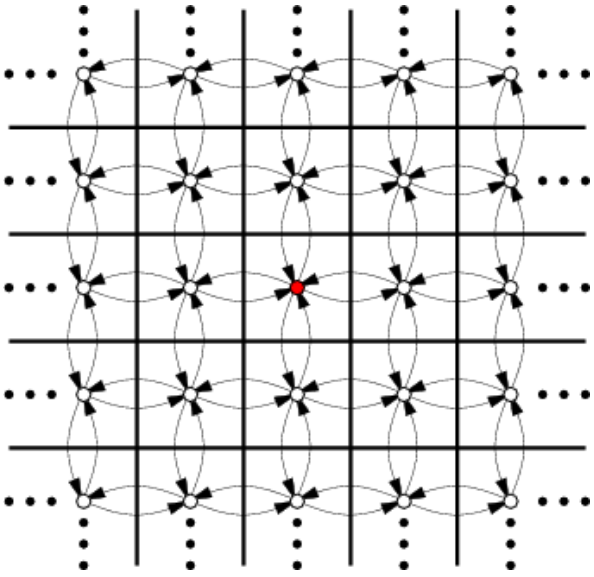


Figura 66: Grafo de transição de estados para exemplos envolvendo planos infinitos (LAVALLE, 2006)

Compreendendo que o grafo de transição de estados é revelado incrementalmente através da aplicação de ações, uma exigência para o algoritmo de busca é que seja

sistemático. Se o grafo é finito, isso significa que o algoritmo vai visitar todos os estados, o que permite definir quando uma solução existe ou não, em um tempo finito.

Para ser sistemático o algoritmo precisa saber quais os estados já foram visitados, de outro modo poderia ficar preso em ciclos e rodar pra sempre. Garantindo que a busca não seja redundante é suficiente para garantir que o algoritmo seja sistemático.

O algoritmo que representa essa busca geral pode ser representado pela Figura 67

Algorithm 1: FORWARD_SEARCH

```
1  Q.Insert(x_1) and mark x_1 as visited
2  while Q not empty do
3      x ← Q.GetFirst()
4      if x ∈ X_G then
5          return SUCCESS
6      end
7      forall the u ∈ U(x) do
8          x' ← f(x, u)
9          if x' not visited then
10             Mark x' as visited
11             Q.Insert(x')
12         end
13         else
14             Resolve duplicate x'
15         end
16     end
17 end
18 return FAILURE
```

Figura 67: Algoritmo de busca exaustiva
(LAVALLE, 2006)

Sendo que este padrão de algoritmo de busca, trabalha com três tipos de estado: Os *estados não-visitados* são aqueles que não foram visitados ainda. No inicio todos os estados, menos x_1, são considerados não visitados, os *estados mortos* são aqueles que já foram visitados e cada próximo estado possível foi visitado também, e por último os *estados vivos* que são os estados encontrados que possuem estados adjacentes não vistados (sendo x_1 o único estado vivo inicialmente).

A fila Q, local onde os estados vivos estão armazenados, apresentada no algoritmo é trabalhada de diferentes formas de acordo com cada mecanismo de busca, onde a diferença fundamental entre os algoritmos é a função de prioridade usada para ordenar Q (LAVALLE, 2006). Outros algoritmos podem ser encontrados também em (CHOSET, 2005), e o algoritmo escolhido foi o $A*$ (Figura 68).

Algorithm 24 A^* Algorithm

Input: A graph

Output: A path between start and goal nodes

1: **repeat**
2: Pick n_{best} from O such that $f(n_{best}) \leq f(n), \forall n \in O$.
3: Remove n_{best} from O and add to C.
4: If $n_{best} = q_{goal}$, EXIT.
5: Expand n_{best}: for all $x \in \text{Star}(n_{best})$ that are not in C.
6: **if** $x \notin O$ **then**
7: add x to O.
8: **else if** $g(n_{best}) + c(n_{best}, x) < g(x)$ **then**
9: update x's backpointer to point to n_{best}
10: **end if**
11: **until** O is empty

Figura 68: Algoritmo de busca A^*
(CHOSET, 2005)

O número de estados em uma estrutura de transição para especificar as trajetórias de eventos admissíveis pode depender exponencialmente em algum parâmetro do sistema. Em tais casos, algoritmos simples para verificação ou síntese (por exemplo, procura de espaço de estados) tornar-se rapidamente computacionalmente intratável (RAMADGE; WONHAM, 1989).

Sendo uma RP um tipo de árvore de estados, foi implementado o algoritmo na linguagem PROLOG, com o objetivo de encontrar uma solução ótima para atingir o estado x_{obj}. Esta solução ótima se torna necessária em ambientes industriais, os quais desejam que seu sistema consiga trabalhar com segurança e produzir um produto com o mínimo de insumos em um tempo hábil.

A escolha do algoritmo é justificada por ser uma *best-first search* e garantir encontra um caminho de menor custo a partir de um determinado nó inicial a um nó objetivo (de uma ou mais metas possíveis). O A^* atravessa o grafo, segue-se um caminho de custo ou distância total, menor que o esperado, mantendo uma fila de prioridade ordenada (Q) de segmentos de caminhos alternativos ao longo do caminho. Um rico detalhamento sobre este algoritmo e muitos outros semelhantes podem ser encontrados em (CHOSET, 2005) e (LAVALLE, 2006).

Neste presente trabalho, o A^* implementado em programação lógica trata o grafo de estados como um "labirinto", sendo cada nó um *room* a ser visitado. Após todas as funções de evolução e predicados serem criados, é possível aplicar a busca passando para o algoritmo estado inicial e o objetivo que se deseja alcançar, na forma da função 5.26, e ele retorna o melhor caminho a ser realizado.

$$maze(EstadoInicial,EstadoFinal,Path) \tag{5.26}$$

Onde *EstadoInicial* é o ponto de partida da busca, *EstadoFinal* o objetivo final (responsável por finalizar a pesquisa quando for encontrado), e *Path* é a saída/caminho à ser realizado. Internamente os *rooms* são representados pela função 5.27 (similar a fun. 5.25, contudo agora representam variáveis e portanto são escritas em caixa-alta) e os caminhos parciais em Q (*Agenda*) são representados pela função 5.28, onde "F" é o valor de $f(n)$ do nó na "cabeça" do caminho parcial, "G" é o valor de $g(n)$, e "List" é uma lista de *rooms* a qual representa o percurso parcial, na ordem inversa habitual.

$$rm(X1,X2,X3,...,X12) \tag{5.27}$$

$$pp(F,G,List) \tag{5.28}$$

A função 5.28 acaba por ser conveniente para armazenar os valores tanto de $f(n)$ e $g(n)$ em cada caminho parcial. A rigor, só $g(n)$ precisa realmente ser armazenados ($f(n)$ sempre pode ser recalculado a partir de $g(n)$ e $h(h)$); onde $h(n)$ é a distância Manhattan entre um *room* e um outro *room* alvo.

Após chamar a função 5.26, a regra que se aplica para esta função é determinada como:

```
1 maze(StartRoom,EndRoom,Solution) :-
    setTarget(EndRoom),
3   astar([pp(_,0,[StartRoom])],ReversedSolution),
    reverseList(ReversedSolution,Solution).
```

O *setTarget* é um procedimento utilitário que estabelece a meta da pesquisa na árvore de estados como um "*target*(rm(0,0,0,0,0,0,1,0,0,0,0,0))" (o objetivo é alcançar o x_7 do grafo de estados), limpando primeiramente todos os alvos que sobraram de corridas anteriores.

```
setTarget(Target) :-
2   retractall(target(_)),
```

```
  assert(target(Target)).
```

A função *astar*(*Agenda*,*ReversedSolutionPath*) representa o código do algoritmo A^*, indicando que possui uma entrada (agenda de pesquisa atual) e retorna uma saída (caminho para solução, retornado em ordem inversa). Sendo a cláusula 1 a cláusula de terminação, a qual detecta o nó meta à frente do primeiro caminho parcial na agenda, e a cláusula 2 assume que a busca deve continuar, gerando todos os sucessores legais do nó na "cabeça" do primeiro caminho parcial da agenda. Adicionando-os de forma apropriada na classificação dos caminhos parciais.

```
1 astar([pp(_,_,[Goals|RestOfPath])|_],[Goals|RestOfPath]) :-
    target(Goals).
3
  astar([PP|OtherPPs],Rsol) :-
5   findall(Successor,makeSuccessor(PP,Successor),NewPPs),
    makeNewAgenda(NewPPs,OtherPPs,Agenda),
7   astar(Agenda,Rsol).
```

O *makeSuccessor*(*PartialPath*,*NewPartialPath*) é um procedimento que tem o *PartialPath* como entrada e retorna um novo caminho parcial (um sucessor para o *PP* de entrada, o qual ele retorna como saída *NewPartialPath*). Já o procedimento *makeNewAgenda*(*NewPPs*,*CurrentAgenda*,*NewAgenda*) é responsável por adicionar os novos caminhos parciais na agenda corrente, criando uma nova agenda de busca.

```
1 makeSuccessor(pp(_,G,[rm(X1,X2,...,X11,X12)|Rooms]),pp(Fn,Gn,[rm(X1new,
    X2new,...,X11new,X12new),rm(X1,X2,...,X11,X12)|Rooms])) :-
  next_state(rm(X1,X2,...,X11,X12), NovoEstado),
3 retornavalores(NovoEstado,X1new,X2new,...,X11new,X12new),
  not(member(rm(X1new,X2new,...,X11new,X12new),[rm(X1,X2,...,X11,X12)|
    Rooms])),    /* cycle-check */
5 pegag(rm(X1,X2,...,X11,X12),rm(X1new,X2new,...,X11new,X12new),G,Gn),
    /* new g(n)     */
  Fn = Gn.
```

```
makeNewAgenda([],Agenda,Agenda) :- !.
2
makeNewAgenda([NewPP|OtherNewPPs],CurrentAgenda,FinalAgenda) :-
```

```
4    insert(NewPP, CurrentAgenda, Agenda),
     makeNewAgenda(OtherNewPPs, Agenda, FinalAgenda).
```

Sendo *insert*(*newPP*,*Agenda*,*NewAgenda*) a função que insere um único caminho parcial por vez na agenda, busca sempre manter uma posição correta na lista de acordo com seu valor de função de avaliação.

```
1  insert(NewPP,[],[NewPP]) :- !.

3  insert(NewPP,[PP|OtherPPs],[NewPP,PP|OtherPPs]) :-
     lt(NewPP,PP), !.
5
   insert(NewPP,[PP|OtherPPs],[PP|Merged]) :-
7    insert(NewPP,OtherPPs,Merged).
```

Onde *lt*(*PP*1,*PP*2) e *reverseList*(*A*,*B*) são procedimentos auxiliares para mesclagem e inversão de ordem dos elementos respectivamente. E a função *member* sendo utilizada para verificar se determinado nó se encontra na lista de nós já verificados.

```
1  lt(pp(F1,_,_),pp(F2,_,_)) :- F1 < F2.
```

```
1  reverseList(A,B) :- revaux(A,[],B).

3  revaux([],Acc,Acc) :- !.
   revaux([Ha|Ta],Acc,B) :- !,
5    revaux(Ta,[Ha|Acc],B).
```

```
1  member(Name, [Name|_]).

3  member(Name, [_|Tail]):-
   member(Name, Tail).
```

O algoritmo roda enquanto não encontrar o objetivo determinado; caso não encontre o objetivo e todos os ramos tenham sido visitados ele para e indica um resultado

falso. Entretanto, se o objetivo escolhido for encontrado, o programa retorna o melhor caminho do estado inicial para o estado objetivo.

Em (QUEIROZ; CURY, 2002) o autor afirma que não é necessário para os supervisores conter a informação completa do comportamento da planta. Assim, para comandar a execução das sub-plantas modelados, suas máquinas de estado são implementadas simultaneamente em nível de sistema do produto. A transição controlada é considerada aceita se o seu estado anterior está ativo e os supervisores não indicarem que aquele é um estado bloqueado. Uma transição descontrolada é considerada aceita se o seu estado anterior e seu evento correspondente estão ativos. A evolução paralela das sub-plantas assíncronas seguem comandos executados (transições controláveis) e respostas (transições incontroláveis) dos procedimentos operacionais, sinalizando eventos para os supervisores.

O nível de Procedimentos Operacionais funciona como uma interface entre o sistema de produto teórico e real do sistema. Neste nível, o programa interpreta os comandos abstratos do Sistema de produtos como procedimentos lógicos que orientam o funcionamento de cada subsistema particular. Estes procedimentos de baixo nível geram os sinais de saída do sistema de controle e leem os sinais de entrada, fornecendo um sistema respostas lógicas que reflitam a ocorrência de eventos incontroláveis.

A resposta obtida pelo mecanismo de inferência é uma sequência de transições (Figura 69) que leva o sistema do estado x_0 a x_{obj}, de maneira ótima e eficiente, em relação ao número de operações possíveis e combinações para alcançar o objetivo desejado, com segurança e sincronia garantida pela modelagem DeSCART.

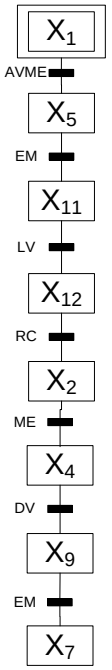


Figura 69: Código SFC resultante - *Distributing*

Após a determinação da sequência de transições, a plataforma é capaz de elaborar o código em *Sequential Function Chart - SFC* para ser aplicado em CLP's (Figura 70). Dentre as estabelecidas como padrão em (JOHN; TIEGELKAMP, 2001) foi escolhida SFC por sua estreita ligação com as RP's, além de permitir uma representação de estrutura do sincronismo bem representada (MORAIS, 2013).

A metodologia para a síntese de programas de controle de supervisão foi aplicado com sucesso em uma célula de produção comandado por um PLC em (QUEIROZ; CURY, 2002). A abordagem modular local, aliado a um algoritmo de minimização supervisor, é apontada como uma técnica muito vantajosa para a síntese de supervisores ótimos reduzidos.

