UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

BRUNO MANOEL DOBROVOLSKI

APLICAÇÃO DE CONTROLE SUPERVISÓRIO PARA A COORDENAÇÃO DE ESTEIRAS CONCORRENTES EM LINHAS DE PRODUÇÃO INDUSTRIAL

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2017

BRUNO MANOEL DOBROVOLSKI

APLICAÇÃO DE CONTROLE SUPERVISÓRIO PARA A COORDENAÇÃO DE ESTEIRAS CONCORRENTES EM LINHAS DE PRODUÇÃO INDUSTRIAL

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Engenharia de Computação da Coordenação de Engenharia de Computação - COENC - da Universidade Tecnológica Federal do Paraná - UTFPR, Câmpus Pato Branco, como requisito parcial para obtenção do título de Engenheiro de Computação.

Orientador: Prof. Dr. Marcelo Teixeira

PATO BRANCO

2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

TERMO DE APROVAÇÃO

Às 13 horas e 50 minutos do dia 23 de novembro de 2017, na sala V003, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, reuniu-se a banca examinadora composta pelos professores Marcelo Teixeira (orientador), Bruno Cesar Ribas e Gustavo Weber Denardin para avaliar o trabalho de conclusão de curso com o título **Aplicação de controle supervisório para a coordenação de esteiras concorrentes em linhas de produção industrial**, do aluno **Bruno Manoel Dobrovolski**, matrícula 1343661, do curso de Engenharia de Computação. Após a apresentação o candidato foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Marcelo Teixeira
Orientador (UTFPR)

_____
Bruno Cesar Ribas
(UTFPR)

_____
Gustavo Weber Denardin
(UTFPR)

_____
Profa. Beatriz Terezinha Borsoi
Coordenador de TCC

_____
Prof. Pablo Gauterio Cavalcanti
Coordenador do Curso de
Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

DOBROVOLSKI, Bruno Manoel. Uma aplicação envolvendo controle supervisório para a coordenação de esteiras concorrentes em linhas de produção industrial. 2017. 43f. Monografia (Trabalho de Conclusão de Curso 2) - Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Este trabalho propõe uma solução para um problema de coordenação de esteiras em uma linha de produção industrial aplicando *Teoria de Controle Supervisório* (TCS). O trabalho melhora o método atual usado para determinar ações no fluxo de produção das esteiras, o qual é essencialmente manual, e a aplicação se dá na empresa *Atlas Eletrodomésticos*. Tecnicamente, o trabalho envolve as etapas de modelagem, síntese e implementação de um controlador para múltiplas esteiras que operam em paralelo e de maneira concorrente. A etapa de modelagem mapeia sensores e atuadores e regras de coordenação para uma representação em máquinas de estados. Então, a etapa de síntese recebe essa estrutura de modelos e obtém um conjunto modular de controladores que são então testados e implementados em linguagem C. Para a etapa de modularização de síntese, faz-se o uso de técnicas de *Controle Modular Local*. Já a implementação explora o uso de modelos de máquinas estados genéricas, com foco em flexibilidade de aplicação, independente de dispositivo e tecnologia. No estudo de caso, 4 supervisores foram implementados em uma única aplicação, com execução concorrente e sem um sistema operacional, o que agrega performance e baixo consumo de recursos computacionais.

Palavras-chave: Sistema a Eventos Discretos, Controle Supervisório, Controle Modular Local, Implementação, Máquinas de Estados Genéricas.

ABSTRACT

DOBROVOLSKI, Bruno Manoel. An Application of Supervisory Control for the coordination of concurrent conveyors in industrial production lines. 2017. 43f. Monograph - Computer Engineering course, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

This work proposes a solution to a problem of coordination of conveyors in an industrial production line, applying *Supervisory Control Theory* (SCT). The work improves the current method used to determine actions in the production flows, which is essentially manual, and the application takes place into the company *Atlas Home appliances*. Technically, the work involves the steps of modeling, synthesizing and implementing a controller for multiple conveyors operating in parallel and concurrently. The modeling step maps sensors and actuators, and coordination rules for representation in state-machines. Then, the synthesis step receives this structure of models and obtains a modular set of controllers that are then tested and implemented in C language. For the modularization synthesis step, the techniques of *Local Modular Control* are used. The implementation explores the use of generic state-machine models, focusing on flexibility, and device and technology independence. In the case study, 4 supervisors were implemented in a single application, with concurrent execution and without an operating system, which adds performance and low consumption of computational resources.

Keywords: Discrete-Event Systems, Supervisory Control, Local Modular Control, Implementation, Generic State Machines.

1 INTRODUÇÃO

A indústria vem buscando continuamente maior desempenho e qualidade em todas as suas etapas, para assim garantir a eficiência do processo produtivo e evitar desperdícios. Parte dessas melhorias tem sido atribuídas a aplicações de técnicas de automação sobre determinadas tarefas sequenciais. O ganho obtido com a adição de um controlador ótimo em uma linha de produção, por exemplo, pode representar um incremento substancial da produtividade e na lucratividade da fábrica, aspectos decisivos para a sobrevivência de mercado (GUTIERREZ; PAN, 2008).

Historicamente, as soluções para a implementação de sistemas de controle industriais utilizam *Controladores Lógicos Programáveis* (CLPs). Os CLPs suportam uma programação simples e intuitiva, são robustos e requerem pouca manutenibilidade. Por serem de simples programação, ocorre na maioria das vezes que o sistema de controle em si é projetado e implementado com base exclusivamente na inspiração do engenheiro. Essa alternativa, além de tender a ser ineficiente (especialmente para sistemas de grande porte) é suscetível a erros e pode apresentar um desempenho inferior à uma alternativa sintetizada por métodos mais elaborados.

Atualmente, a competitividade de mercado, associada à abundância de tecnologia, vêm promovendo a necessidade de otimização dos processos industriais. Nesse contexto torna-se inviável depender da inspiração do projetista para se obter o controle de um processo. O fato de o sistema não conhecer a possibilidade de uma falha ou de deixar a cargo de um operador determinar uma possível medida paliativa, enquanto a manutenção ocorre, pode contrair a produção programada para um determinado período de tempo e comprometer em cadeia o restante da produção.

Uma abordagem eficiente para a síntese automática de sistemas de controle é provida pela *Teoria de Controle Supervisório* (TCS), a qual encapsula toda complexidade na etapa de projeto, para entregar uma solução ótima, escalável e integrável a outros sistemas. A principal vantagem da TCS é que o projetista pode agora explorar um grau de abstração maior, e teoricamente mais simples, na hora de planejar o sistema de controle, deixando o cálculo e a implementação do controlador em si a cargo de uma operação automática. Como resultado, a TCS fornece uma versão do controlador na forma de uma máquina de estados, que pode então ser traduzida

automaticamente para uma série de formatos, incluindo formatos compatíveis com CLPs e também com linguagens de programação de dispositivos mais baratos e de maior desempenho, como é o caso dos microcontroladores. Este trabalho explora a versão modular dessa teoria sobre um problema real da indústria e conduz a etapa de implementação usando microcontroladores e técnicas avançadas de programação.

1.1 PROBLEMÁTICA

Em sua planta industrial de $34.000m^2$, localizada em Pato Branco-PR, a Atlas possui capacidade de produção de 2 milhões de peças/ano, com as linhas de produto *Top*, *Expert*, *Europa* e *Cooktops*, atende ao mercado brasileiro e exporta para 3 continentes (ATLAS, 2016). Após visita técnica na empresa, observou-se que a etapa de montagem, nas linhas de produção, ocorre de forma paralela e, ao fim, todos os produtos encaixotados são depositados manualmente em uma esteira base, que liga todas as linhas de produção a uma esteira central, conforme mostra a Figura 1.

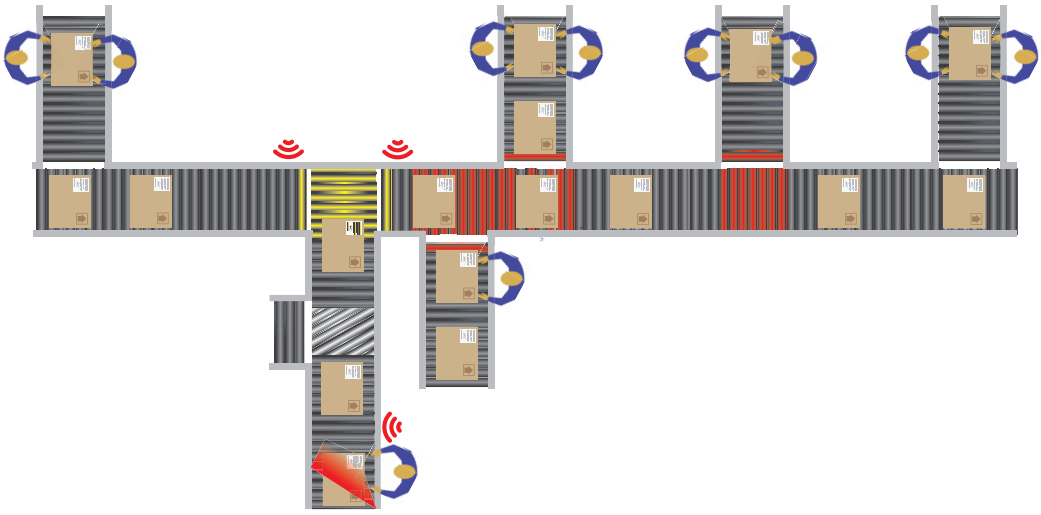


Figura 1: Vista superior do modelo das esteiras
Fonte: Autor (2016)

O objetivo da esteira central é direcionar as caixas das cinco linhas de produção existentes a uma etapa final, que desloca caixas de forma unitária em frente a um leitor de código de barras, para então serem despachadas para a estocagem ou carregamento.