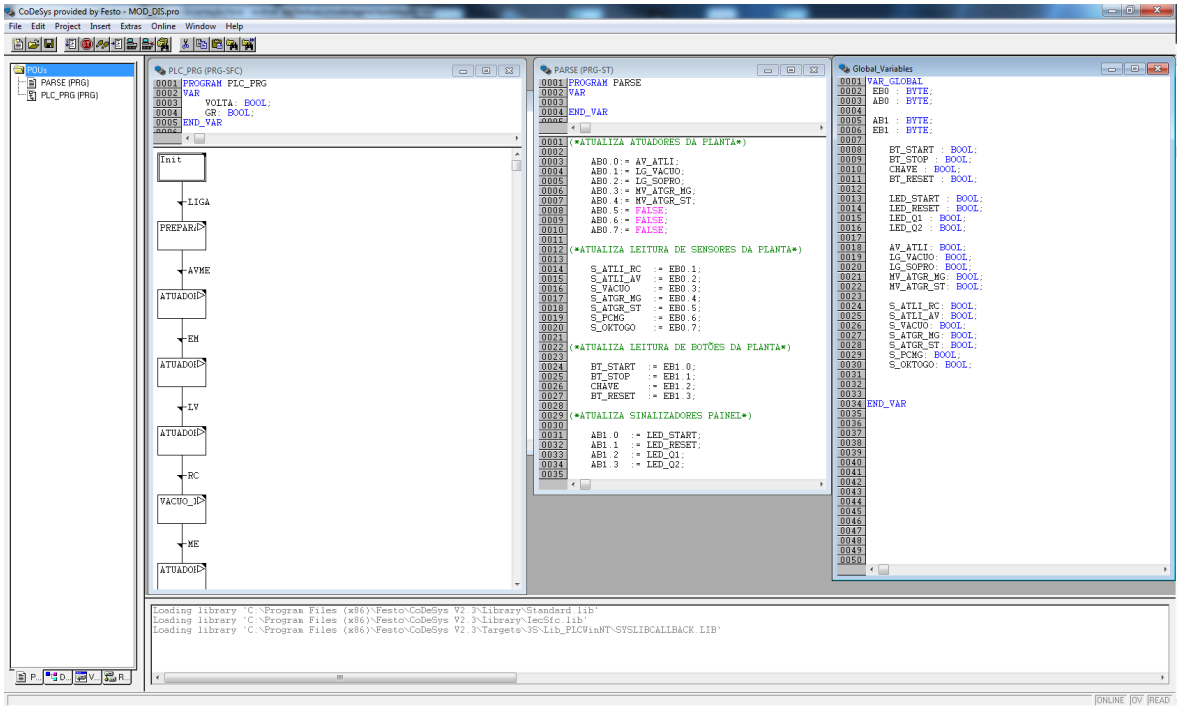


Figura 70: Código SFC aplicado no Software CoDeSys V2.3 - *Distributing*

6 RESULTADOS

6.1 INTRODUÇÃO

Após a apresentação da metodologia proposta no Capítulo 5, testes serão realizados para validar seus resultados. Para esse fim, cinco módulos de bancadas acadêmicas descritos no Capítulo 2 serão modelados segundo a metodologia DeSCART, cadastrados no aplicativo construído, e em seguida seu código em SFC será gerado.

A ferramenta utilizada em (KOSMATOPOULOS; CHRISTODOULOU, 1995) faz uso de uma linguagem baseada no processo de álgebra e redes de Petri, para a geração de especificações de processo com ênfase especial na sequência de operações e alocação de recursos. Esta linguagem de alto nível para sistemas a eventos discretos (SEDs), é chamado processo de Redes de Petri Algébrica (ou PPN).

A linguagem PPN é uma linguagem geral para a modelagem de um SED, mas no trabalho de Kosmatopoulos é utilizado para a modelagem de alto nível de especificações de roteamento de sistemas para alocação de recursos, também podendo ser formalmente convertido numa representação de autômatos.

Nas seções subsequentes as bancadas serão apresentadas resumidamente, descrevendo-se seus estados e sensores, e quais são as restrições usadas para modelar os autômatos dos subsistemas. Então, será apresentado cada autômato global A_G correspondente à estação e seu respectivo código de PLC.

6.2 DISTRIBUTING

A estação *Distributing* (descrita na Seção 2.2.1) tem como objetivo carregar uma peça de montagem que está no magazine para que possa ser entregue à próxima estação, o módulo *Separating*.

A estação é composta pelos atuadores linear e giratório, por um dispositivo de vácuo/sopro para segurar/expulsar a peça, e por sensores de vácuo, de posição do atuador linear, de posição no atuador giratório, e presença de peça no *buffer*. O estado dos atuadores e sensores compõe o estados do sistema, descritos pela tabela 1. As transições possíveis para esse sistema são apresentadas na tabela 2.

Tabela 1: Tabela de estados - Distributing

Sigla	**Significado**
ALR	Atuador Linear Recuado
ALA	Atuador Linear Avançado
GM	Atuador Giratório na Posição do Magazine
GE	Atuador Giratório na Posição da Próxima Estação
SV	O vácuo está Desligado
CV	O vácuo está Ligado
SS	O Sopro está Desligado
CS	O Sopro está Ligado
CP	A peça se encontra no Magazine ou há peça no sistema
SP	A peça está solta na próxima estação e não há mais peças no sistema

Tabela 2: Tabela de transições - Distributing

Sigla	**Significado**	e_i
AV	Avança Atuador Linear	e_1
RC	Recua Atuador Linear	e_2
ME	Atuador Giratório gira para a Posição da Próxima Estação	e_3
EM	Atuador Giratório gira para a Posição do Magazine	e_4
LV	O vácuo é Ligado	e_5
DV	O vácuo é Desligado	e_6
AVME	Avança Atu. Lin. e Atu. Gi. para próx. est.	e_7
CARREGAR	Estrada de peça no magazine	e_8

Para garantir a integridade dos atuadores é necessário impedir que o atuador linear avance quando o atuador giratório se encontra na posição magazine. Descrevendo os autômatos de cada sensor e atuador usando a metodologia DeSCART se chega à representação da Figura 71.

Os autômatos da Figura 71 são cadastrados no aplicativo e geram o autômato global, A_G^{DIS}, conforme a Figura 72:

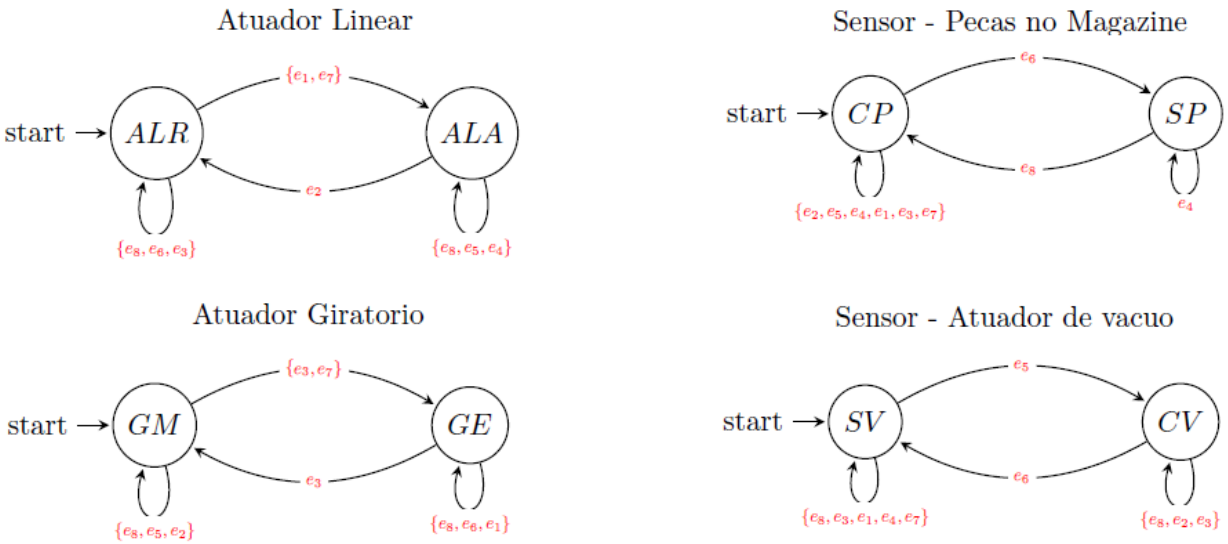


Figura 71: Autômatos simples da Estação *Distributing*

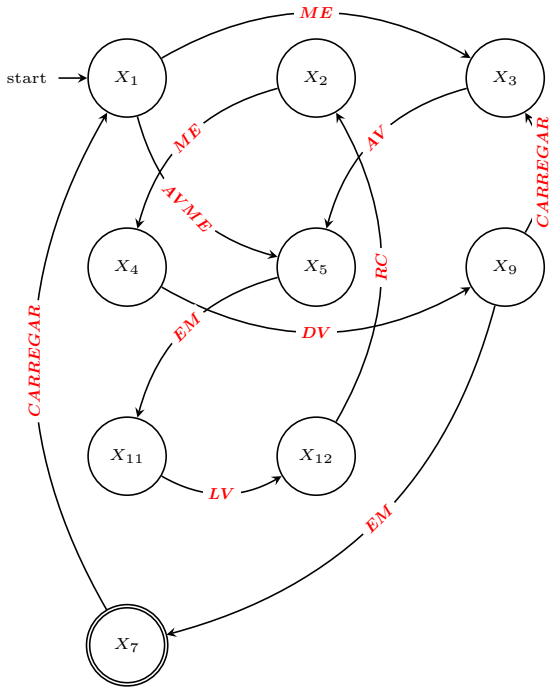


Figura 72: Diagrama de estados do autômato global - Distributing

O código otimizado resultante apresentado pelo mecanismo de inferência é mostrado na Figura 73, cuja legenda é apresentada na tabela 3. Note que o atuador e os estados relativos ao sopro não foram representados, por estarem subjetivos no evento *DV*.

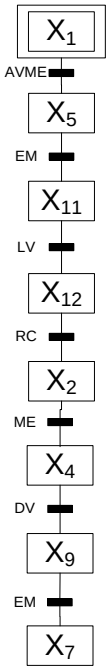


Figura 73: Código SFC resultante - *Distributing*

Tabela 3: Tabela dos estados do autômato global - *Distributing*

States	X_i^{DIS}
$ALRxGMxSVxCP$	X_1
$ALRxGMxCVxCP$	X_2
$ALRxGExSVxCP$	X_3
$ALRxGExCVxCP$	X_4
$ALAxGExSVxCP$	X_5
$ALRxGMxSVxSP$	X_7
$ALRxGExSVxSP$	X_9
$ALAxGMxSVxCP$	X_{11}
$ALAxGMxCVxCP$	X_{12}

Com o código SFC resultante, o próximo passo é transcrever essa resposta para o *software* **CoDeSys V2.3**. Este software em conjunto com o **CIROS Studio** permite ao programador testar seu código antes de transferi-lo para as bancadas, garantindo a integridade dos atuadores e obtenção do objetivo. É possível observar toda a comunicação dos *softwares* através do programa **EzOPC** (Figura 74), que auxilia na análise de sinais dos sensores e atuadores, simulando os bornes do CLP.

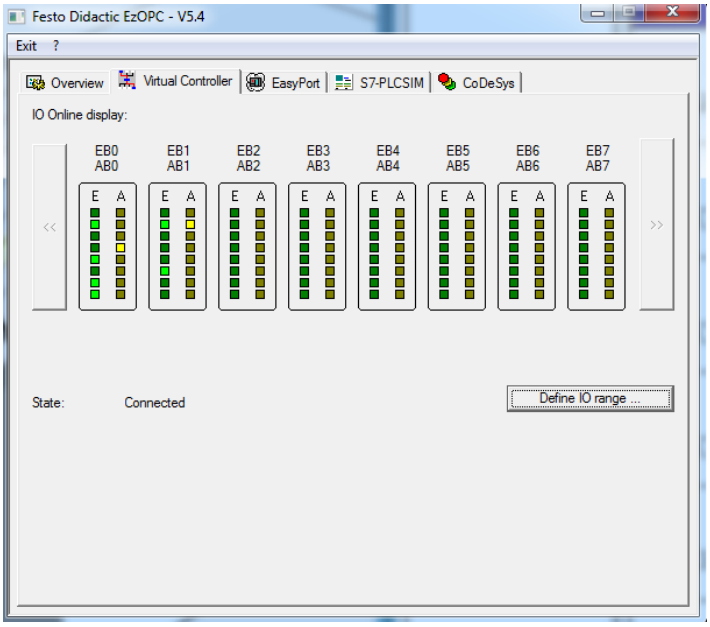


Figura 74: *Software* EzOPC indicando a comunicação dos programas **CoDeSys V2.3** e **CIROS Studio**

A figura 75 apresenta o resultado do código resultante programado no software **CoDeSys V2.3** e a simulação do mesmo no **CIROS Studio** paralelamente.

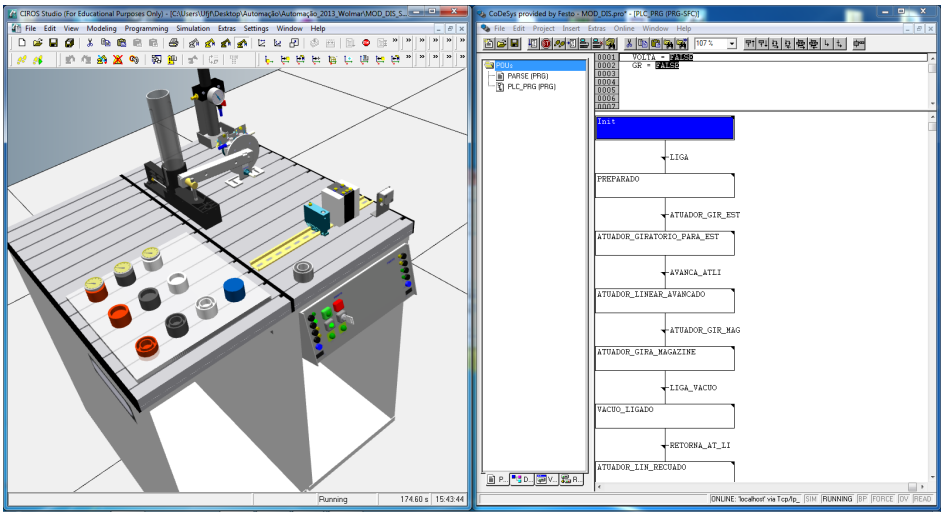


Figura 75: *Softwares* **CoDeSys V2.3** e **CIROS Studio** - iniciando a simulação do módulo *Distributing*

O programa de cada módulo é desenvolvido para começar a rodar após o usuário pressionar o botão *start* (Figura 76) e efetuar as próximas ações de acordo com o código resultante especifico de cada módulo (Figuras 77,78,79).

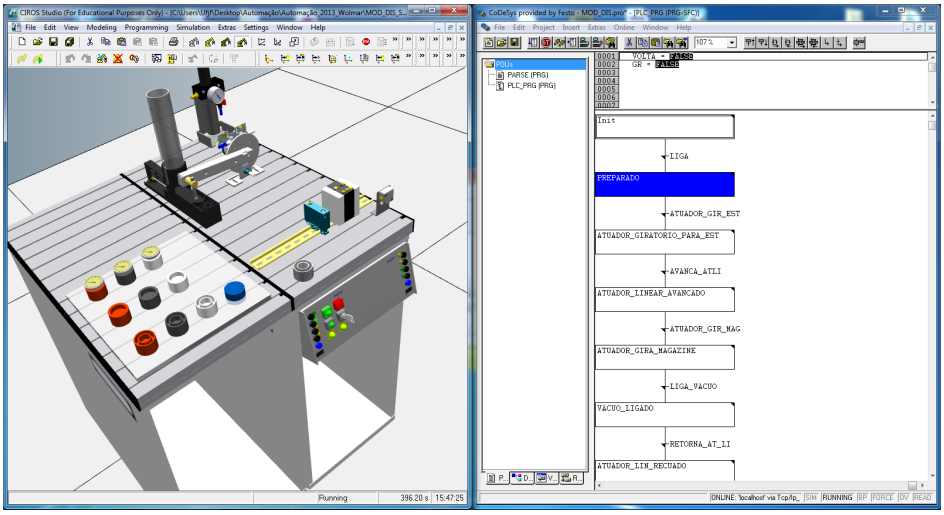


Figura 76: Módulo *Distributing* preparado para receber peça

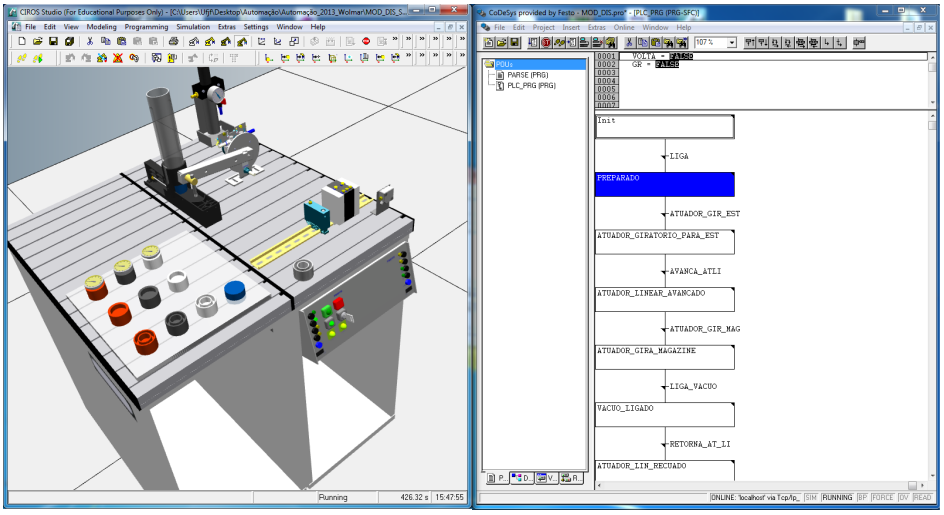


Figura 77: Módulo *Distributing* com peça no *buffer*

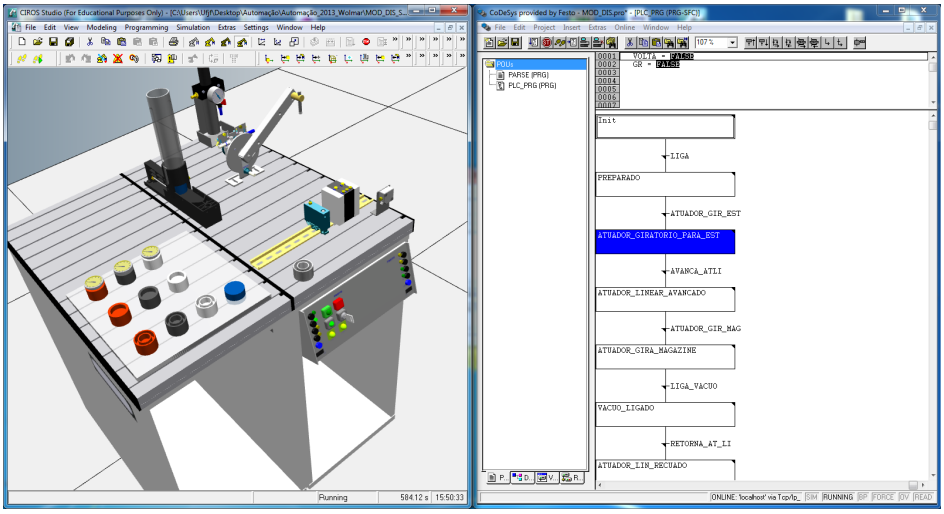


Figura 78: Módulo *Distributing* - atuador giratório para próxima estação

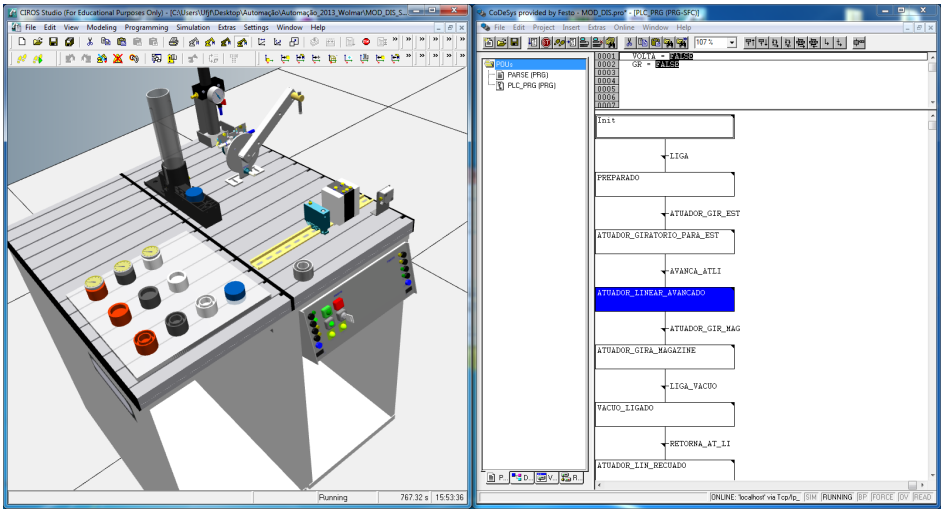


Figura 79: Módulo *Distributing* - atuador linear avançado

Após cada ação ser realizada, os retângulos vão indicando a evolução da simulação. As ações representadas pelas figuras 78 e 79 podem ser realizadas simultaneamente, como expresso no resultado final da resposta do código proposto pelo **Visual Prolog**, como na mostrado na Figura 80:

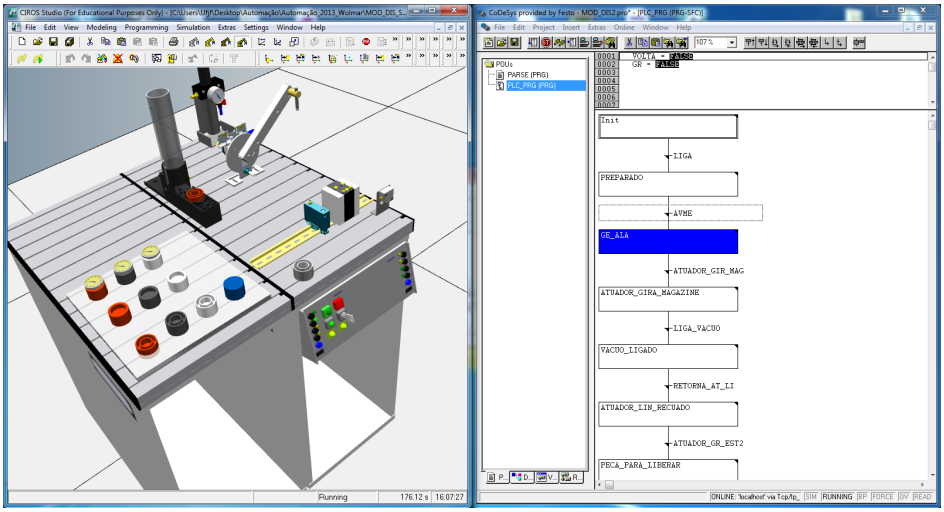


Figura 80: Módulo *Distributing* - atuador linear avançado e giratório para pró. estação

Para finalizar o processo, é necessário a volta do atuador giratório para a magazine, ligar o sopro, e recuar o atuador linear (Figura 81). Feito isso, a peça esta pronta para ir para a próxima estação e ser liberada (Figura 82).

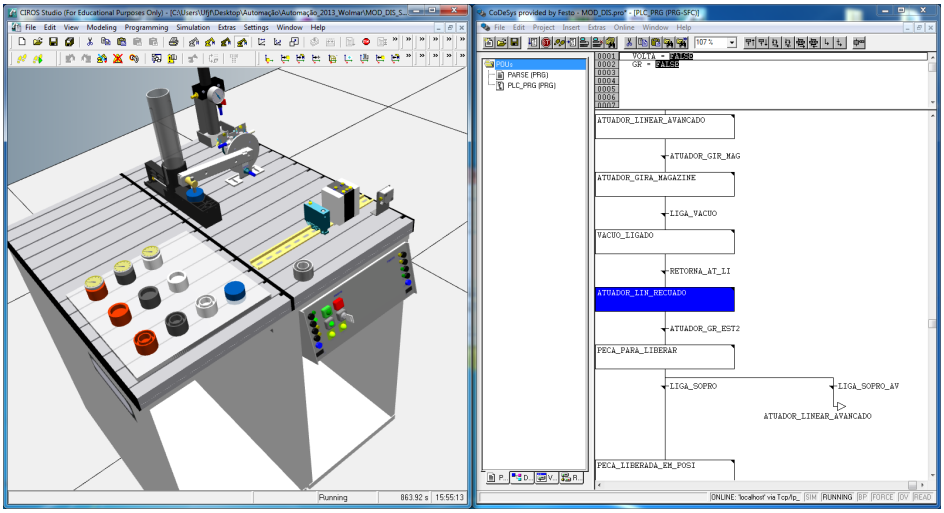


Figura 81: Módulo *Distributing* - atuador linear recuado e giratório na magazine

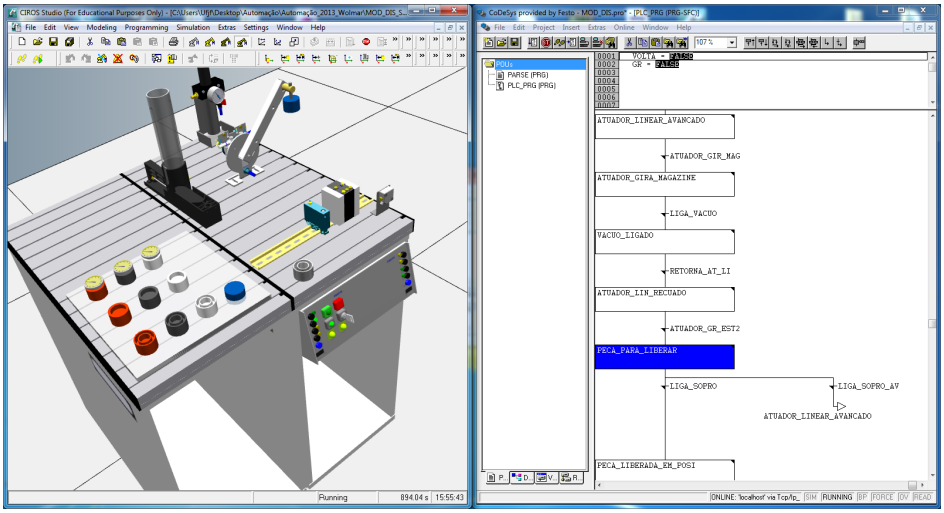


Figura 82: Módulo *Distributing* - atuador giratório com peça para pró. estação

Na Figura 82 é possível perceber que caso exista mais peças no *buffer* o módulo continua um loop de transferência de peça até terminar o estoque.

6.3 SEPARATING

O módulo *Separating* (descrita na seção 2.2.3) tem como objetivo identificar entre dois tipos de peça e desviar para a esteira correta, para duas sequências de processamento distintas. Este módulo requer atenção especial na programação durante a medição da peça, pois a profundidade da peça indica qual esteira deve transportá-la. Essa medição requer um determinado tempo, para evitar erro de identificação.

A estação é composta por duas esteiras, um atuador de parada e um desviador,

e por sensores de presença de peça na entrada da esteira 1, na saída da esteira 1, na entrada da esteira 2, e sensor de tipo de peça. O estado dos atuadores e sensores compõe o estados do sistema, descritos pela tabela 4. As transições possíveis para esse sistema são apresentadas na tabela 5.

Tabela 4: Tabela de estados - Separating

Sigla	**Significado**
DE1	Esteira 1 Desligada
LE1	Esteira 1 Ligada
DE2	Esteira 2 Desligada
LE2	Esteira 2 Ligada
APA	Atuador de Parada Avançado
APR	Atuador de Parada Recuado
ADR	Atuador Desviador Recuado
ADA	Atuador Desviador Avançado
INI	Peça no inicio da esteira 1
MEDIR	Peça no medidor de peças
PE1	Peça identificada como camisa de cilindro em transporte na esteira 1
P1	Peça camisa de cilindro pronta
PE2	Peça identificada como corpo medidor em transporte na esteira 1
TP2	Peça em transporte na esteira 2
P2	Peça corpo medidor pronta

Tabela 5: Tabela de transições - Separating

Sigla	**Significado**	e_i
L1	Liga Esteira 1	e_1
D1	Desliga Esteira 1	e_2
L2	Liga Esteira 2	e_3
D2	Desliga Esteira 2	e_4
RP	Recua Atuador de Parada	e_5
AP	Avança Atuador de Parada	e_6
AD	Avança Atuador de Desviador	e_7
RD	Recua Atuador de Desviador	e_8
TR1	Transporta Peça Camisa de Cilindro	e_9
TR2	Transporta Peça Corpo Medidor	e_{10}
DESLAP	Desliga a Esteira 1 e Avança Atuador de Parada	e_{11}
DESLRD	Desliga a Esteira 1 e Recua Atuador Desviador	e_{12}
CARREGAR	Estrada de peça no magazine	e_{13}

Descrevendo os autômatos de cada sensor e atuador usando a metodologia DeSCART se chega à representação da Figura 83.