A dinâmica da linha de montagem possui fatores que influenciam no tempo de escoamento da produção. Existe a chance de um colaborador errar o posicionamento da caixa na esteira de saída da sua linha de montagem. Este mesmo colaborador pode ainda precisar conhecer o tempo correto em que coloca a caixa na esteira central, pois, caso erre, poderá haver a colisão desta caixa com outra que já esteja

na esteira central, vinda de outra linha de montagem. Estas regiões de colisão são identificadas em vermelho na Figura 1.

Ainda que siga sem percalços para a esteira central, o produto pode colidir com outros, pois a esteira final está ao longo da esteira central, ou seja, há uma região de conflito - em amarelo - para onde a produção escoa. Neste ponto do processo, existe um par de sensores que só permite a passagem de uma nova caixa caso não exista outra na região crítica.

Já na esteira final, um separador - em branco - garante uma distância entre cada caixa, facilitando a leitura do código de barras. Porém, neste ponto pode haver interferência humana também, como, por exemplo, com a reinserção de produtos retirados para teste de qualidade. Por vezes, essas caixas são inseridas (empurradas pelo colaborador) sem que o separador consiga atuar, fazendo com que duas caixas prossigam muito próximas uma da outra, comprometendo a etapa de leitura.

Caso a leitura tenha sucesso, o pacote identificado segue para carregamento. Do contrário, a esteira é desligada automaticamente e um colaborador deve fazer a conferência manual do motivo da parada. A parada pode ocorrer por vários motivos, dos quais três se destacam: (i) o posicionamento rotacionado da caixa pelo colaborador na esteira base; (ii) a etiqueta com avarias; (iii) ou ainda a parada determinada pela expedição. A Figura 2 apresenta um gráfico com os dados do relatório real dos motivos de parada das esteiras, no período de 20/05/16 à 13/07/16.

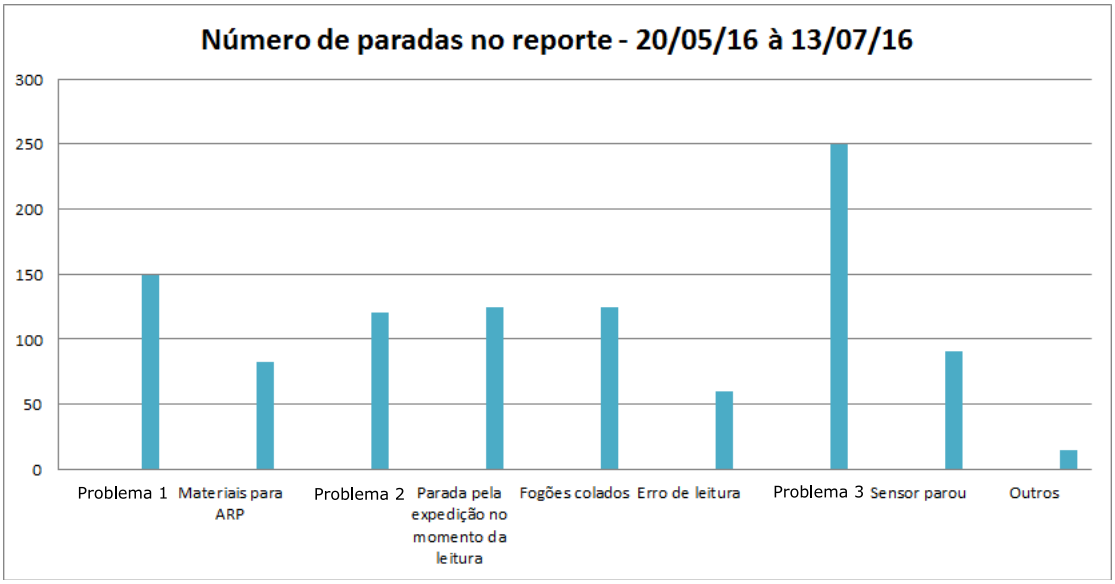


Figura 2: Gráfico dos motivos de parada da esteira
Fonte: Autor(2016)

O problema apresentado na Figura 2 como *Problema 3*, com a maior incidência no período, já foi resolvido com uma correção de processo. Os itens citados

como Problema 1, 2 e 3, tiveram a descrição omita por determinação da empresa.

É notório que estes problemas de coordenação de esteiras podem interferir no tempo de produção da empresa. Para fins práticos, tais interferências são inadmissíveis e, com o aumento da competitividade de mercado, é imprescindível que um processo concorrente, como é o caso, explore ao máximo o seu comportamento paralelo, ou seja, ser controlado de forma ótima dentro de um conjunto de requisitos, dependendo minimamente da ação humana para a tomada de decisões.

1.2 JUSTIFICATIVA

Uma possível solução a esse problema, seria a programação manual de um conjunto de regras. Contudo, a tentativa empírica de se abordar e explicitar todas as combinações possíveis para a coordenação das esteiras, esbarra na incapacidade humana para perceber o todo. Isto é, com a quantidade escalável de esteiras. Portanto, memorizar as sequências possíveis de atuação se torna um problema com crescimento da complexidade pode ser de ordem exponencial em relação aos componentes do sistema. Isso inviabiliza uma solução minimamente restritiva e passa a remeter a um comportamento controlado subótimo, o qual, na prática, é pouco atrativo.

Ao observar a operação em que o produto não é identificado pelo leitor ao fim da linha, nota-se a necessidade do reposicionamento da caixa e o retorno a uma posição anterior na esteira, ou seja, retrabalho para aquele item. A solução por software de uma operação de retrabalho pode não ser tão simples, pois é necessária a identificação de qual a posição do item reoperado, para que este seja reconhecido como defeituoso após uma quantidade de tentativas (TEIXEIRA, 2013a). Apesar de existirem soluções desenvolvidas para esse tipo de problema, como Variáveis e Distinguidores (TEIXEIRA, 2013b), uma solução física, como um adesivo, traria resultados mais eficientes para este problema frente o aumento da complexidade de implementação.

O ramo industrial vem buscando soluções que agreguem a otimização dos processos. Grande parte dessa busca se dá em redefinir e melhorar a lógica de controle usada para coordenar os componentes de atuação em chão de fábrica. Nesse sentido, o estudo de sistemas de controle dirigidos por eventos discretos (SEDs) se mostra eficiente no desenvolvimento dessas soluções (SANTOS, 2013).

O desafio observado ao controlar um SED é, portanto, encontrar uma lógica de controle que possa coordenar o sistema de forma objetiva, simples e ótima. A *Teo-*

ria de Controle Supervisório (TCS), metodologia esta que foi proposta por Ramadge e Wonham (1989), é uma alternativa formal que pode viabilizar a obtenção automática de lógicas ótimas de controle para SEDs . Essa abordagem, baseada em *Linguagens Formais* e *Autômatos*, recebe como entrada um modelo para a planta do sistema e um conjunto de especificações para então, fornecer como saída um supervisor (controlador) calculado por meio de uma operação matemática de síntese, que restringe a planta ao comportamento desejado agindo de maneira minimamente restritiva e não-bloqueante, requisitos vitais na indústria.

Inúmeras extensões vêm sendo associadas à TCS na literatura como forma de adequá-la ao propósito industrial, haja visto que a sua estrutura baseada em máquinas de estados, quando em grande escala, pode tornar trabalhosa a análise e manipulação, tanto manual, quanto computacionalmente.

Neste trabalho, os conceitos apresentados acima sendo combinados para resolver o problema da coordenação das esteiras de escoamento de produtos acabados da empresa Atlas Eletrodomésticos.

1.3 OBJETIVOS

A seguir apresentam-se os objetivos gerais e específicos do trabalho.

1.3.1 OBJETIVO GERAL

Viabilizar a modelagem, a síntese e a implementação de um sistema de *Controle Supervisório*, capaz de coordenar, de maneira ótima, um sistema de esteiras de escoamento de produção.

1.3.2 OBJETIVOS ESPECÍFICOS

Esquematicamente, o objetivo geral da proposta foi alcançado por meio de uma sequência de passos, caracterizados como segue:

- Elaborar uma análise detalhada do processo existente;
- Identificar o conjunto de componentes, sensores, atuadores, entre outros, etapas já automatizadas e os novos equipamentos necessários para o alcance dos pressupostos;
- Modelar a planta do sistema em malha aberta;

- Definir um conjunto de especificações para o processo integrando-as à planta para o fechamento da malha de controle;
- Sintetizar um controlador modular que respeite o conjunto de especificações, que seja não-bloqueante e que atue de maneira minimamente restritiva sobre o processo;
- Implementar o sistema de controle, ilustrando suas características ao coordenar as esteiras e os ganhos provenientes das escolhas de projeto, como a adoção do controle modular.

2 FUNDAMENTAÇÃO TEÓRICA

Vários sistemas comumente vistos diariamente possuem técnicas de controle agregadas, as quais determinam ações a serem tomadas. Um sistema de controle pode ser compreendido como a composição de subsistemas e processos (ou plantas) que são unidos para o propósito de se obter uma saída com o desempenho desejado, para uma certa especificação (NISE, 2007). Esta definição de sistema compreende todo tipo de controle conhecido hoje, como controle clássico, analógico, digital, adaptativo, entre outros (OGATA; YANG, 1970).

2.1 SISTEMAS DE CONTROLE

As técnicas utilizadas para a obtenção de um controlador dependem basicamente do tipo do processo a ser controlado. Em um sistema que evolui de maneira contínua no tempo, por exemplo, é comum o emprego de equações diferenciais para a programação da saída do controlador, após determinado tempo, dada uma determinada entrada. Em outros casos, o processo a ser controlado pode não depender do tempo, mas de eventos assíncronos que ocorrem esporadicamente, em intervalos irregulares (CASSANDRAS; LAFORTUNE, 2008). Nesses casos, o emprego de equações diferenciais se torna inadequado e é comum que engenheiros optem por adotar diagramas de estados para a obtenção de uma estrutura de controle que absorva a ocorrência espontânea de eventos. Em alguns casos, ainda, pode ocorrer que um mesmo processo integre estruturas dirigidas por eventos e pelo tempo, o que remete aos chamados sistemas *híbridos*. Dada a natureza desse trabalho, a atenção será voltada pelos sistemas independentes no tempo, os quais foram descritos em mais detalhes a seguir.

2.2 SISTEMAS A EVENTOS DISCRETOS

Quando discretizado, um sistema pode ser descrito por um conjunto enumerável de estados, como $\{0, 1, 2, 3, ...\}$, e as transições partindo de cada um destes estados são observadas com pontos discretos no tempo. Isso caracteriza a classe de sistemas dinâmicos independentes do tempo, chamados de *Sistemas a Eventos Dis-*

cretos (SEDs). Para Cury (2001), SED é um sistema dinâmico que evolui de acordo com a ocorrência abrupta de eventos físicos em intervalos de tempo, em geral, irregulares e desconhecidos. Dada essa dinâmica de comportamento, um SED pode ser descrito naturalmente por diagramas de estados, apresentados a seguir.

2.2.1 MODELAGEM EM SED

A representação de um sistema real por um modelo permite a realização de testes que talvez não fossem possíveis no sistema físico. Porém, para a construção adequada de um modelo, são necessárias algumas considerações. A definição sobre o domínio discreto requer a utilização de métodos que deem suporte à controlabilidade necessária.

Dentre os formalismos mais difundidos para a modelagem de SEDs, como as *Redes de Petri* e a *Teoria das filas*, a *Teoria dos Autômatos e Linguagens* se destaca por permitir a síntese automática de controladores ótimos (TEIXEIRA, 2013a).

2.2.2 LINGUAGENS FORMAIS

A estrutura básica de uma linguagem formal, assim como uma linguagem falada, é a palavra/cadeia/string, formada por letras/símbolos/caracteres, tirados de um alfabeto. Por exemplo, o conjunto que representa o alfabeto da língua portuguesa tem 26 letras $\{a, b, c, ..., x, y, z\}$ que, quando combinadas, levam à construção de palavras válidas no idioma.

Em linguagens formais, a ideia é análoga, exceto pelo fato de que as linguagens geradas podem possuir um sentido claro apenas no contexto computacional. Por exemplo, um alfabeto particular pertinente a teoria da computação é o binário $\{0, 1\}$ (LEWIS; PAPADIMITRIOU, 1997), que pode levar a construção de cadeias cujo significado pode não parecer tão nítido quanto uma linguagem falada.

Do ponto de vista formal, define-se o alfabeto como um conjunto finito de símbolos, para o qual existem infinitos arranjos de elementos que, justapostos, formam palavras (MENEZES, 1998). Portanto, considerando um alfabeto denotado por Σ, o conjunto de todas as cadeias a serem compostas com elementos de Σ é chamada de Σ^* (HOPCROFT; ULLMAN; MOTWANI, 2001). Cada uma destas cadeias em Σ^* pode ser considerada uma palavra e pode conter múltiplos ou nenhum símbolo. Neste último caso, é chamada de palavra/cadeia vazia, e denotada por ϵ (LEWIS; PAPADIMITRIOU, 1997). O tamanho de uma palavra é definido pela quantidade de caracteres associa-

dos a ela. Portanto, seja s uma palavra de Σ^*, onde s possui apenas o evento a, seu tamanho $\mid s \mid$ é 1, da mesma forma, se $j \in \Sigma^*$ e $j = abc$, logo $\mid j \mid = 3$.

Considerando uma cadeia $p = ghj$ onde $g, h, j \in \Sigma^*$, definimos os elementos g e j como as operações morfológicas de prefixo e sufixo, respectivamente e h como subcadeia (RAMADGE; WONHAM, 1989). Qualquer conjunto de palavras sobre Σ - ou seja, um subconjunto de Σ^* - será uma linguagem, esta pode ser compreendia como um idioma sobre o alfabeto supracitado, á que, cada idioma é um conjunto $L \subset \Sigma^*$. A notação de linguagem toma aqui a mesma concepção.

Sendo uma linguagem L, onde $L \subseteq \Sigma^*$, o prefixo-fechamento $\overline{L}$ de uma linguagem se dá pela seguinte definição (MENEZES, 1998):

$$\overline{L} = \{s \in \Sigma^* \mid (\exists t \in \Sigma^*), st \in L\} \tag{1}$$

Logo, se L for prefixo-fechada, então $\overline{L} = L$. Ao ler esta definição, entende-se que para qualquer cadeia s de L, deve existir também, uma sub-cadeia prefixo de s. Por exemplo, para duas linguagens $L_1 = \{\epsilon, a, s, d, aa, ad, aas\}, L_2 = \{\epsilon, a, s, d, asd, aas\}$ sobre o alfabeto $\Sigma = \{a, s, d\}$, L_1 é dita prefixo-fechada, pois todos os prefixos da linguagem são também elementos. Já L_2 não é prefixo-fechado, pois o prefixo aa da cadeia aas não é elemento de L_2.

Considerando que cada elemento de Σ fosse associado a eventos de um sistema, então uma cadeia composta por eventos de Σ poderia ser associada uma determinada sequência de passos ocorridos no sistema. De maneira análoga, Σ^* representaria todas as possíveis cadeias compostas com eventos em Σ e $L \subseteq \Sigma^*$ poderia representar, dentre todas as cadeias, justamente aquelas de interesse, ou seja, as que são possíveis no sistema.

A questão que surge nesse ponto é: como uma linguagem L poderia ser definida de modo tal que refletisse exatamente as ações do sistema que se está modelando? Uma alternativa seria modelar o próprio sistema por um diagrama de estados e, a partir dele, reconhecer a linguagem de interesse. Ou seja, para a representação de um processo real por meio de linguagens é necessário dispor de uma estrutura que possibilite manipular e analisar essas linguagens (CASSANDRAS; LAFORTUNE, 2008). Uma dessas estruturas são os Autômatos Finitos.

2.2.3 AUTÔMATOS FINITOS

Autômatos são dispositivos capazes de reconhecer linguagens de acordo com regras bem definidas (CASSANDRAS; LAFORTUNE, 2008), a serem representados sobre máquinas finitas de estados. Por efinição, um Autômato Finito (AF) é uma quíntupla ordenada G (MENEZES, 1998):

$$G = \langle \Sigma, Q, q^o, Q^\omega, \rightarrow \rangle \tag{2}$$

Onde:

- Σ é o alfabeto finito;
- Q o conjunto finito de estados;
- $q^o \in Q$ é o estado inicial;
- $Q^\omega \subseteq Q$ é o conjunto de estados marcados;
- $\rightarrow \subseteq Q \times \Sigma \times Q$ é a relação de transição entre estados o conjunto que define a transição entre estado.

A representação gráfica de autômatos é inspirada na teoria de grafos e composta por nós e arestas, como mostra a Figura 3.

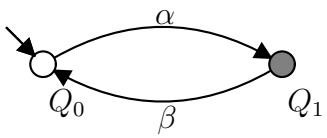


Figura 3: Exemplo básico de um autômato
Fonte: Autor(2016)

A seta sobre o estado Q_0 denota ser o estado inicial, e o preenchimento em cinza de Q_1 denota ser o estado final, logo $q^o = Q_0$ e $Q^\omega = \{Q_1\}$; As transições entre os estados Q_0 e Q_1 com a ocorrência dos eventos α e β, tal que $\Sigma = \{\alpha, \beta\}$. Então, uma transição é definida como $Q_0 \xrightarrow{\alpha} Q_1$.(CASSANDRAS; LAFORTUNE, 2008)

Um autômato G pode ser associado a duas linguagens, $L(G)$, que representa todas as cadeias que o autômato possa representar, e $L^\omega(G)$ que é composta por todas as *strings* marcadas pela linguagem, ou seja, possui cadeias com sufixo pertencente a Q^ω. É notável que $L^\omega(G) \subseteq L(G)$. Uma cadeia é considerada marcada

or G quando o evento inicial é o mesmo que q^o e seu evento final pertence ao conjunto Q^ω de estados finais(CASSANDRAS; LAFORTUNE, 2008).

Da forma definida acima, em um autômato há sempre um único estado inicial e cada evento σ ocorrido é determinístico, ou seja, toda operação de transição ocorrida, desde que rotulada com um mesmo evento, leva obrigatoriamente a apenas um estado (CASSANDRAS; LAFORTUNE, 2008). Formalmente, um autômato será determinístico se, para qualquer de suas transições, $x \stackrel{a}{\rightarrow} y_1$ e $x \stackrel{a}{\rightarrow} y_2$ necessariamente implica que $y_1 = y_2$. Para este trabalho, assumiu-se que autômatos são determinísticos.

A modelagem de sistemas reais por autômatos envolve, naturalmente, uma quantidade finita de eventos, logo, o alfabeto composto por estes eventos, será também finito. Por este motivo, é de interesse desse trabalho uma classe particular de linguagens denominadas *regulares*. Uma linguagem é dita regular quando pode ser reconhecida por um AF, ou seja, o alfabeto de estados é finito, mesmo que existam infinitas cadeias possíveis. (CASSANDRAS; LAFORTUNE, 2008).

2.3 TEORIA DE CONTROLE SUPERVISÓRIO

Uma abordagem formal que permite incorporar a controlabilidade de eventos na síntese de controladores para SEDs é a *Teoria de Controle Supervisório* (TCS) (RAMADGE; WONHAM, 1989). Na TCS, o conjunto Σ de eventos modelados é subdivido em *controláveis*, denotado por Σ_{c}, os quais representam as ações que podem sofrer intervenção ou serem acionadas manualmente e *não controláveis*, denotado Σ_{u}, cujas as ocorrências não são diretamente manipuláveis (RAMADGE; WONHAM, 1987b).

O elemento da TCS que efetivamente implementa as ações de controle na planta, sintetizando a questão da controlabilidade, ou seja, um sistema que permita ações de controle sem perder sua funcionalidade, é o *supervisor*. Um supervisor S é uma estrutura associada à planta G que, após qualquer cadeia $s \in L(G)$, observa os eventos possíveis na planta e informa, dentre eles, quais devem ser habilitados (CURY, 2001).