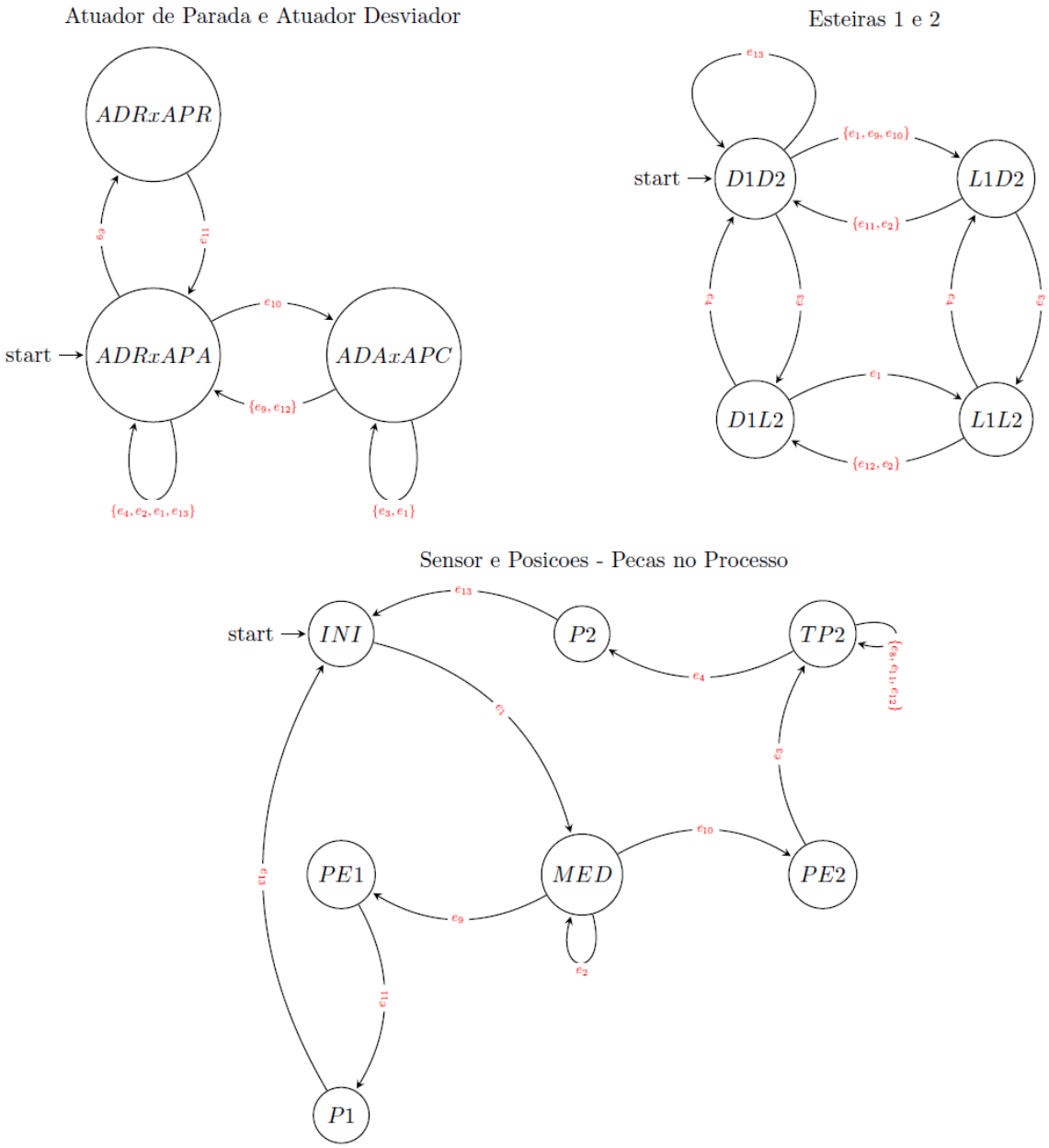


Figura 83: Autômatos simples da Estação *Separating*

Os autômatos da Figura 83 são cadastrados no aplicativo e geram o autômato global, A_G^{SEP}, conforme a Figura 84:

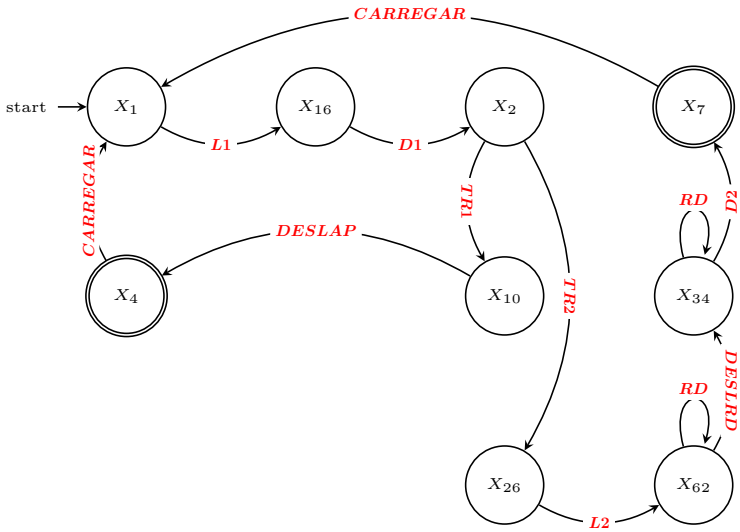


Figura 84: Diagrama de estados do autômato global - *Separating*

O código otimizado resultante apresentado pelo mecanismo de inferência é mostrado na Figura 85, cuja legenda é apresentada na tabela 6.

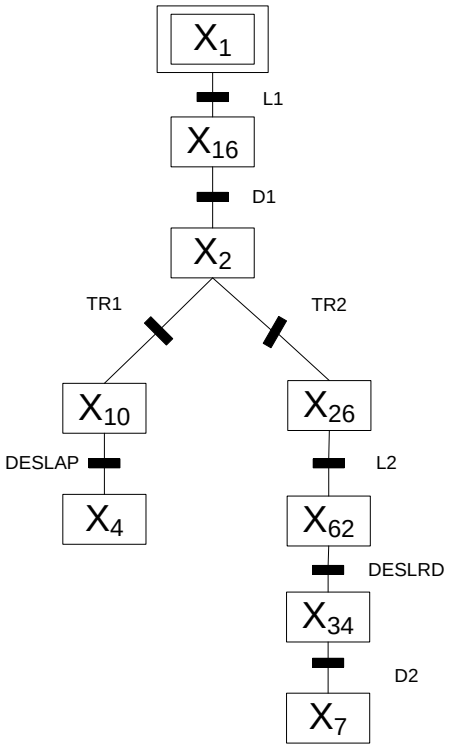


Figura 85: Código SFC resultante - *Separating*

Tabela 6: Tabela dos estados do autômato global - *Separating*

States	X_i^{SEP}
$ADRxAPAXD1D2XINI$	X_1
$ADRxAPAXL1D2XMED$	X_{16}
$ADRxAPAXD1D2XMED$	X_2
$ADRxAPAXD1D2XP1$	X_4
$ADRxAPAXD1D2XP2$	X_7
$ADRxAPRXL1D2XPE1$	X_{10}
$ADRxAPAXD1L2XTP2$	X_{34}
$ADAxAPRXL1L2XTP2$	X_{62}
$ADAxAPRXL1D2XPE2$	X_{26}

A Figura 86 apresenta o resultado do código resultante programado no software **CoDeSys V2.3** e a simulação do mesmo no **CIROS Studio** paralelamente. O programa deste módulo, assim como do anterior, é desenvolvido para começar a rodar após o usuário pressionar o botão *start* (Figura 87) e efetuar as próximas ações de acordo com o código resultante especifico de cada módulo (Figuras 88,89,90).

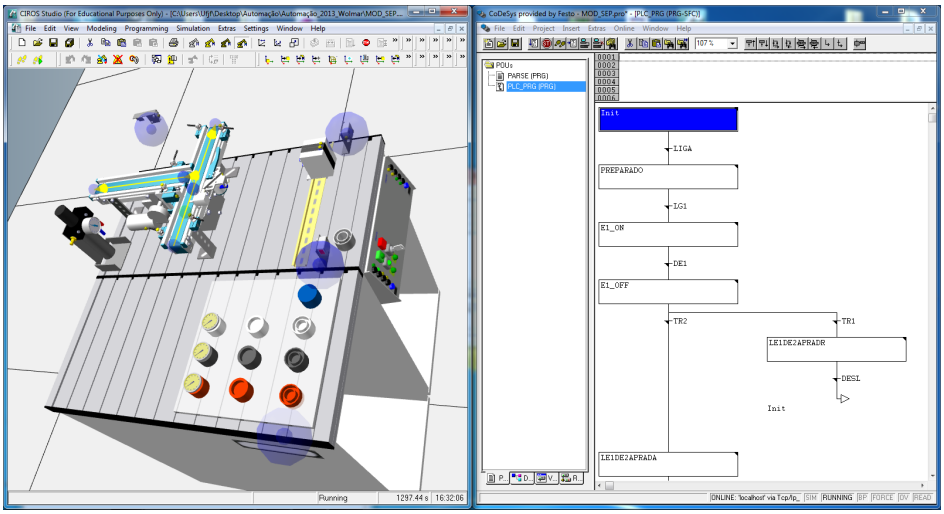


Figura 86: *Softwares* **CoDeSys V2.3** e **CIROS Studio** - iniciando a simulação do módulo *Separating*

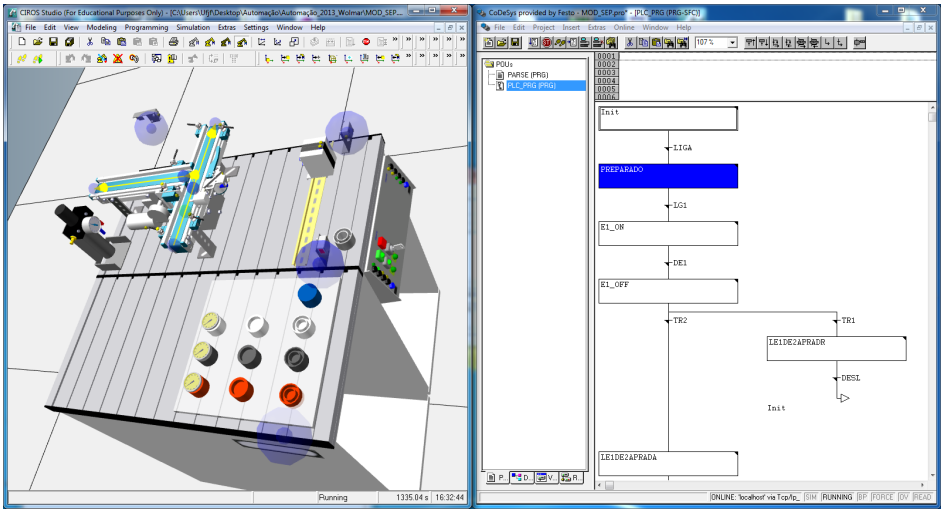


Figura 87: Módulo *Separating* preparado para receber peça

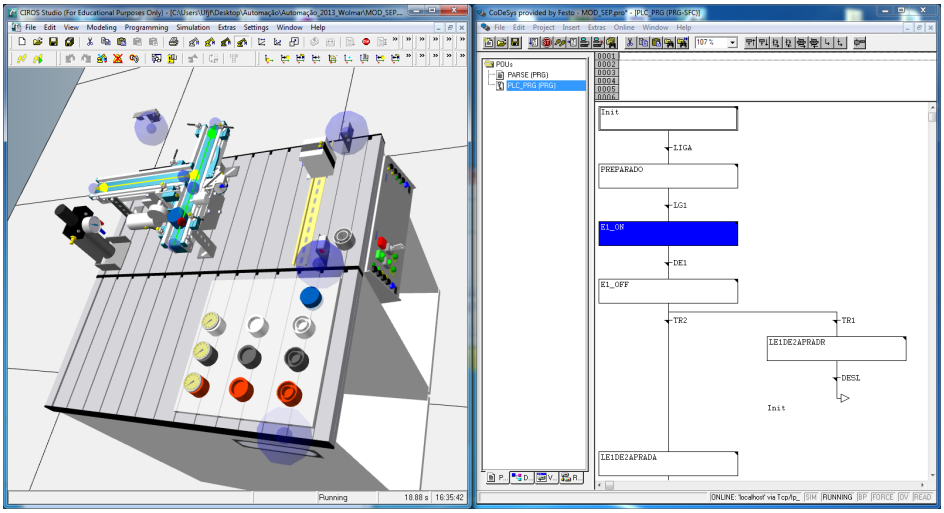


Figura 88: Módulo *Separating* com peça entrando no processo

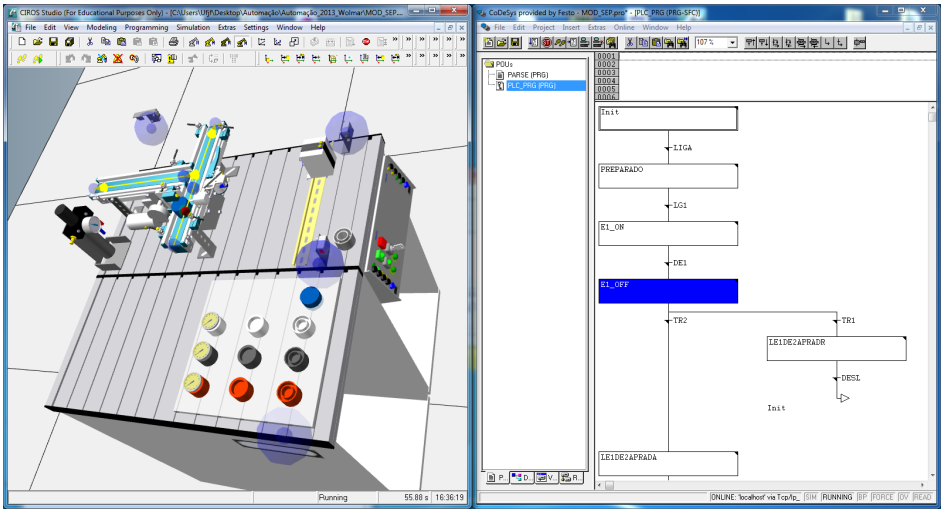


Figura 89: Módulo *Separating* verificando o tipo de peça para separação

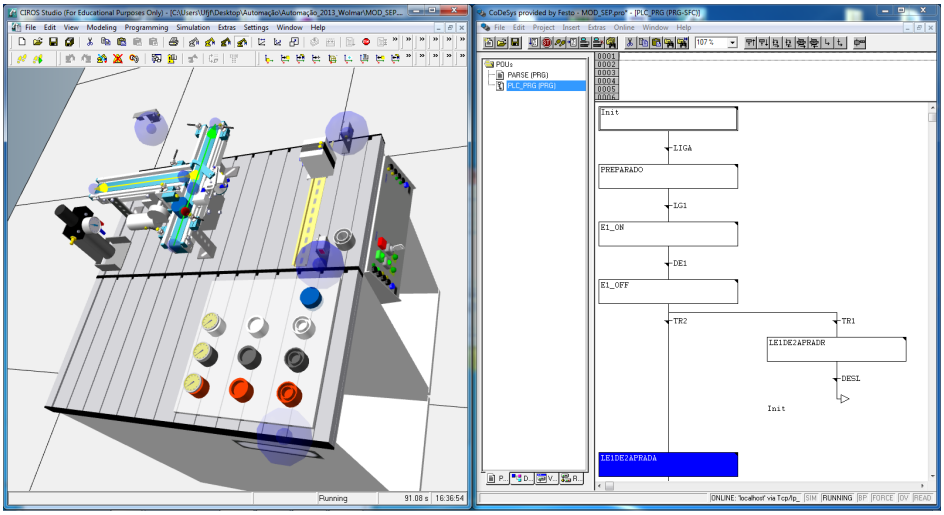


Figura 90: Módulo *Separating* liberando peça para separação

Na simulação do Módulo *Separating*, as seta indicam se as esteiras E1 e E2 (Figura 91) estão ligadas ou não (verde para ligada e amarela para desligada). Este processo também é capaz de ao verificar que a peça medida não seja uma peça para separação, liberar a mesma sem que ocorra o avanço do atuador de desvio (Figura 92).

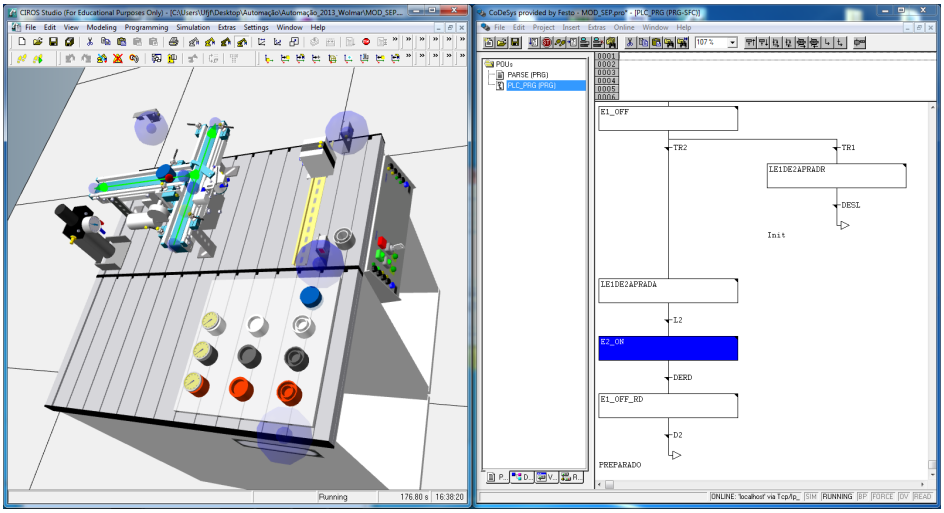


Figura 91: Módulo *Separating* com esteiras E1 e E2 ligadas

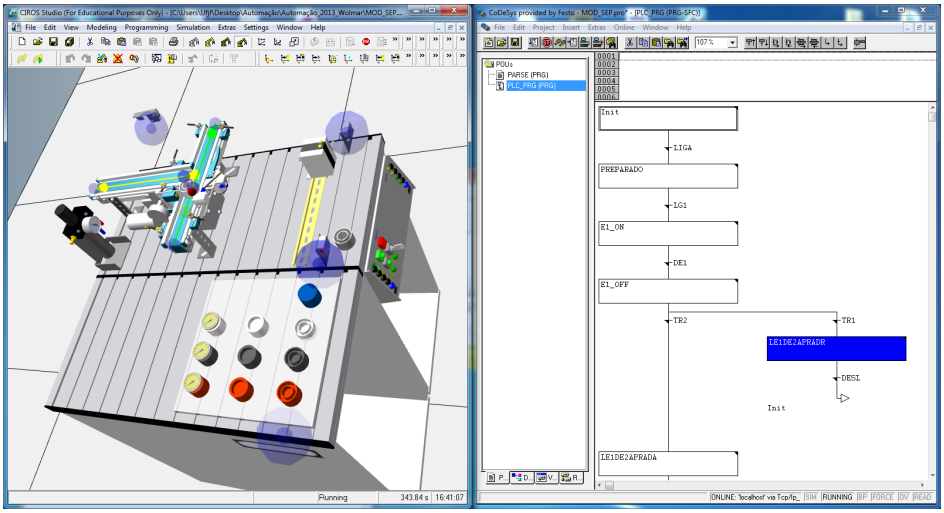


Figura 92: Módulo *Separating* liberando peça sem que ocorra separação

Ao terminar, tanto faz se a peça foi separada ou não, o sistema volta para o estado "inicial" e aguarda novas peças para separar ou não.

6.4 PICK AND PLACE

O módulo *Pick and Place* (descrita na seção 2.2.4) tem como objetivo posicionar peças sobre o corpo do medidor, para continuação de sua montagem.

A estação é composta por uma esteira transportadora, um módulo de parada elétrico, sensor óptico difuso, sensor óptico de barreira, e Módulo *Pick&Place*. O estado dos atuadores e sensores compõe o estados do sistema, descritos pela tabela 7. As transições possíveis para esse sistema são apresentadas na tabela 8.

Tabela 7: Tabela de estados - Pick and Place

Sigla	**Significado**
ESTD	Esteira Desligada
ESTL	Esteira Ligada
APR	Atuador de Parada Recuado
APA	Atuador de Parada Avançado
RCPK	Módulo Pick and Place na Posição Recuado
AVPK	Módulo Pick and Place na Posição Avançado
SBPK	Módulo Pick and Place na Posição Alto
DCPK	Módulo Pick and Place na Posição Baixo
SV	O Vácuo está Desligado
CV	O Vácuo está Ligado
INI	Peça no Inicio da Esteira
PP	Peça no Atuador de Parada
CP	Módulo Pick and Place Montando a Peça
MP	Peça pronta

Tabela 8: Tabela de transições - Pick and Place

Sigla	**Significado**	e_i
LE	Liga Esteira	e_1
DE	Desliga Esteira	e_2
AVP	Avança Atuador de Parada	e_3
RCP	Recua Atuador de Parada	e_4
FRENP	Avança o Módulo Pick and Place	e_5
ATRP	Recua o Módulo Pick and Place	e_6
DESP	Desce o Módulo Pick and Place	e_7
SOBP	Sobe o Módulo Pick and Place	e_8
LV	Liga o Vácuo	e_9
DV	Desliga o Vácuo	e_{10}
LIG	Liga a Esteira e Avança o Atuador de Parada	e_{11}
DESL	Desliga a Esteira e Recua o Atuador de Parada	e_{12}

No módulo *Pick and Place*, a esteira não pode ser ligada ou o atuador de parada não pode ser avançado quando o **Módulo Pick and Place** se encontrar avançado, pois poderia ocorrer danos ao sistema. Descrevendo os autômatos de cada sensor e atuador usando a metodologia DeSCART se chega à representação da Figura 93.

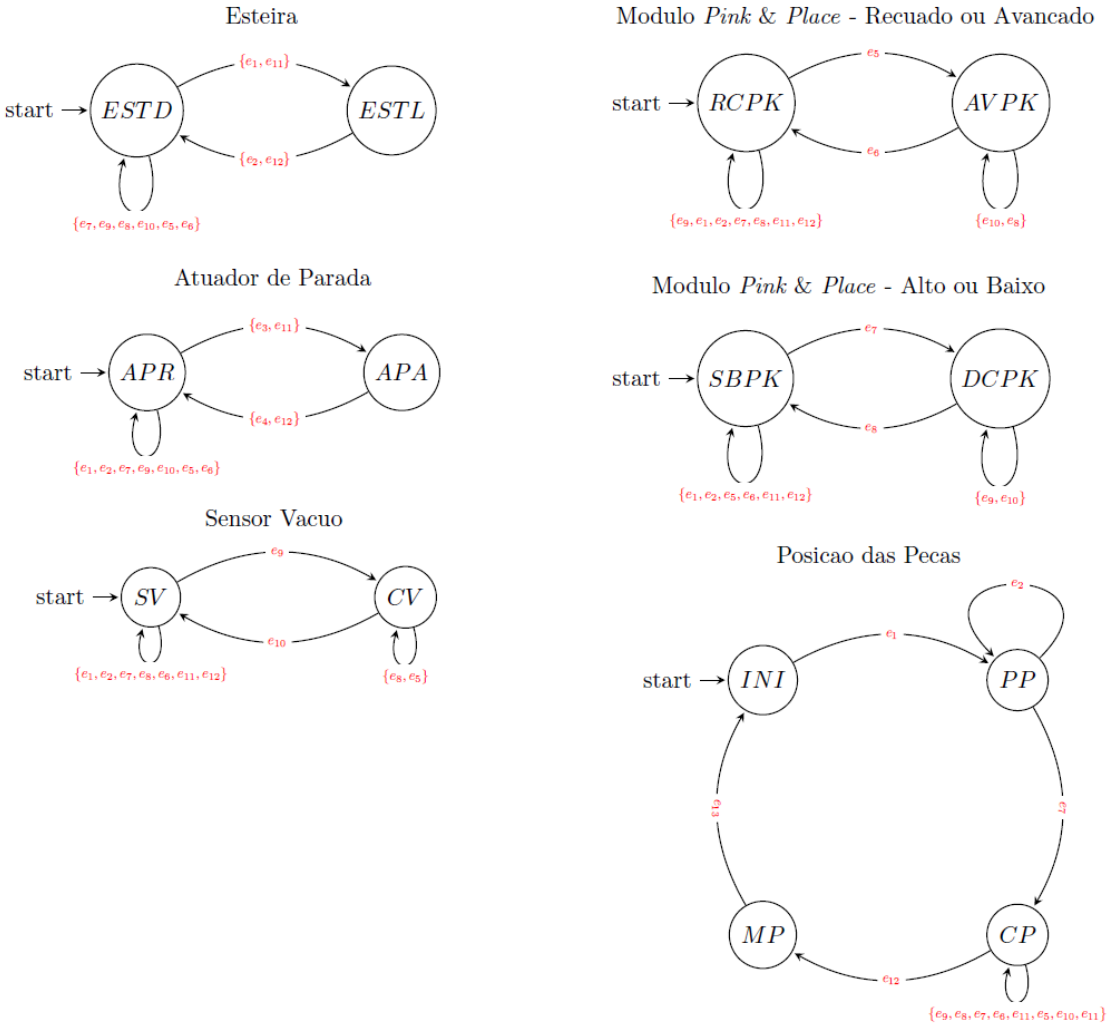


Figura 93: Autômatos simples da Estação *Pick and Place*

Os autômatos da Figura 93 são cadastrados no aplicativo e geram o autômato global, A_G^{PIC}, conforme a Figura 94:

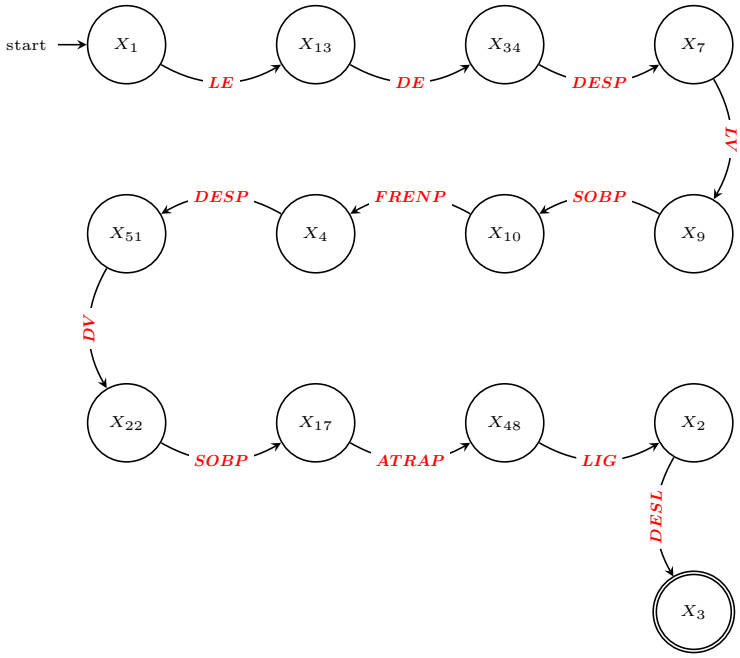


Figura 94: Diagrama de estados do autômato global - *Pick and Place*

O código otimizado resultante apresentado pelo mecanismo de inferência é mostrado na Figura 95, cuja legenda é apresentada na tabela 6.

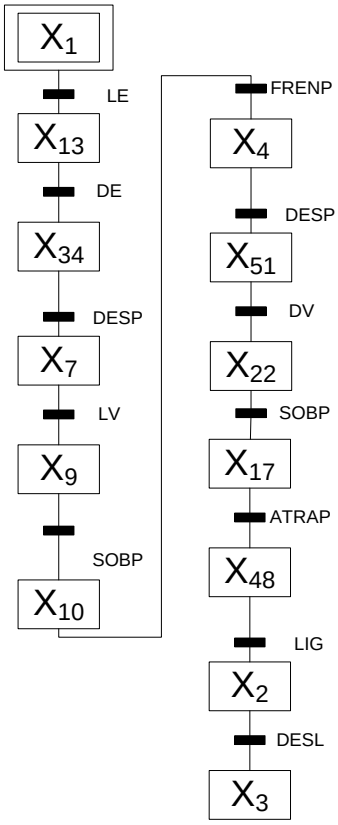


Figura 95: Código SFC resultante - *Pick and Place*

Tabela 9: Tabela dos estados do autômato global - *Pick and Place*

States	X_i^{PIC}
$ESTDxAPRxRCPKxSBPKxSVxINI$	X_1
$ESTLxAPRxRCPKxSBPKxSVxPP$	X_{13}
$ESTDxAPRxRCPKxSBPKxSVxPP$	X_{34}
$ESTDxAPRxRCPKxDCPKxSVxCP$	X_7
$ESTDxAPRxRCPKxSBPKxSVxINI$	X_1
$ESTDxAPRxRCPKxDCPKxCVxCP$	X_9
$ESTDxAPRxRCPKxSBPKxCVxCP$	X_{10}
$ESTDxAPRxAVPKxSBPKxCVxCP$	X_4
$ESTDxAPRxAVPKxDCPKxCVxCP$	X_{51}
$ESTDxAPRxAVPKxDCPKxSVxCP$	X_{22}
$ESTDxAPRxAVPKxSBPKxSVxCP$	X_{17}
$ESTDxAPRxRCPKxSBPKxSVxCP$	X_{48}
$ESTLxAPRxRCPKxSBPKxSVxCP$	X_2
$ESTDxAPRxRCPKxSBPKxSVxMP$	X_3

De modo a não ficar massante a apresentação de todas as simulações, os próximos resultados são apresentados de uma forma alternativa, buscando apresentar outros detalhes dos módulos, plataformas de simulação e desenvolvimento. O Módulo *Pick and Place* trabalha com um tipo de peça para "corpo de medidor" (Figura 96).

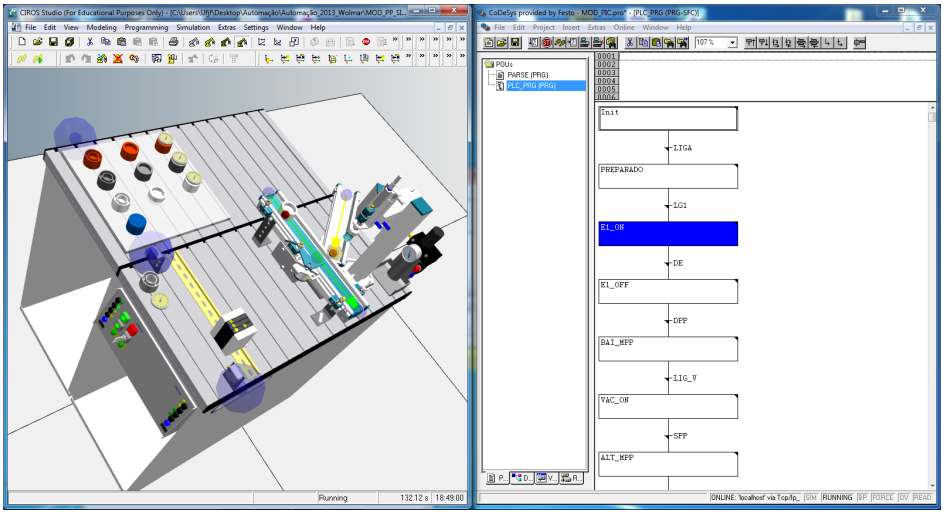


Figura 96: Módulo *Pick and Place* encaminhando peça para processo

Uma característica interessante do *software* **CIROS Studio** é indicar a posição

das peças para o usuário durante a simulação. Na Figura 97 tanto a peça corpo de medidor, quanto a peça medidor são referenciadas por uma "bola" vermelha que indica a posição das peças em suas respectivas esteiras. Ao ligar o vácuo, a peça medidor perde a indicação da esfera vermelha para apontar que a mesma já não se encontra posicionada sobra a esteira (Figura 98).