O *Problema de Controle Supervisório* (PCS) consiste, então, em obter um supervisor S tal que sua ação sobre G (S/G) satisfaça o conjunto de especificações $K\ L^\omega(S/G) \subseteq K$) e, para isso, desabilite apenas eventos controláveis.

A *controlabilidade* é uma condição necessária e suficiente para a existência de S. Uma linguagem $K \subseteq \Sigma^*$ é *controlável* em relação a uma linguagem prefixo-

fechada $L(G)$ se

$$\overline{K}\Sigma_{\mathrm{u}} \cap L(G) \subseteq \overline{K} \tag{3}$$

Ou seja, se após qualquer prefixo s, um evento $\mu \in \Sigma_{\mathrm{u}}$ é elegível em $L(G)$, então μ não é desabilitado, i.e., $s\mu \in \overline{K}$ (LEWIS; PAPADIMITRIOU, 1997).

Se uma especificação $K \subseteq L(G)$ é controlável em relação a $L(G)$, então a solução de controle pode ser implementada por um autômato V que representa K, para $K = L^{\omega}(V) \cap L^{\omega}(G)$ e $\overline{K} = L(V) \cap L(G)$. Se K não for controlável, se faz necessário reduzi-la à sua *máxima sublinguagem controlável* (CURY, 2001).

$$\mathrm{sup}\mathcal{C}(K,G) = \bigcup \{\, K' \subseteq K \mid K' \text{ é controlável em relação a } L(G) \,\}\,. \tag{4}$$

O processo de computar $\mathrm{sup}\mathcal{C}(K,G)$ é conhecido como *síntese* (CASSANDRAS; LAFORTUNE, 2008) e nada mais é do que uma operação matemática que implementa a noção de controlabilidade, i.e., o algoritmo de síntese extrai de K um sub-modelo K' que trata da impossibilidade de interferir diretamente em eventos em Σ_{u}.

Assim, $\mathrm{sup}\mathcal{C}(K,G)$ representa o comportamento menos restritivo possível de ser implementado por um supervisor S ao controlar G, respeitando a especificação K. Se $\mathrm{sup}\mathcal{C}(K,G)$ for ainda não-bloqueante, então $L^{\omega}(S/G) = \mathrm{sup}\mathcal{C}(K,G)$ e $L^{\omega}(S/G) = \overline{\mathrm{sup}\mathcal{C}(K,G)}$ é uma solução *ótima* para o PCS.(MONTGOMERY, 2004)

2.4 CONTROLE MODULAR

Em alguns problemas de controle, principalmente naqueles envolvendo sistemas de grande porte, compostos por inúmeros subsistemas e especificações, a obtenção de um supervisor monolítico pode ser computacionalmente inviável. Isso, porque uma operação monolítica requer a composição, em uma única estrutura, dos autômatos considerados no problema, o que pode levar ao que se conhece como *explosão do espaço de estados*. Essa situação ocorre quando os recursos necessários para tratar de um problema (uma execução computacional) ultrapassam a capacidade e os recursos disponíveis para tal. Não é difícil gerar a explosão do espaço de estados. No presente trabalho, por exemplo, para cada linha de produção incluída ao sistema supervisor geral o número de estados e transições são multiplicados por 3. Para o modelo com 3 linhas há cerca de 15 mil transições, quando adicionada a quarta linha esse valor sobe para 45 mil.

Alternativamente, nas extensões da TCS têm-se sugerido a modularização do problema de controle, de modo que partes possam ser tratadas individualmente e, de maneira mais simples. Então, condições são providas para mostrar que a ação conjunta das soluções modulares resolve o problema global de controle. São inúmeras as formas de modularização sugeridas, incluindo o particionamento baseado em estágios composicionais (MOHAJERANI, 2011), em eventos gerados pelo sistema (CAI; WONHAM, 2010), em especificações de controle (RAMADGE; WONHAM, 1987a), em plantas e especificações (QUEIROZ; CURY, 2000a).

2.5 CONTROLE MODULAR CLÁSSICO

O *Controle Modular Clássico* (CMC) (RAMADGE; WONHAM, 1987a; WONHAM; RAMADGE, 1988) representa uma alternativa para minimizar a sobrecarga computacional decorrente de um procedimento monolítico de síntese. No CMC, ao invés de se construir um único supervisor S que atenda a um conjunto global de especificações $I = \{1, \cdots, m\}$, sintetiza-se um supervisor modular S_i para cada especificação $i \in I$. A Figura 4 ilustra a arquitetura do CMC.

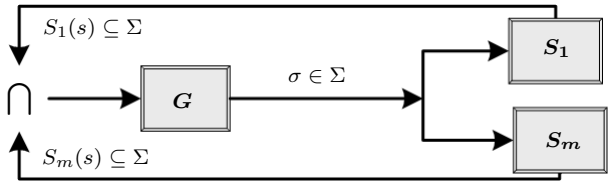


Figura 4: Estrutura de controle no CMC.

Toda vez que, após uma cadeia $s \in \Sigma^*$ qualquer, um evento $\sigma \in \Sigma$ for observado na planta, cada supervisor modular mapeia um subconjunto $S_i(s) \subseteq \Sigma$ de eventos a serem habilitados. Assim, se um evento é observado e não fizer parte de ambos os subconjuntos $S_i(s)$, ele é desabilitado na planta.

Um dos principais benefícios do CMC é a flexibilidade agregada pela resolução de um problema de controle por módulos, o que permite alterar, inserir ou remover um requisito em particular, sem afetar o sistema como um todo. Além disso, as operações individuais de síntese são, em geral, mais simples se comparadas à uma operação monolítica.

2.5.1 COMPLEXIDADE DE SÍNTESE NO CMC

Seja um SED modelado por $p + q$ subsistemas, com no máximo m estados cada, de modo que m^{p+q} é o número de estados do autômato que modela a planta G.

Ainda, seja G restrito pelas especificações $E^i,\ i \in I = \{1, \cdots, e\}$, tendo E^i no máximo n estados. Considerando que cada E^i restringe apenas uma parte p dos subsistemas $p+q$, e que q compõe os subsistemas que não são afetados por nenhuma das especificações, então q não precisa fazer parte da síntese dos supervisores modulares.

Logo, a complexidade de síntese no CMC é na ordem de $\mathcal{O}[(n.m^p)^2]$ e, portanto, é claramente mais simples do que no PCS, onde é na ordem de $\mathcal{O}[(n^e.m^{p+q})^2]$. Na prática, as vantagens do CMC em relação ao PCS vertem de duas fontes principais: (i) do fato de que as especificações não são compostas entre si; e (ii) do uso de um modelo parcial da planta na síntese, o qual não contempla os subsistemas assíncronos com as especificações.

Porém, note que é pouco razoável que um subsistema que não precise ser controlado (não compartilhe eventos com nenhuma especificação) faça parte do problema de controle. Logo, pode-se assumir que a planta G compõe, na verdade, todos os $p+q$ subsistemas. Como o fator $p+q$ determina o crescimento de G, então o CMC se torna degradado pela impossibilidade de tratar de problemas de grande porte. Uma extensão do CMC, o Controle Modular Local, revitaliza o conceito de controle supervisório modular, e é apresentada na sequência.

2.6 CONTROLE MODULAR LOCAL

O *Controle Modular Local* (CML) (QUEIROZ; CURY, 2000a; QUEIROZ; CURY, 2000b) contorna uma das principais lacunas do CMC: o fato de que, embora a síntese seja descentralizada, o modelo da planta considerado para a síntese é ainda monolítico, o que pode ser um empecilho computacional para resolver problemas de controle.

Similarmente à abordagem clássica, o CML também projeta a construção de um supervisor para cada especificação $i \in I$ envolvida no problema. A diferença é que cada supervisor é obtido a partir de uma versão parcial da planta e, logo, exerce a função de um *supervisor local* para o sistema. A Figura 5 sumariza a arquitetura do CML.

Nessa arquitetura, plantas locais são denotadas por G^i_{loc} e são construídas por composições específicas de modelos de subsistemas. Quando um evento $\sigma \in \Sigma$ é observado localmente, o respectivo supervisor S^i_{loc} mapeia um subconjunto de eventos $S^i_{loc}(s) \subseteq \Sigma$ a serem habilitados na planta local. Eventos possíveis em G^i_{loc} que não fazem parte de $S^i_{loc}(s) \subseteq \Sigma$ são inibidos pela ação de controle.

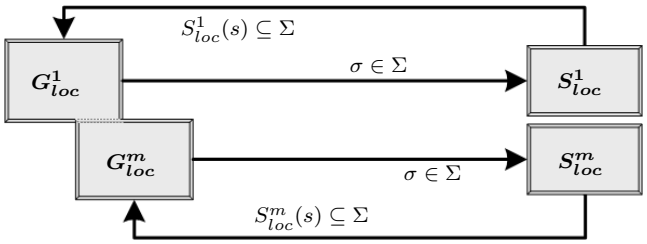


Figura 5: Estrutura de controle no CML.

A escolha por quais plantas devem ser compostas localmente é regida pela intersecção entre os alfabetos das plantas modulares e o de uma especificação em particular. A Figura 6 exemplifica a composição de plantas locais para um caso específico envolvendo $m = 3$ modelos de especificações e $n = 4$ modelos de plantas assíncronas.

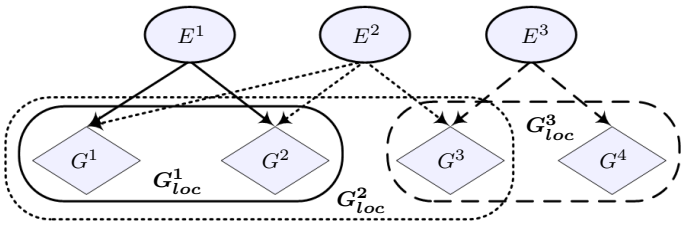


Figura 6: Composição de plantas locais no CML.