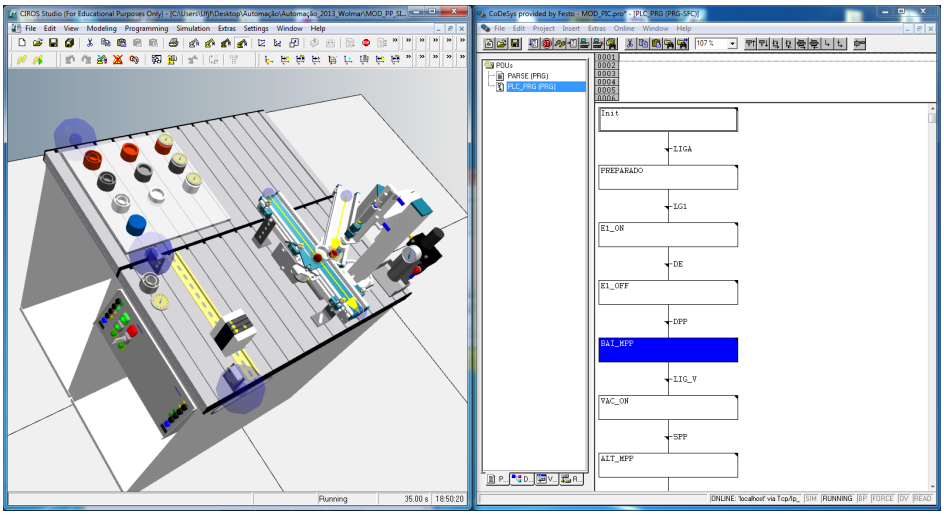


Figura 97: Módulo *Pick and Place* esteira desligada e módulo *Pick&Place* recuado na posição inferior

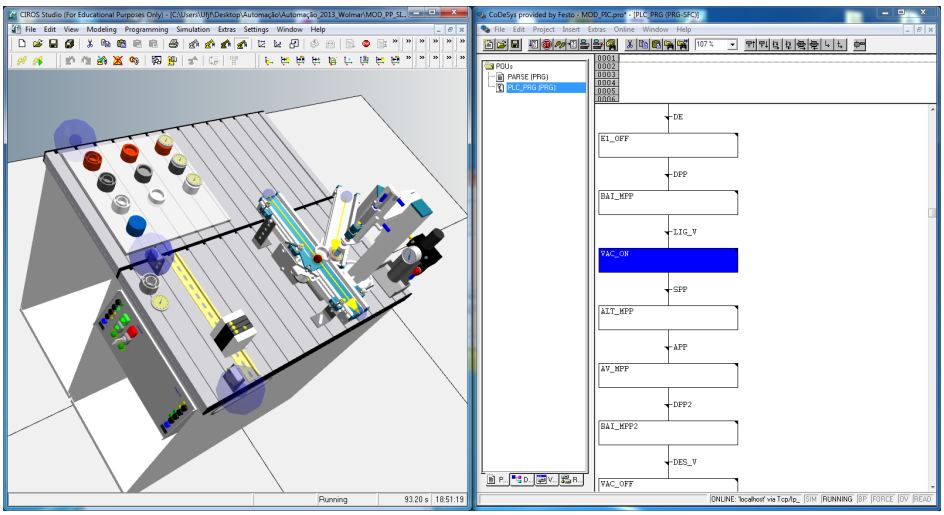


Figura 98: Módulo *Pick and Place* esteira desligada e módulo *Pick&Place* recuado na posição inferior com vácuo ligado

Outras indicações podem ser vistas em todos os atuadores dos módulos simulados. Por exemplo, as luzes azuis do atuador *Pick&Place* indicão quando ele está na parte superior e com vácuo ligado (Figura 99).

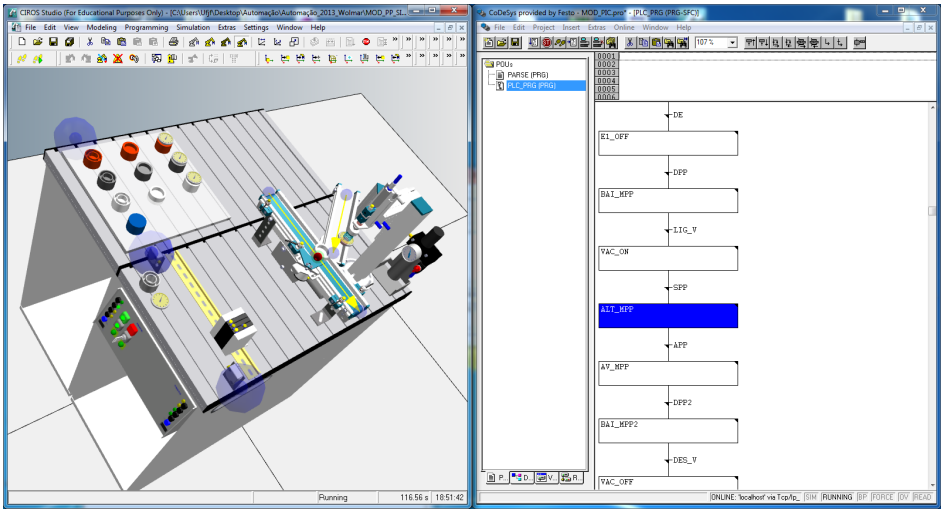


Figura 99: Módulo *Pick and Place* esteira desligada e módulo *Pick&Place* recuado na posição superior com vácuo ligado

6.5 FLUIDIC MUSCLE PRESS

A estação *Fluidic Muscle Press* (descrita na seção 2.2.5) tem como objetivo pressionar o medidor no cilindro do corpo de medidor, concluindo a montagem do medidor.

A estação é composta por um módulo linear rotativo, sensor difuso, e um módulo *fluidic muscle press*. O estado dos atuadores e sensores compõe o estados do sistema, descritos pela tabela 10. As transições possíveis para esse sistema são apresentadas na tabela 11.

Tabela 10: Tabela de estados - Fluidic Muscle Press

Sigla	**Significado**
GA	Garra Aberta
GF	Garra Fechada
MLEA	Módulo Linear na Estação Anterior
MLPE	Módulo Linear na Próxima Estação
MRA	Módulo Rotativo na Estação Anterior
MRP	Módulo Rotativo na Estação de Processamento
MRE	Módulo Rotativo na Próxima Estação
CP	Prensa Acionada
SP	Prensa Desligada
INI	Peça no Inicio da Estação
PRENSA	Peça na Prensa
PP	Peça Pronta em Fase de Transporte Para a Próxima Estação
PPE	Peça pronta no Final da Estação

Tabela 11: Tabela de transições - Fluidic Muscle Press

Sigla	**Significado**	e_i
FG	Fecha a Garra	e_1
AG	Abre a Garra	e_2
AVL	Avança o Módulo Linear para a Próxima Estação	e_3
RCL	Recua o Módulo Linear para a Estação Anterior	e_4
AP	Move o Módulo Rotativo da Estação Anterior para a Estação de Processamento	e_5
PE	Move o Módulo Rotativo da Estação de Processamento para a Próxima Estação	e_6
PA	Move o Módulo Rotativo da Estação de Processamento para a Estação Anterior	e_7
EP	Move o Módulo Rotativo da Próxima Estação para a Estação de Processamento	e_8
LIGP	Liga a Prensa	e_9
DESLP	Desliga a Prensa	e_{10}

A fim de não danificar esta estação, é preciso garantir que o atuador linear não se mova quando o módulo rotativo estiver na posição de processamento, e que o módulo rotativo também não se mova a partir do momento em que a prensa estiver atuando sobre a peça. Descrevendo os autômatos de cada sensor e atuador usando a metodologia DeSCART se chega à representação da Figura 100.

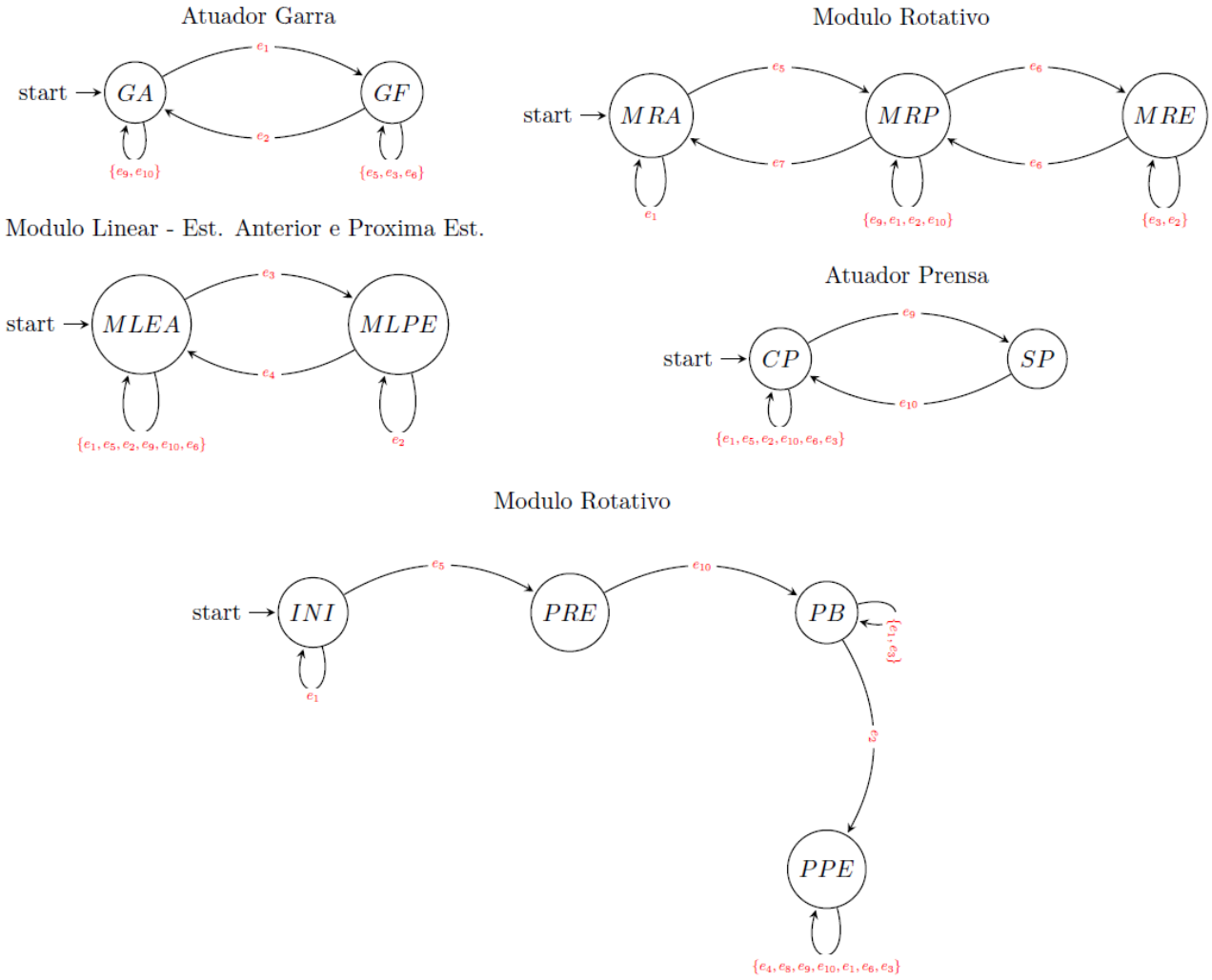


Figura 100: Autômatos simples da Estação *Fluidic Muscle Press*

Os autômatos da Figura 100 são cadastrados no aplicativo e geram o autômato global, A_G^{FMP}, conforme a Figura 101:

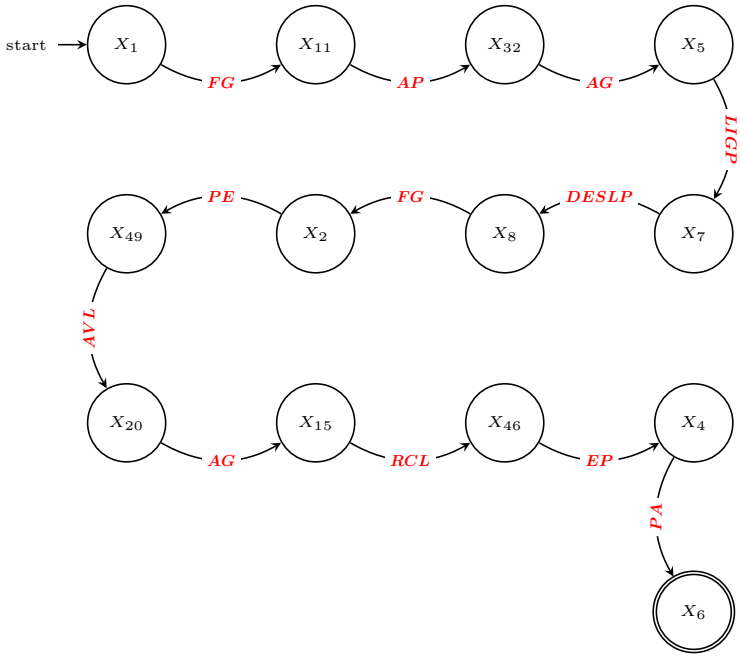


Figura 101: Diagrama de estados do autômato global - *Fluidic Muscle Press*

O código otimizado resultante apresentado pelo mecanismo de inferência é mostrado na Figura 102, cuja legenda é apresentada na tabela 12.