Na figura, os conectivos entre as especificações e as plantas denotam o compartilhamento de eventos entre eles. Note que as plantas locais G^1_{loc}, G^2_{loc} e G^3_{loc}, construídas respectivamente para as especificações E^1, E^2 e E^3, são compostas por todas e somente as plantas que possuem eventos em comum com a especificação. De maneira mais formal, para um SED cuja planta é modelada por autômatos assíncronos $G^j,\ j \in J = \{1, \cdots, n\}$, definidos em Σ^{G^j}, seja $E^i,\ i \in I = \{1, \cdots, m\}$ um conjunto de especificações modeladas em Σ^{E^i}. Para cada E^i, defina uma planta local G^i_{loc} tal que:

$$G^i_{loc} = \Big|\Big|^{j \in J_i} G^j \tag{5}$$

$$J_i = \{j \in J \mid \Sigma^{G^j} \cap \Sigma^{E^i} \neq \emptyset\}. \tag{6}$$

O CML define, então, o uso de G^i_{loc} na síntese de supervisores locais. Na sequência, o PCS é reintroduzido para incorporar essa ideia.

Dois aspectos relevantes surgem no contexto do PCS-ML. O primeiro é a necessidade de se conhecer as implicações do uso de uma planta local (parcial) para obter um supervisor S^i_{loc}, em substituição de uma planta monolítica. O segundo aspecto remete à relação global de equivalência entre o conjunto de soluções locais para o PCS-ML e uma solução monolítica para o PCS. Esses dois aspectos são apresentados a seguir.

2.6.1 UMA ANÁLISE LOCAL DO PCS-ML

Como cada supervisor S^i_{loc} é obtido a partir de uma planta local, é fundamental investigar a equivalência de S^i_{loc} em relação a um supervisor correspondente, caso esse fosse obtido usando o modelo completo da planta. Mostrar tal equivalência passa, sobretudo, por mostrar que o “complemento" da planta, ou seja, os modelos que não fazem parte da planta local, não é de fato necessário à síntese. Essa equivalência é formalizada a seguir e se apoia no conceito de planta complementar.

Para E^i, G^j e G^i_{loc},supracitados, seja C^i tal que:

$$C^i = \{j \in J \mid \Sigma^{G^j} \cap \Sigma^{E^i} = \emptyset\}. \tag{7}$$

Uma planta complementar G^i_c é definida para G^i_{loc} como:

$$G^i_c = \begin{cases} \|_{j \in C^i} G^j & \text{se } C^i \neq \emptyset \\ G^i_{loc} & \text{do contrário .} \end{cases} \tag{8}$$

Interpretando o formalisto, temos que uma planta complementar para G^i_{loc} compõe todo G^j que não faz parte da planta local. Se G^i_{loc} acopla todo G^j, então não há complemento de planta e G^i_c é assumida como sendo a própria planta local. A seguinte proposição mostra o efeito de desconsiderar uma planta complementar da síntese de supervisores locais.

Como porposição (QUEIROZ; CURY, 2000a) - Para E^i, G^i_{loc} e G^i_c, como definidos acima, é verdade que

$$\sup\mathcal{C}(E^i, G^i_{loc}) \parallel L^\omega(G^i_c) = \sup\mathcal{C}(E^i, G^i_{loc} \parallel G^i_c). \tag{9}$$

Ou seja, se um modelo de subsistema não é afetado por eventos de uma especificação, então ele pode ser removido da síntese local sem que isso interfira no cálculo do supervisor. O efeito disso, globalmente, é investigado como segue.

2.6.2 UMA ANÁLISE GLOBAL DO PCS-ML

Ainda que supervisores locais sejam não-bloqueantes, a ação conjunta desses supervisores leva a um comportamento global que pode ser conflitante. Isso porque, sendo cada supervisor local obtido com base em uma especificação particular, sua síntese não leva em conta os demais aspectos de controle, definidos por outras especificações. Assim sendo, cada supervisor observa e controla localmente a

tomada de recursos do sistema, pelos subsistemas. Logo, toda vez que um recurso é compartilhado por mais de um subsistema, nasce a possibilidade de conflitos, os quais seriam naturalmente prevenidos pela síntese monolítica. A condição necessária e suficiente para a garantia do não-conflito é:

$$\left|\right|_{i=1}^{m} \overline{L^{\omega}(S^i_{loc}/G^i_{loc})} = \overline{\left|\right|_{i=1}^{m} L^{\omega}(S^i_{loc}/G^i_{loc})}. \tag{10}$$

Ou seja, sempre que um prefixo é compartilhado pelos supervisores locais, ao menos uma cadeia marcada, contendo aquele prefixo, também é compartilhada.

O teste de não-conflito é a etapa mais complexa para resolver o PCS-ML, já que requer a composição de todos os supervisores locais. Isso se equipara, na verdade, à complexidade para tratar do PCS monolítico, o que reverte parte das vantagens do CML. Algumas técnicas, porém, permitem reduzir a complexidade desse teste. O uso de abstrações de linguagens (PENA; CURY; LAFORTUNE, 2009) e procedimentos de verificação composicional (FLORDAL; MALIK, 2009), por exemplo, permitem que o teste de não-conflito seja equivalentemente processado sem necessitar, entretanto, da plena composição dos supervisores locais.

Se o supervisores locais forem globalmente não-conflitantes, então a ação conjunta desses supervisores leva a um comportamento equivalente ao do supervisor monolítico correspondente. (QUEIROZ; CURY, 2000b) Formalizando essa afirmação, dados G e K, como no PCS, e G^i_{loc} e E^i, como no PCS-ML, se o conjunto $\{\sup\mathcal{C}(E^i, G^i_{loc}),\ i = 1, \cdots, m\}$ for não-conflitante, então

$$\sup\mathcal{C}(K, G) \quad = \quad \left|\right|_{i=1}^{m} \sup\mathcal{C}(E^i, G^i_{loc}). \tag{11}$$

Os casos em que essa condição não é observada requerem políticas de resolução de conflitos. A centralização de módulos locais específicos, por exemplo, é uma alternativa natural para isso. Outras alternativas incluem técnicas de coordenação de eventos (WONG; WONHAM, 1998), de prioridade de execução (CHEN; LAFORTUNE; LIN, 1995), etc.

2.6.3 COMPLEXIDADE DA SÍNTESE MODULAR LOCAL

Seja um SED modelado por p subsistemas com no máximo m estados cada. Então, m^p é o número máximo de estados para o autômato que modela a planta. Ainda, seja esse sistema restrito por e especificações modeladas com no máximo n estados. Como cada especificação restringe apenas uma parte k dos p subsistemas,

então o cálculo de um supervisor modular local tem complexidade na ordem de

$$\mathcal{O}[(n.m^k)^2].$$

Comparativamente, no CMC a complexidade é na ordem de

$$\mathcal{O}[(n.m^p)^2]$$

e, no caso monolítico, na ordem de

$$\mathcal{O}[(n^e.m^p)^2].$$

Claramente, a síntese no CML tende a ser computacionalmente mais vantajosa do que no CMC e, consequentemente, do que no procedimento monolítico. De fato, o fator de complexidade p é diretamente dependente do número de subsistemas envolvidos no problema, enquanto que o fator k varia apenas em razão do número de subsistemas afetados pela especificação. Isso torna o porte do sistema, sob certo aspecto, irrelevante no CML.

3 DESENVOLVIMENTO

Este capítulo apresenta materiais, métodos e as etapas de projeto, modelagem, síntese, simulação e implementação de um sistema de controle para as esteiras que integram as linhas de produção da Atlas Eletrodomésticos.

3.1 MATERIAIS E MÉTODOS

A etapa de análise do sistema aconteceu diretamente na empresa, coletando dados necessários, como quantidade de sensores, quantidade de produtos por linha e requisitos necessários para a empresa.

A etapa seguinte foi a modelagem da planta do sistema e de seus requisitos de controle. Ambos foram submetidos a um processo de análise e documentação da versão textual das estruturas a serem representados via modelo. Então, cada estrutura foi modelada usando-se o formalismo das máquinas de estados finitos (HOPCROFT; ULLMAN; MOTWANI, 2001), por meio do *software Supremica* (AKESSON, 2013).

Após a construção, os modelos foram simulados e as ações de controle validadas. Houve uma etapa de análise manual do cumprimentos das especificações sobre a planta, etapa esta que precedeu à síntese do controlador minimamente restritivo e não-bloqueante, que foi conduzido conforme a *Teoria de Controle Supervisório* (RAMADGE; WONHAM, 1989).

Por fim, passou-se para a etapa de implementação do controlador para testes de bancada. O autômato que representa o supervisor do sistema foi codificado em linguagem *C*, e o sistema foi implementado e executado, simulando ações reais.

3.2 DEFINIÇÕES E MODELAGEM

A etapa de análise e definição do formato dos modelos permitiu uma observação modular do sistema, modificando o modelo inicialmente proposto. Para os itens da Figura 2, as soluções não computacionais foram: (i) Instrução de trabalho e duplicação do leitor de código de barras, evitando problema de posicionamento; (ii) etiquetadora automática, para manipulação sem avarias da etiqueta. Portanto, a

modelagem teve seu foco em otimização do fluxo de produtos para escoamento da produção. A proposta apresentada na Figura 7, representa um modelo macro desse cenário. Cada componente será explicado a seguir

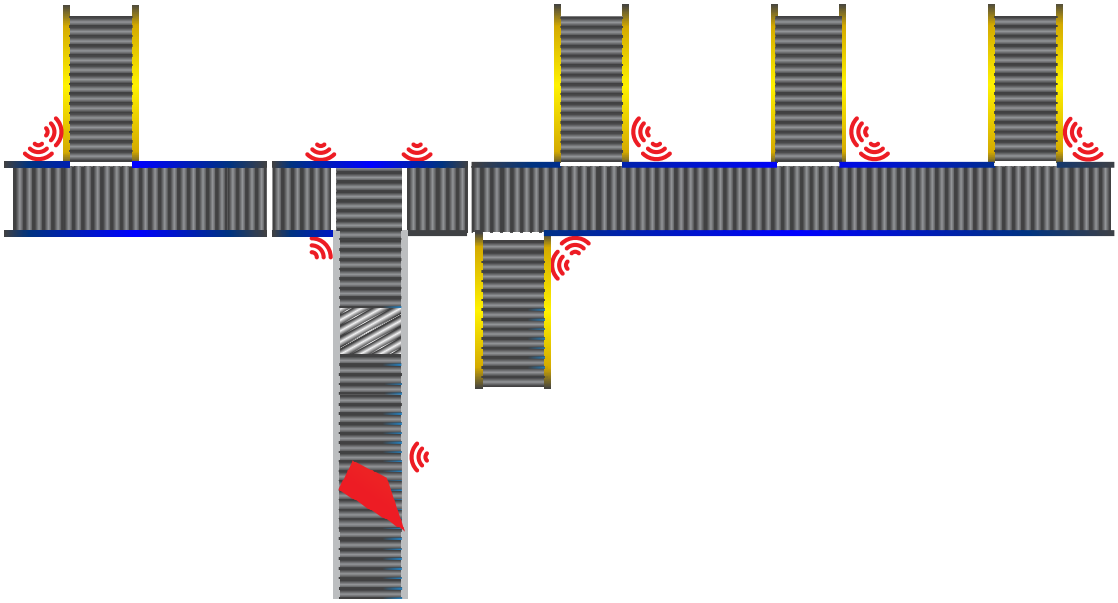


Figura 7: Vista superior do modelo das esteiras
Fonte: Autor (2017)

3.2.1 ESTEIRAS E SENSORES

Como é possível observar na Figura 7, apesar de tamanhos diferentes, todas as esteiras contam com sensores de presença. Nas esteiras de linha (com lateiras amarelas) o sensor é posicionado ao fim, já nas centrais (laterais azuis), há sensores que antecedem a intersecção com cada linha. Outro ponto observado na construção desse modelo, foi possibilitar que cada linha tivesse dependência apenas da esteira central, para que a escalabilidade de linhas fosse possível, permitindo ao supervisor de cada linha observar apenas os sensores da linha e o que a antecede na esteira central.

Uma característica desejável na máquina de estados que irá modelar as esteiras é que ela não deve parar ao identificar a presença de apenas uma caixa, seja ela vinda da linha ou já do escoamento central. Por outro lado, ela deve possibilitar essa parada em casos de conflito.

O primeiro passo da modelagem consistiu em identificar os eventos do sistema. A esteira possui os eventos E^i_{on} e E^i_{off}, que são controláveis, pois é possível dispará-los, ou seja, são saídas do sistema. Já os eventos dos sensores (Si_on e Si_off) são sinais de leitura, ou seja, não-controláveis.

Usando esses eventos, é possível modelar uma máquina de estados genérica que representa fielmente o comportamento de cada esteira. O modelo proposto é apresentado na Figura 8.

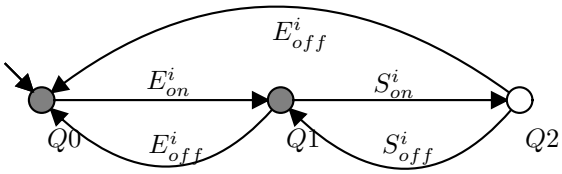


Figura 8: Modelo genérico das esteiras
Fonte: Autor (2017)

Neste modelo de planta é possível observar que a partir do estado inicial o evento de ligar a esteira (E^i_{on}) evolui para o estado em que é possível desliga-la (E^i_{off}) ou reconhecer uma caixa (S^i_{on}). Caso uma caixa seja reconhecida, há novamente a possibilidade de desligar a esteira e também é possível reconhecer a saída da caixa da área de detecção do sensor (S^i_{off}). Desta forma, em tempo de implementação é possível aguardar a instrução de desligamento da esteira ou a saída da caixa. Os estado marcados $Q0$ e $Q1$ denotam uma tarefa completa, de detecção do produto ou de desligamento da esteira

3.2.2 ESCOAMENTO DAS LINHAS DE PRODUÇÃO

Como explicado na seção 3.2.1, observou-se que todas as linhas tem o mesmo comportamento e podem ser individualizadas, portanto o modelo apresentado na Figura 9 foi tomado como escopo para desenvolvimento da planta e especificações para as linhas de montagem.

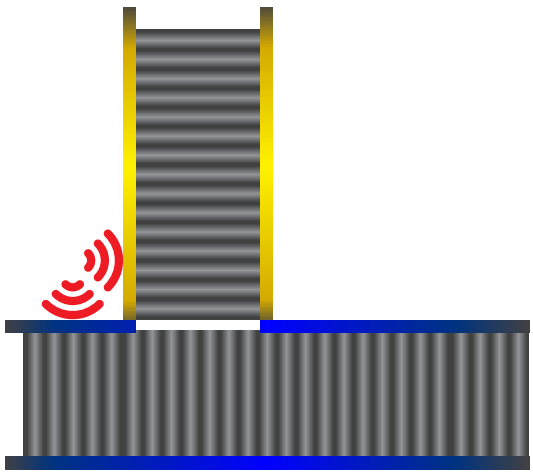


Figura 9: Componentes da região de escoamento das linhas de produção
Fonte: Autor (2017)

A Figura 9 apresenta 2 esteiras, cada uma com um sensor de presença.

Em amarelo, representa a linha de montagem e tem seu fluxo direcionado à esteira central, em azul. Um sensor está fixo ao fim da linha de montagem, próximo a região de descarga da linha, e o outro está na esteira central a fim de detectar a passagem de caixas provenientes das outras diversas linhas.

Quando detectada uma caixa na linha, uma ação simples de controle seria parar a esteira central, absorver a caixa e então reativá-la. Porém, esta solução gera a perda de desempenho e, com o aumento de demanda, esta esteira horizontal ficaria maior parte do tempo parada. A solução proposta neste trabalho permite aproveitar melhor o funcionamento das esteiras. A ideia é desativá-las apenas em casos de conflito entre caixas provenientes de diferentes esteiras. A especificação apresentada na Figura 10 permite que o fluxo da linha se mantenha até que uma caixa com maior prioridade, vinda do decorrer da esteira central requisite a parada da linha.

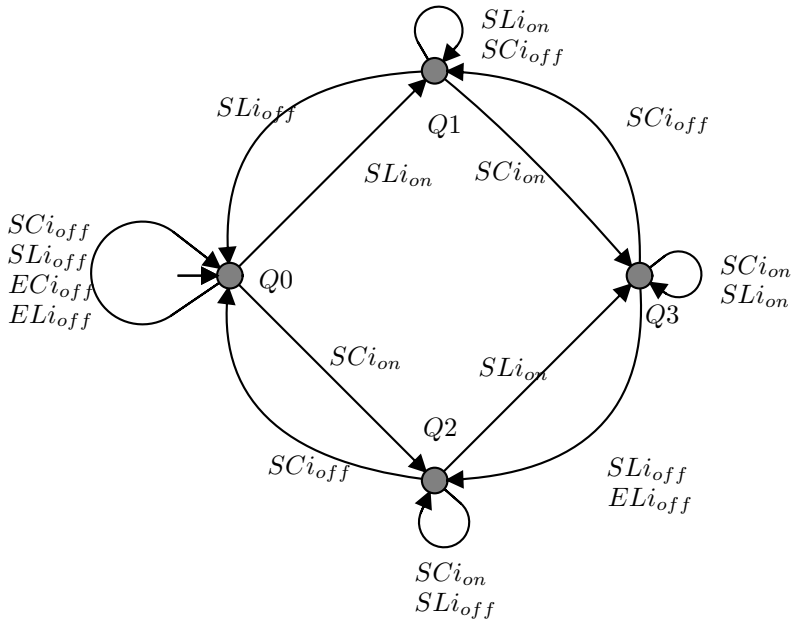


Figura 10: Especificação genérica para Linha e Esteira central
Fonte: Autor (2017)

Na TCS, uma especificação é uma regra, modelada por meio de uma máquina de estados que, quando associado ao modelo da planta, define uma ação proibitiva sobre a planta. Por exemplo, após a composição da planta de uma esteira de linha com uma esteira central, a especificação terá a função desabilitar eventos, todos e somente, aqueles que levariam as esteiras a colidirem duas caixas.

No modelo da Figura 10, os eventos SC e SL são os sensores da esteira central e de linha, respectivamente, e os eventos EC e EL são referentes às esteiras. A especificação tem sua ação de controle no estado Q_3. Para alcançar Q_3, necessita-se dos eventos (SCi_{off} e SCi_{on}) em qualquer ordem. No estado $Q3$ só é permitida a

máquina de estados continuar a identificar os sensores e desligar a esteira da linha (ELi_{off}). O evento desabilitado é o de desligamento da esteira central, atendo ao requisito de prioridade para este fluxo. De forma sucinta, a especificação que evolve linha e esteira central força o desligamento da primeira quando são identificadas caixas nos dois sensores, evitando assim a colisão e, caso exista detecção em apenas uma dos sensores, o fluxo não é alterado.

3.2.3 ESCOAMENTO PARA EXPEDIÇÃO

Considerando a impossibilidade de parada da esteira central e a disputa pelo recurso na esteira de expedição, o que obrigaria a alternar o bloqueio de escoamento nas duas direções, tornou-se inviável uma solução considerando cada esteira central como apenas um elemento. Para tal problema, a modelagem deu-se em considerar cada esteira central de forma fracionada. Primeiramente a parte que dá vazão as linhas de montagem e, em sequência, a parte que escoa os produtos para a esteira de expedição. A motivação para essa abordagem aplica-se em ainda utilizar o mesmo modelo visto no autômato da Figura 8 para as esteiras esquerda e direita dessa seção.