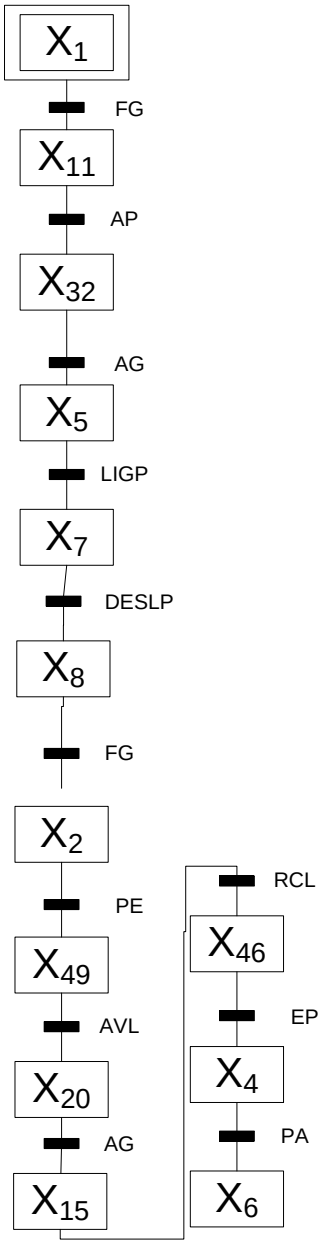


Figura 102: Código SFC resultante - *Fluidic Muscle Press*

Tabela 12: Tabela dos estados do autômato global - *Fluidic Muscle Press*

States	X_i^{FMP}
$GAxMLEAxMRAxCPxINI$	X_1
$GFxMLEAxMRAxCPxINI$	X_{11}
$GFxMLEAxMRPxCPxPREN$	X_{32}
$GAxMLEAxMRPxCPxPREN$	X_5
$GAxMLEAxMRPxSPxPREN$	X_7
$GAxMLEAxMRPxCPxPB$	X_8
$GFxMLEAxMRPxCPxPB$	X_2
$GFxMLEAxMRExCPxPB$	X_{49}
$GFxMLPExMRExCPxPB$	X_{20}
$GAxMLPExMRExCPxPPE$	X_{15}
$GAxMLEAxMRExCPxPPE$	X_{46}
$GAxMLEAxMRPxCPxPPE$	X_4
$GAxMLEAxMRAxCPxPPE$	X_6

Apesar do código resultante ser trabalhado e programado no *software* **CoDeSys V2.3** na forma *structure function charge*, os sensores a atuadores são definidos na forma de texto estruturado (Structured Text - ST) e são definidos como variáveis booleanas (Figura 103). Para cada módulo existe uma tabela auxiliar responsável por indicar qual entrada/saída pertence a cada atuador/sensor (Figura 104).

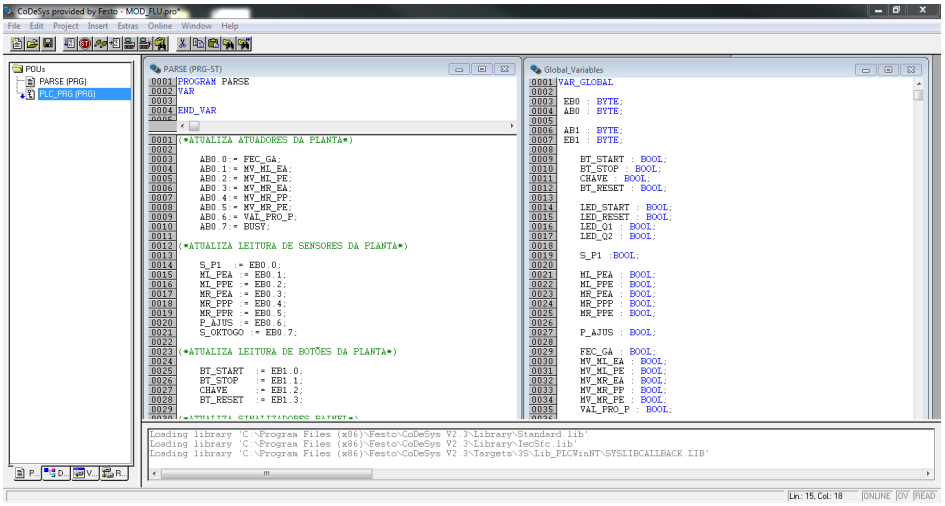


Figura 103: *Software* **CoDeSys V2.3** - Variáveis Globais e PARSE do Módulo *Fluidic Muscle Press*

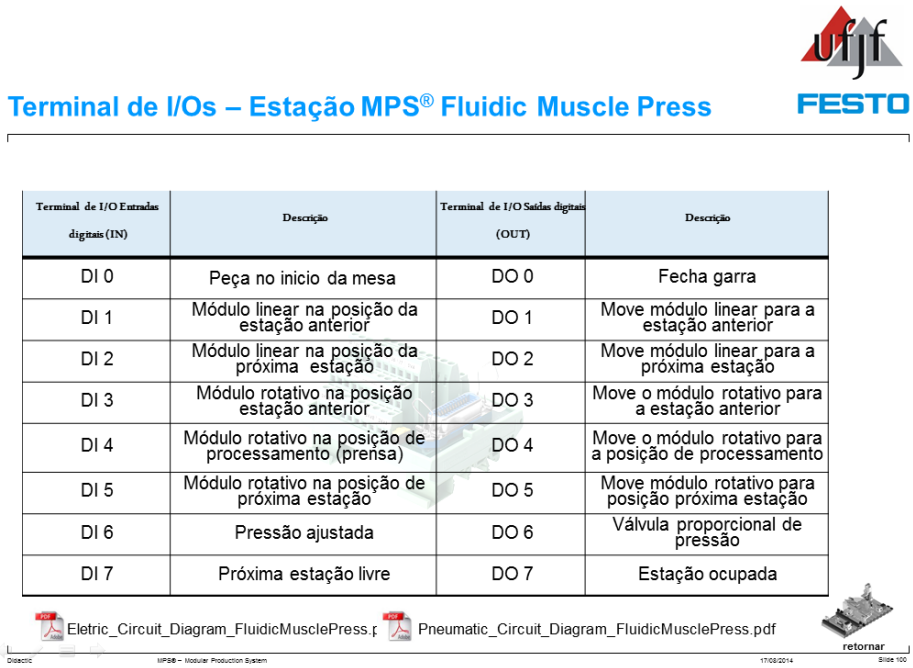


Figura 104: Tabela indicativa do terminal de entrada e saída do Módulo *Fluidic Muscle Press* (POLA, 2013)

Cada estado e ação representados em SFC são na verdade um código implementado em ST (Figura 105), indicando exatamente os valores encontrados para cada estado gerado no código resultante do mecanismo de inferência.

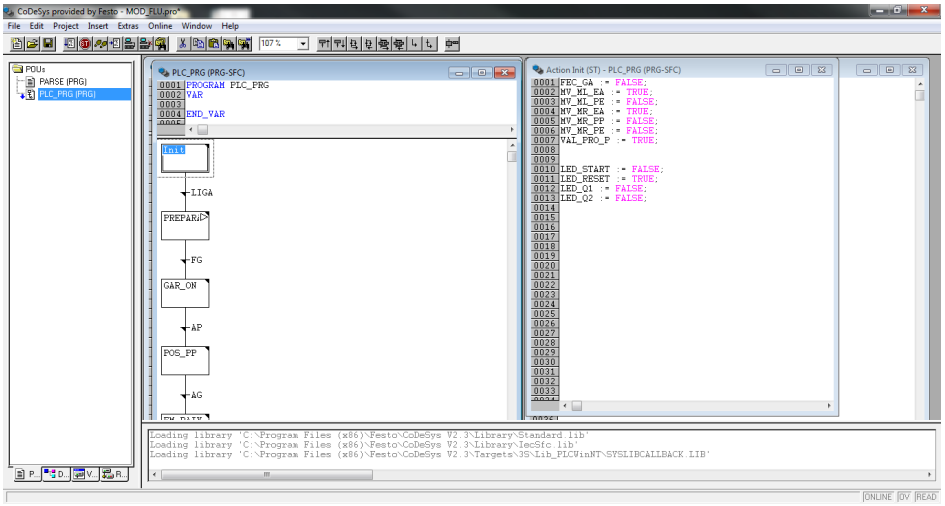


Figura 105: *Software* **CoDeSys V2.3** - Código SFC e ST

6.6 SORTING

O objetivo da estação *Sorting* (descrita na seção 2.2.2) é separar as peças. A estação é capaz de separar três tipos de peça e as classificar como metálica, escura ou nenhuma das duas opções anteriores.

A estação é composta por dois sensores ópticos difusos, um sensor indutivo, um sensor retro reflexivo, atuador de parada, esteira transportadora, e dois módulos desviador. O estado dos atuadores e sensores compõe o estados do sistema, descritos pela tabela 13. As transições possíveis para esse sistema são apresentadas na tabela 14.

Tabela 13: Tabela de estados - Sorting

Sigla	**Significado**
ESTD	Esteira Desligada
ESTL	Esteira Ligada
APA	Atuador de Parada Avançado
APR	Atuador de Parada Recuado
AT1RC	Atuador 1 Recuado
AT1AV	Atuador 1 Avançado
AT2RC	Atuador 2 Recuado
AT2AV	Atuador 2 Avançado
INI	Peça no inicio da esteira
PC1	Peça Detectada é Metálica
P1P	Peça Metálica Separada
PC2	Peça Detectada é Não-Metálica e Escura
P2P	Não-Metálica e Escura Metálica Separada
PC3	Peça Detectada é Não-Metálica e Não-Escura
P3P	Não-Metálica e Não-Escura Metálica Separada

Tabela 14: Tabela de transições - Sorting

Sigla	**Significado**	e_i
LE	Liga Esteira	e_1
DE	Desliga Esteira	e_2
RCAP	Recua Atuador de Parada	e_3
AVAP	Avança Atuador de Parada	e_4
AVA1	Avança Atuador 1	e_5
RCA1	Recua Atuador 1	e_6
AVA2	Avança Atuador 2	e_7
RCA2	Recua Atuador 2	e_8
TR	Transporta Peça, Liga a Esteira e Recua Atuador de Parada	e_9
DESL	Desliga a Esteira e Avança Atuador de Parada	e_{10}

A fim de não danificar esta estação, é preciso garantir que o sensor o qual indica se a rampa está cheia ou não seja capaz de parar a atividade da estação caso informe

que a rampa chegou ao seu limite de peças. Descrevendo os autômatos de cada sensor e atuador usando a metodologia DeSCART se chega à representação da Figura 106.

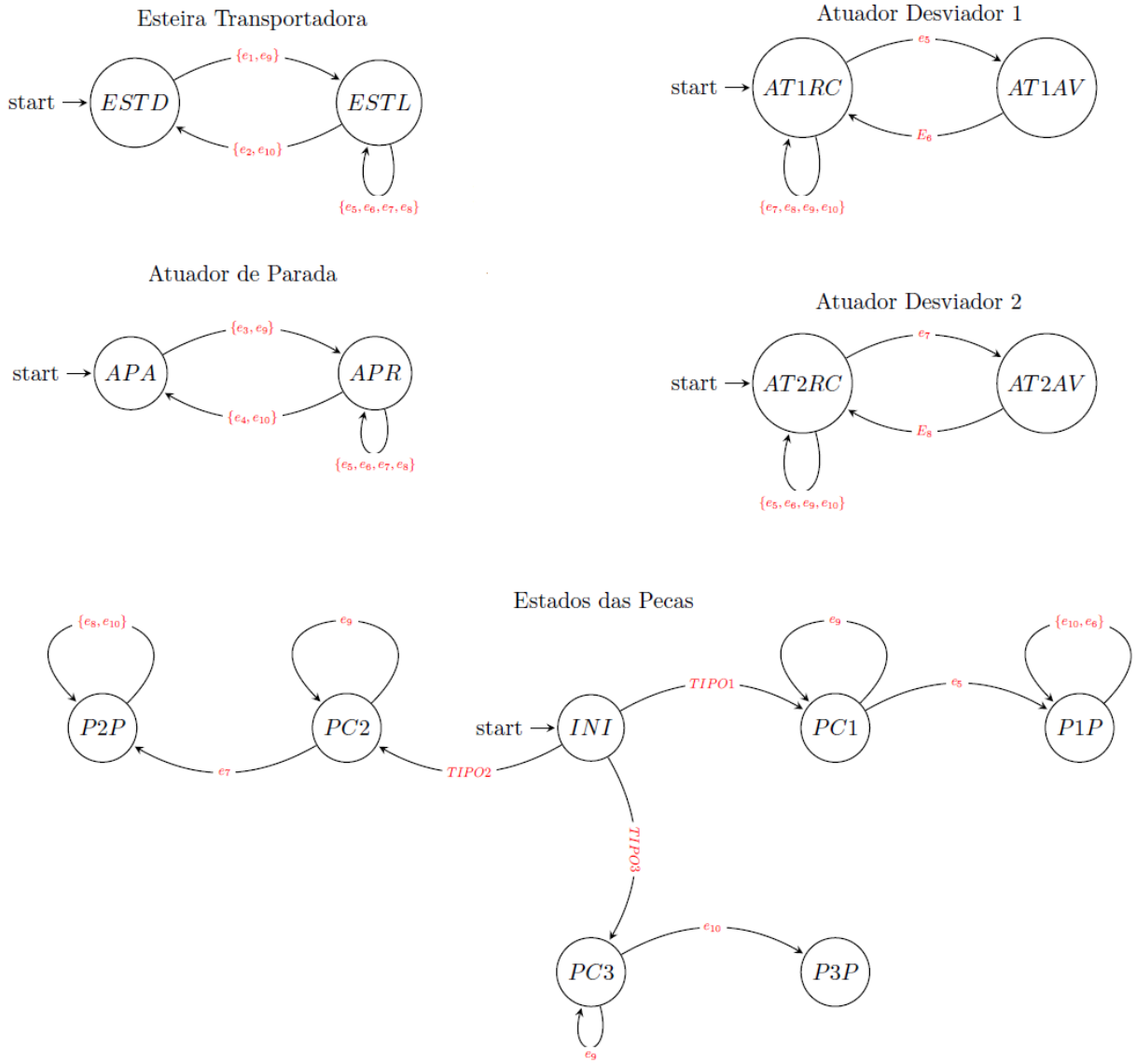


Figura 106: Autômatos simples da Estação *Sorting*

Os autômatos da Figura 106 são cadastrados no aplicativo e geram o autômato global, A_G^{FMP}, conforme a Figura 107:

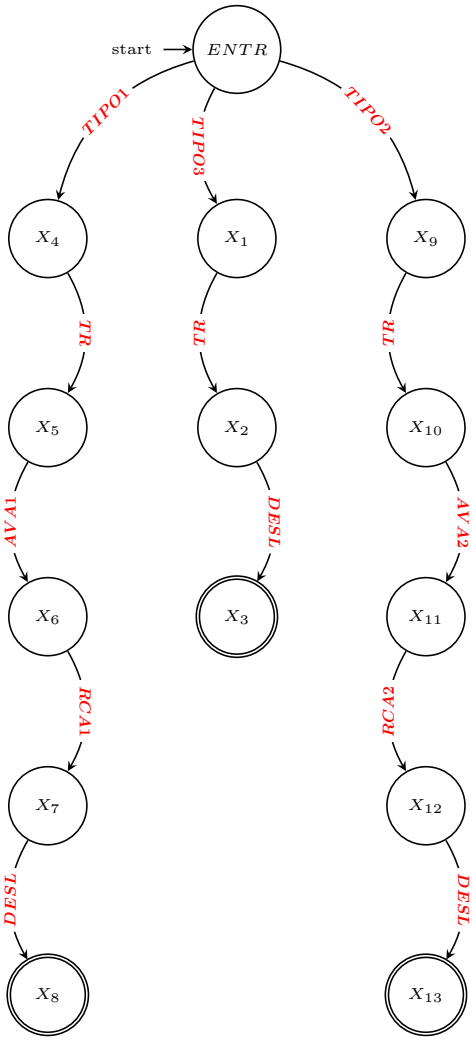


Figura 107: Diagrama de estados do autômato global - *Sorting*

O código otimizado resultante apresentado pelo mecanismo de inferência é mostrado na Figura 108, cuja legenda é apresentada na tabela 15.