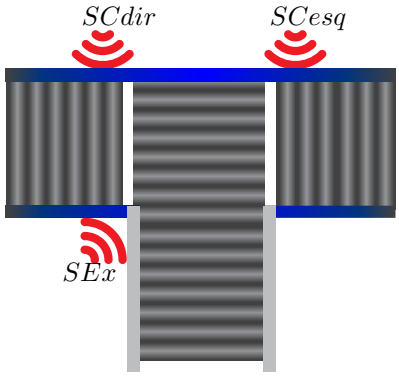


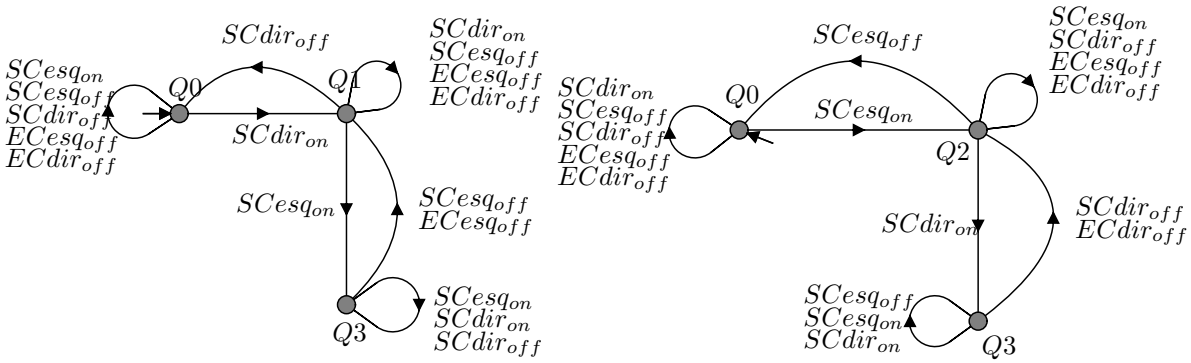
Figura 11: Modelo para região de conflito entre esteiras centrais e de expedição
Fonte: Autor (2017)

3.2.3.1 EXCLUSÃO MÚTUA

O recurso compartilhado na Figura 11 é o início da esteira com laterais em cinza, que agora representa a etapa final de deslocamento dos produtos, das esteiras centrais (azuis) para a expedição (1 única esteira). Esta disputa pelo recurso precisa garantir uma utilização equilibrada e ótima, tornando o fluxo contínuo quando não há requisição de um dos lados (centrais) e garantindo a ordem de entrega do recurso quando há requisição simultânea. Como solução, a modelagem dessa etapa de problema foi composta por 3 esteiras e 3 sensores, dois de chegada para as estei-

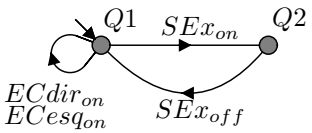
ras centrais (SCdir e SCesq) e um de liberação da área compartilhada, representado como SEx.

O controlador construído para essa região conta com 3 especificações apresentadas na Figura 12, para restringir o funcionamento da esteira esquerda (12b) e direita (12a) e o terceiro que compõe as duas esteiras e o sensor da expedição (12c). Como a abordagem é via CML, foram gerados 3 supervisores.



(a) Especificação para esteira direita

(b) Especificação para esteira esquerda



(c) Especificação para esteira expedição

Figura 12: Especificações para exclusão mútua na esteira de expedição
Fonte: Autor (2017)

O funcionamento das especificações apresentadas na Figura 12 nos itens (a) e (b) são complementares. Apesar de possuírem a mesma lógica de funcionamento, uma opera sobre a esteira esquerda e outra sobre a direita. A restrição que compõe esses modelos é em relação à ordem chegada de produto ao fim da esteira. O lado que detectar a chegada antes, terá prioridade na utilização do espaço concorrente na esteira de expedição. A cada vez que essa esteira é ocupada, o sensor *SEx* detecta a caixa e bloqueia o acionamento das duas esteiras centrais, isso ocorre para evitar a colisão de caixas na região de transição das esteiras, já que as especificações de exclusão mútua não contemplam o tempo de transição da caixa na região compartilhada.

3.2.3.2 ESTEIRA EXPEDIÇÃO

Como não houve nenhuma ação de controle atrelada à essa esteira, sua modelagem não foi necessária, por isso, considerou-se que ela sempre estará ligada. Caso ocorra sua parada, o sensor SEx, apresentado na Figura 11, irá bloquear o funcionamento das esteiras centrais, evitando possíveis problemas e priorizando na totalidade a expedição, que de fato concentra o maior fluxo entre todas às esteiras. O modelo deste sensor é apresentado na Figura 13

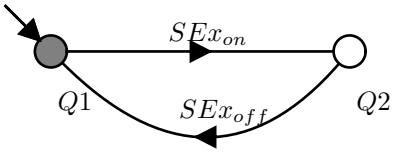


Figura 13: Modelo para sensor da esteira de expedição
Fonte: Autor (2017)

3.2.4 SÍNTESE MODULAR DE CONTROLE

A síntese modular foi obtida pela análise da relação de cada uma das especificações com todas as plantas. A Tabela 1 apresenta essa relação. Cada coluna denota o supervisor obtido, da esquerda para a direita são: relação da esteira central com a linha; expedição e esteiras centrais; exclusão mútua para esteira central esquerda; e exclusão mútua para esteira central direita.

Tabela 1: Relação de plantas e especificações para gerar os supervisores
Fonte: Autor (2017)

		Especificações		
	Linha e Central	**Expedição**	**Exc_Esquerda**	**Exc_Direita**
Linha	x			
Central Linha	x			
Central Direita		x	x	x
Central Esquerda		x	x	x
Sensor Expedição		x		

Os dados dos 4 supervisores obtidos são apresentados na Tabela 2, em que (|Q|) denota o número de estados, (|Σ|) denota o tamanho do alfabeto e ($\rightarrow$) expressa a quantidade de transições. A última linha apresenta um supervisor equivalente, caso fosse obtido sem usar o CML, para efeitos comparativos. O ganho, em termos de tempo, na implementação é visível, pois frente as 13332 transições sem o CML, foram implementadas apenas 211. Já o esforço em termos de memória pode ser estimado pela quantidade de estados, sendo que o pior caso modular implementa

18 estados, enquanto que o supervisor monolítico implementa 1674.

Tabela 2: Número de estados, tamanho do alfabeto e quantidade de transições dos Supervisores
Fonte: Autor(2017)

Supervisores	$\|Q\|$	$\|\Sigma\|$	$\|\rightarrow\|$
1 - Expedição	18	10	72
2 - Exclusão mútua Esquerda	18	8	58
3 - Exclusão mútua Direita	18	8	58
4 - Central e Linha	9	8	23
5 - Todas as plantas e especificações	1674	22	13332

Abaixo, na Figura 14, é apresentada a simulação de uma das etapas de validação do modelo proposto. As plantas e as especificações vistas na imagem representam o supervisor da área de expedição, visto na Figura 11. A sequência de eventos proposta para o teste é apresentada na lateral esquerda da Figura 14, onde as duas esteiras são ligadas e, também nas duas, o sensor dispara o evento. Nas especificações *E_Dir_Exp* e *E_Esq_Exp* é possível ver a evolução causada pelo último evento (em rosa). Com a sequência apresentada o produto chegou primeiramente no lado direito, portanto o sistema apresenta como evolução possível apenas os eventos em verde, e, nesse estado, o único evento controlável habilitado é desligar a esteira esquerda.

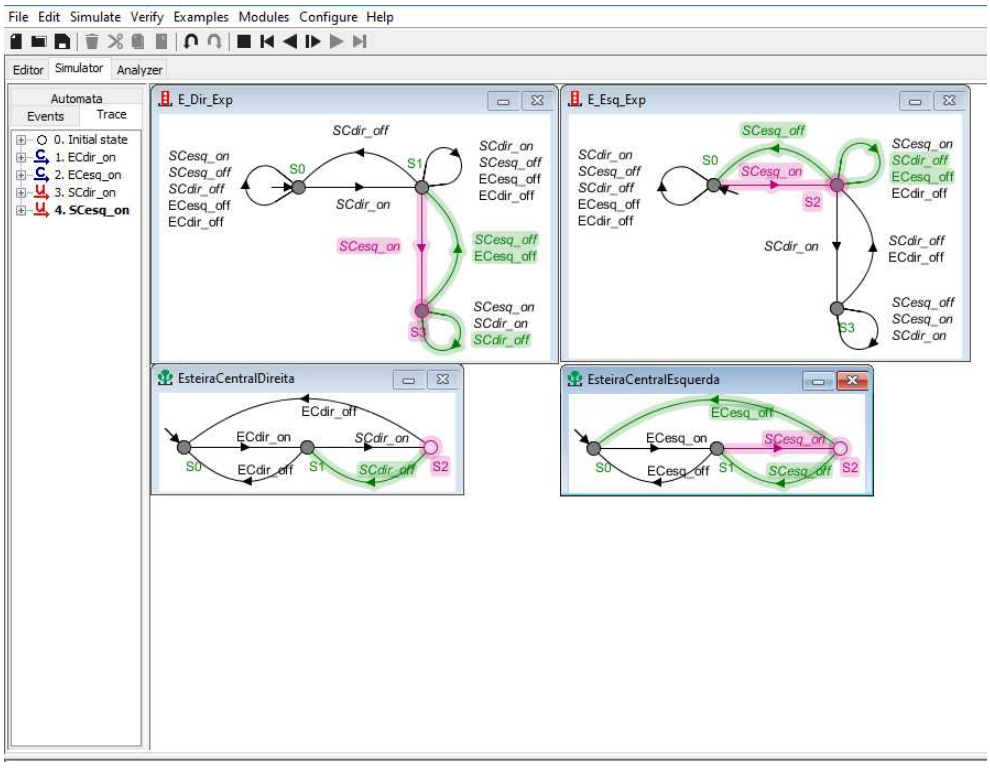


Figura 14: Simulação demonstrativa de sequência de eventos
Fonte: Autor (2017)

Todos os demais requisitos podem ser conferidos via simulação, executando o *software* Supermica, que está disponível junto aos modelos e implementação no github do autor, disponível no link https://github.com/bruno-manoel-dbki/TCC.