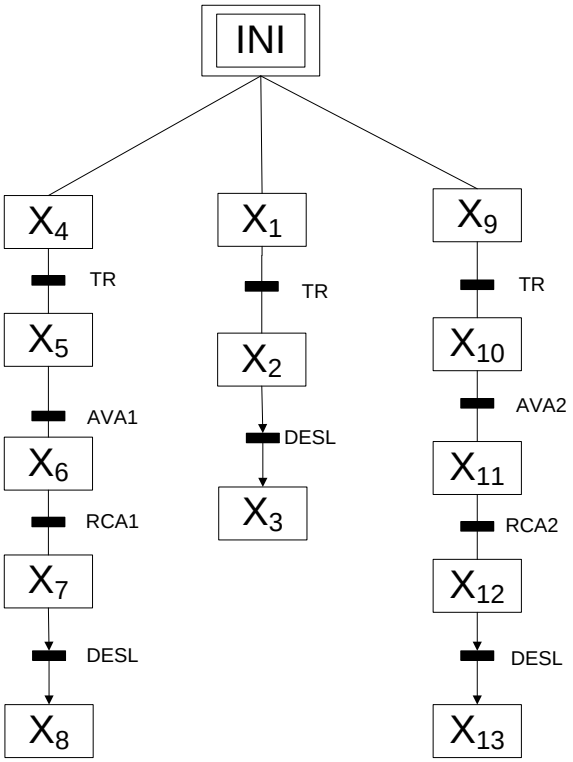


Figura 108: Código SFC resultante - *Sorting*

Tabela 15: Tabela dos estados do autômato global - *Sorting*

States	X_i^{SOR}
$ESTDxAPAxAT1RCxAT2RCxPC3$	X_1
$ESTLxAPRxAT1RCxAT2RCxPC3$	X_2
$ESTDxAPAxAT1RCxAT2RCxP3P$	X_3
$ESTDxAPAxAT1RCxAT2RCxPC1$	X_4
$ESTLxAPRxAT1RCxAT2RCxPC1$	X_5
$ESTLxAPRxAT1AVxAT2RCxP1P$	X_6
$ESTLxAPRxAT1RCxAT2RCxP1P$	X_7
$ESTDxAPAxAT1RCxAT2RCxP1P$	X_8
$ESTDxAPAxAT1RCxAT2RCxPC2$	X_9
$ESTLxAPRxAT1RCxAT2RCxPC2$	X_{10}
$ESTLxAPRxAT1RCxAT2AVxP2P$	X_{11}
$ESTLxAPRxAT1RCxAT2RECxP2P$	X_{12}
$ESTDxAPAxAT1RCxAT2RECxP2P$	X_{13}

Ao utilizar a linguagem SFC, a programação torna-se intuitiva. Auxiliando na modelagem dos sistemas aproximando autômatos, simulação e programação do CLP visualmente (Figura 109). Tornando o ato de programar mais intuitivo e seguro para os atuadores do sistema.

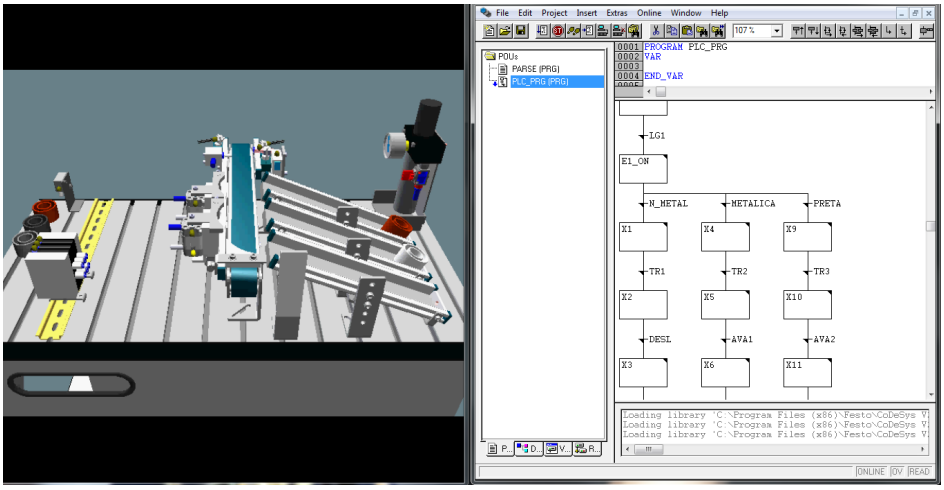


Figura 109: Módulo *Sorting* em simulação - código referente no *software* **CoDeSys V2.3**

7 CONCLUSÃO E TRABALHOS FUTUROS

7.1 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um método para simplificar a tarefa de programação de controladores lógicos programáveis, que visasse a garantia de segurança dos componentes do sistema, bem como minimizasse o número de operações que levassem o estado inicial do sistema até o seu estado objetivo.

Para o desenvolvimento dessa metodologia foram utilizados os conceitos de Autômatos e Redes de Petri para a modelagem do sistema. Os Capítulos 3 e 4 apresentam todo o arcabouço matemático e teórico para utilização destes conceitos, bem como boas fontes bibliográficas, caso o leitor deseje um maior aprofundamento.

Os Autômatos foram usados para representação de pequenas porções dos sistemas, por serem as estruturas mais simples e intuitivas do estudo de Sistemas a Eventos Discretos. As Redes de Petri foram usadas subsequentemente para análise do sistema global, composto por todos os subsistemas. Seu uso pode ser justificado por ser uma estrutura bem equipada e com recursos necessários para lidar com características fundamentais de controle de sistemas industriais. A Rede de Petri pode ser explorada como um grafo de estados orientado, o que se mostrou bem útil na busca por uma solução ótima.

A Rede de Petri foi representada na forma de um sistema de regras, e um mecanismo de inferência realizou a busca pelo caminho ótimo entre o estado inicial e objetivo, com relação ao número de transições. Este mecanismo foi desenvolvido em linguagem de programação lógica (PROLOG), que é muita adequada a aplicações de lógica e computação simbólica. A ferramenta utilizada para essa linguagem foi o **Visual Prolog**, em usa licença acadêmica. Uma das limitações encontradas na utilização deste *software* foi a restrição no número de argumentos permitidos na execução dos códigos. Isso obrigou a busca de simplificações ainda mais severas dos modelos propostos.

Um aplicativo foi criado como ferramenta de aplicação para a metodologia, e permi-

tiu a elaboração de códigos de programação de CLP em linguagem similar a SFC. Estes códigos foram a base da construção dos códigos na plataforma *CoDeSys*, e subsequentemente validados em simulação no ambiente *CIROS Studio*. Todos os códigos criados obtiveram bom desempenho na realização da tarefa modelada, bem como garantiram a integridade os atuadores.

7.2 TRABALHOS FUTUROS

A continuação natural deste trabalho é a implementação de uma interface mais amigável para o usuário, a fim de que a composição dos autômatos seja realizada de uma forma mais dinâmica; Algo próximo a ferramenta desenvolvida em (HAMADA, 2007), onde o autor desenvolveu um simulador de máquinas de estados finitos.O objetivo seria aproximar o programa desenvolvido a programas comerciais, por exemplo **Visual Object Net++** (DRATH, 2002).

Para a montagem dá árvore de estados, propõe-se buscar novas técnicas de otimização (Programação linear, Programação por Restrições, Algoritmos Híbridos, Métodos Heurísticos, Algoritmos Aproximados) que apresentem um caminho ótimo a ser seguido dentro do universo de caminhos possíveis existentes. A avaliação do tempo de execução de cada ação pode ser incluída, para tornar a análise de otimalidade mais completa.

Devido à restrição do número de argumentos aceitos pela *Visual Prolog* em usa versão acadêmica, se propõe também o desenvolvimento de novos mecanismos de inferência que não possuam restrições similares, e que não tenham vínculos comerciais.

REFERÊNCIAS

BACK, R. T. L. Z. D. B. L. Identificação de gargalos em uma indústria de mangueiras com auxílio da teoria das restrições. *III CONGRESSO BRASILEIRO DE ENGENHARIA DE PRODUÇÃO*, p. 9, 2013.

BOGDAN FRANK L. LEWIS, Z. K. S.; JR., J. M. *Manufacturing Systems Control Design : a matrix-based approach. - (Advances in industrial control)*. [S.l.]: Springer Science, 2006.

BRATKO, I. *Prolog programming for artificial intelligence*. [S.l.]: Pearson education, 2001.

CARDOSO, R. V. J. *Redes de Petri*. [S.l.]: UFSC, 1997.

CARVALHO, M.; FILHO, O. S.; FERNANDES, C. O planejamento da manufatura–práticas industriais e métodos de otimização. *Gestão & Produção*, SciELO Brasil, v. 5, n. 1, p. 34–59, 1998.

CASSANDRAS, S. L. C. G. *Introduction to Discrete Event systems*. [S.l.]: Springer Science, 2008.

CHORLEY, A.; BENCH-CAPON, T. Agatha: Automation of the construction of theories in case law domains. In: *Proceedings of Jurix*. [S.l.: s.n.], 2004. p. 89–98.

CHOSET, H. M. *Principles of robot motion: theory, algorithms, and implementation*. [S.l.]: MIT press, 2005.

CLOCKSIN, W. F.; MELLISH, C. S.; CLOCKSIN, W. *Programming in PROLOG*. [S.l.]: Springer, 1987.

COLMERAUER, A.; ROUSSEL, P. The birth of prolog. In: ACM. *History of programming languages—II*. [S.l.], 1996. p. 331–367.

COMMONER, F. et al. Marked directed graphs. *Journal of Computer and System Sciences*, Elsevier, v. 5, n. 5, p. 511–523, 1971.

DESCARTES, R.; WEISSMAN, D.; BLUHM, W. T. *Discourse on the Method: And, Meditations on First Philosophy*. [S.l.]: Yale University Press, 1996.

DIDACTIC, F. *The current range of Festo Didactic products*. [S.l.]: Festo Didactic GmbH & Co. KG., 2013.

DRATH, R. *Visual Object Net, Version 2.7a*. 2002. Disponível em: <www.r-drath.de/Home/VisualObjectNet.html>.

GOEKING, W. *Da máquina a vapor aos softwares de automação*. Maio 2010. Disponível em: <www.osetoreletrico.com.br/web/component/content/article/57-artigos-e-materias/343-xxxx.html>.

HAMADA, M. Web-based active e-learning tools for automata theory. In: SPECTOR, J. M. et al. (Ed.). *ICALT*. [S.l.]: IEEE Computer Society, 2007. p. 877–879. ISBN 978-0-7695-2916-5.

ITO YUJI KOBAYASHI, K. S. M. *Automata, Formal Languages and Algebraic Systems*. [S.l.]: World Scientific Publishing Company (August 31, 2010), 2010.

JIANG, S.; KUMAR, R. Supervisory control of discrete event systems with ctl* temporal logic specifications. *SIAM Journal on Control and Optimization*, SIAM, v. 44, n. 6, p. 2079–2103, 2006.

JOHN, K.-H.; TIEGELKAMP, M. *IEC 61131-3 : programming industrial automation systems : concepts and programming languages, requirements for programming systems, aids to decision-making tools"*. Berlin, Heidelberg: Springer-Verlag, 2001. Disponível em: <http://opac.inria.fr/record=b1130044>.

KOSMATOPOULOS, E.; CHRISTODOULOU, M. A state space model for modeling and control of manufacturing processes. In: IEEE. *Emerging Technologies and Factory Automation, 1995. ETFA'95, Proceedings., 1995 INRIA/IEEE Symposium on*. [S.l.], 1995. v. 2, p. 563–571.

KUMAR, R.; GARG, V. K. Modeling and control of logical discrete event systems. Kluwer, 1995.

LABBÉ, S.; LAPITRE, A. Carver: a slicing tool for communicating automata specifications. In: IEEE. *Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium on*. [S.l.], 2006. p. 99–102.

LAI DAVIDE NESSI, M. P. C. A. G. C. S. S. Comparison between two diagnostic tools based on automata and petri nets. *Proceedings of the 9th International Workshop on Discrete Event Systems*, 2008.

LAI, S. et al. A comparison between two diagnostic tools based on automata and petri nets. In: IEEE. *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*. [S.l.], 2008. p. 144–149.

LAVALLE, S. M. *Planning algorithms*. [S.l.]: Cambridge university press, 2006.

LU, B.; LIU, Z. Prolog with best first search. In: IEEE. *Control and Decision Conference (CCDC), 2013 25th Chinese*. [S.l.], 2013. p. 917–922.

MARTINEZ, J.; MURO, P.; SILVA, M. Modeling, validation and software implementation of production systems using high level petri nets. In: IEEE. *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*. [S.l.], 1987. v. 4, p. 1180–1185.

MATA, L. C. da. *Curso de Especialização Técnica em Manutenção e Programação PLC Controllogix*. [S.l.: s.n.], 2010.

MORAIS, P. d. L. C. Cicero Couto de. *Engenharia de Automacao Industrial*. [S.l.]: LTC, 2013.

PETERSON, J. L. *Petri Net Theory and the Modeling of Systems*. [S.l.]: Prentice Hall, 1981.

PETRI, C. A. *Kommunikation mit Automaten*. Dissertação (Mestrado) — Universidade de Bonn, 1962.

POLA, D. F. S. R. *Treinamento MPS UFJF*. [S.l.]: Festo Didactic & GmbH Co. KG., 2013.

QUEIROZ, M. H. D.; CURY, J. E. Synthesis and implementation of local modular supervisory control for a manufacturing cell. In: IEEE. *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*. [S.l.], 2002. p. 377–382.

RAMADGE, P. J.; WONHAM, W. M. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, SIAM, v. 25, n. 1, p. 206–230, 1987.

RAMADGE, P. J.; WONHAM, W. M. The control of discrete event systems. *Proceedings of the IEEE*, IEEE, v. 77, n. 1, p. 81–98, 1989.

SILVA, F. C. L. da. *Uma ferramenta para o ensino de inteligência artificial usando jogos de computador*. Dissertação (Mestrado) — Universidade de São Paulo, Novembro 2007.

SILVA, W.; MACIEL, P. et al. Modelling and analysis in production system: an approach based on petri net. In: IEEE. *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. [S.l.], 2004. v. 5, p. 4354–4359.

THOMAZINI, D.; ALBUQUERQUE, P. U. B. d. Sensores industriais–fundamentos e aplicações. *São Paulo*, v. 3, 2005.

TSINARAKIS, N. C. T. G. J.; VALAVANIS, K. P. Petri net modeling of routing and operation flexibility in production systems. *IEEE International Symposium*, p. 352 – 357, 2005.

WONHAM, W. M.; RAMADGE, P. J. Modular supervisory control of discrete-event systems. *Mathematics of control, Signals and Systems*, Springer, v. 1, n. 1, p. 13–30, 1988.

ZYUBIN, V. E. Hyper-automaton: a model of control algorithms. In: IEEE. *Control and Communications, 2007. SIBCON'07. Siberian Conference on*. [S.l.], 2007. p. 51–57.