3.3 IMPLEMENTAÇÃO

Nesta seção é desenvolvido a implementação sobre um modelo genérico de máquina de estado baseado em ponteiro de funções. Este modelo é adaptado a necessidade deste trabalho. Como não havia uma plataforma de teste onde os sensores e estado dos atuadores pudessem ser lidos, optou-se pela inserção manual de dados para sistema, desta forma a utilização das funções *scanf* e *printf* da biblioteca *stdio* foi necessária. A implementação está disponível no *github* na conta do autor, e pode ser acessada pelo link https://github.com/bruno-manoel-dbki/TCC.

3.3.1 MÁQUINA DE ESTADOS GENÉRICA

Os possíveis dispositivos à serem instalados em meio industrial são variados, desde microcontroladores de baixo custo como MSP430 da Texas Intruments, assim como SOCs (System On a Chip) que podem executar programas de laço infinito ou sistemas operacionais embarcados, como chips da arquitetura ARM ou ainda *Single Board Computers* como Raspberry. Com essa possibilidade viu-se a necessidade do desenvolvimento de um código enxuto, com pouco uso de memória, visando o uso de microcontroladores, mas também não gerar um código atrelado a apenas um hardware, possibilitando a substituição da tecnologia utilizada, logo um modelo genérico que necessite apenas de adequações para entrada e saída dos sensores e atuadores.

A implementação das máquinas de Moore deu-se na linguagem C, tomando por base uma máquina genérica que opera com ponteiro de funções, em que cada estado é uma função. A cada informação enviada para a máquina, ela acessa a função do estado atual, executando a respectiva tarefa e, caso necessário, evolui ao próximo estado. Isso é feito alterando o ponteiro da variável do estado atual. Cada etapa do desenvolvimento dessa máquina será explicado a seguir.

Na função principal do programa, apresentada no Trecho de Código 1, há apenas a chamada da inicialização da máquina que tem como parâmetro a função *Handler* moldada com o tipo de cada uma das máquinas. Desta forma, caso exista a necessidade do retorno de alguma informação da máquina, utilização a função *Handler* que será executada automaticamente pela máquina em um determinado estado,

já em laço infinito, a execução da ação para cada dado recebido. Apesar do Trecho de código refalg:main apresentar a execução das máquinas em laço infinito, isso não deve ocorrer na construção de um sistema real. Essa forma é utilizada para que os dados sejam passados manualmente para a máquina. Em caso de aplicação, sugere-se que a chamada de função ocorra em um *Callback*.

```
1  int main (void)
2  {
3      initSM1((Handle_sm1_t)Handler);
4      initSM2((Handle_sm2_t)Handler);
5      while(1)
6      {
7          ExecSM1(data);
8          ExecSM2(data);
9      }
10     return 0;
11 }
```

Trecho de Código 1: Função principal do modelo genérico de máquina de estados genérica

As definições de tipo necessárias para a máquina são apresentadas no Trecho de Código 2. Para melhor organização e legibilidade, podem ser feitas definições para o nome dos eventos, vistas também nesse Trecho de Código.

```
1  #ifndef SRC_SM_H_
2  #define SRC_SM_H_
3
4  #define event_0         0
5  #define event_1         1
6  #define event_2         2
7  #define event_3         3
8  #define event_4         4
9  #define event_5         5
10 #define event_6         6
11 #define event_7         7
12
13 typedef void (*Handle_sm1_t)(unsigned char *data);
14 typedef int (*Action_sm1_t) (unsigned char data);
15 int ExecSM1( unsigned char data);
16 void initSM1(Handle_lin_t handle);
17
18 typedef void (*Handle_sm2_t)(unsigned char *data);
19 typedef int (*Action_sm2_t) (unsigned char data);
```

```
20 int ExecSM2( unsigned char data);
21 void initSM2(Handle_lin_t handle);
22
23 #endif /* SRC_SM_H_ */
```

Trecho de Código 2: Arquivo de cabeçalho para declaração de eventos e tipos para máquina de estados genérica

A construção da máquina propriamente dita acontece no Trecho de código 3 em que existem 2 etapas, de (i) inicialização e execução, onde a máquina é estruturada e (ii) declaração das funções, para definir os estados e as transições.

Inicialmente é declarado um tipo enumerável com todos os estados da máquina adicionada de um estado final, que tem a função de apenas possibilitar a declaração de tamanho do vetor de ações. Na sequência, uma estrutura de dados que deve essencialmente conter o estado, também um ponteiro da função *handle* e um vetor ponteiro de funções do tipo *Action*, previamente definido. Aqui ainda pode ser adicionada qualquer variável necessária para o controle da máquina. Importante observar que essas declarações devem ficar em um arquivo que contenha apenas uma máquina, pois caso exista dois tipos enumeráveis, que aqui mascaram os estados, haverá conflito. A função de inicialização deve atribuir à variável estado o vértice inicial do autômato, além disso, deve também atribuir ao vetor de ponteiros de função cada uma das funções referente aos estados do autômato. Por fim a declaração da função de execução da máquina ocorre com a chamada da função *Action* na posição do estado atual. A requisição dessa função ocorrerá no código principal do programa e seus parâmetros podem ser estipulados independente da daqueles que serão passados pra evolução da máquina. Aqui também os dados podem ser manipulados como necessário.

```
1  typedef enum{
2      ST_0
3      ST_1,
4      ST_2,
5      ST_FINAL
6  }states_t;
7
8  struct{
9      states_t state;
10     Handle_sm_t handle;
11     Action_sm_t Action[ST_FINAL];
12 }sm;
13 void initSM(Handle_lin_t handle)
```

```
14 {
15     sm.Action[ST_0] = fn_ST_0;
16     sm.Action[ST_1] = fn_ST_1;
17     sm.Action[ST_2] = fn_ST_2;
18     sm.state = ST_0;
19 }
20
21 int ExecSM(unsigned char data)
22 {
23     sm.Action[sm.state](data);
24 }
```

Trecho de Código 3: Modelo de função da máquina de estados genérica

A segunda etapa é a declaração das funções, que devem ter seu cabeçalho condizente com o definido no tipo *Action* do Trecho de Código 3, permitindo a atribuição na função de inicialização. Dentro dessa função haverá o que condiz com a execução do estado. Sugere-se que a declaração seja estática, afim de não permitir acessos indevidos. O modelo dessa declaração é apresentado no Trecho de Código 4 que deve ser replicado para todos os estados declarados

```
1 static int fn_ST_0(unsigned char data)
2 {
3     /*
4     Coisas a serem feitas no estado
5     */
6 }
```

Trecho de Código 4: A estutruta básica da função/estado para este trabalho

3.3.2 MÁQUINAS CONCORRENTES

Era requisito de implementação executar todos os 4 autômatos. A solução poderia tratar cada máquina como um programa a ser executado, porém esse tipo de concorrência de tarefas necessita de um sistema operacional que possa escaloná-las, restringindo o grupo de dispositivos aplicáveis. O modelo genérico utilizado executa em laço infinito, porém, em uma implementação real isso não deverá acontecer. A forma mais indicada contempla as chamadas de execução em *Callbacks* do sistema. Sendo assim, a evolução de estados na máquina ocorrerá em funções, logo o processador é liberado após cada transição ocorrida, desta forma é possível garantir que uma quantidade qualquer de máquinas totalmente diferentes ou que tenham correlação por

seus eventos, ou ainda, que dependam da informação gerada por outra parte de um sistema funcionem de forma sequencial. Essa característica também permite a economia de energia, pois quando não há envio de dados para a máquina, o dispositivo não irá consumir processamento, possibilitando a entrada em modos de economia.

3.3.3 ADAPTAÇÃO AO MODELO DESTE TRABALHO

A utilização do software Supremica (AKESSON, 2013) no desenvolvimento dos modelos, permite uma exportação em arquivos XML, contendo todas as informações da máquina de estados. Para este trabalho foram exportadas os dados de todos os supervisores. Extraindo as informações de eventos, estados e as transições, foi possível construir a máquina de estados para cada um dos supervisores. Como as funções de todos os estados tem a mesma estrutura, foi possível utilizar um desenvolvimento padronizado.

```
1  static int ST_n(unsigned char data, int e)
2  {
3      if(e)
4      {
5          if(data == event_1 || data == event_i)
6              return OK;//SE FOR EVENTO HABILITADO
7          else if(data>= event_0 && data<event_final)
8              return NOK;// SE NO ALFABETO E NAO HABILITADO
9          return OUT; //SE FORA DO ALFABETO
10     }
11     else
12     {
13         if( data == event_1)
14             sm.state = state_i;
15         else if( data == event_i)
16             sm.state = state_1;
17     }
18     return sm.state;
19 }
```

Trecho de Código 5: Função modelo adaptada para os estados dos supervisores

No Trecho de Código 5 nota-se os parâmetros dados a cada estado, onde o caractere *data* é o evento ocorrido e o inteiro *e* é uma variável de controle que caso seja 1, irá verificar se o evento está habilitado neste estado. No Trecho de Código isso é denotado pela condição da linha 3 que, quando satisfeita, passa a verificação se o evento está habilitado neste estado. Isso é visto na linha 5 com *event_1* e *event_i*

como exemplos. Para a condição satisfeita, retorna-se 1, neste caso *OK*. Se a condição não for satisfeita, testa-se se o evento pertence ao alfabeto daquele supervisor, para então retornar -1, definido como *NOK*. Se nenhuma das condições é satisfeita, o evento não irá operar sobre este supervisor, pois não pertence ao seu alfabeto. Isso é necessário para garantir a condição explicitada no Capitulo 2.6 de que um evento só pode ocorrer caso todos os supervisores o tenham habilitado naquele estado. Para a condição da variável *e* não ser 1, ocorrerá o tratamento do evento recebido. Caso ele satisfaça uma das condições deste estado, que devem ser as mesmas verificadas na condição da linha 6, a máquina evoluirá para um próximo estado. Por fim essa função retorna a informação atualizada sobre a posição da máquina. As funções-estado de todas as máquinas obedecem o padrão proposto neste modelo.

Na função principal do programa foram necessárias algumas adaptações em relação ao modelo inicial da máquina genérica. Iniciando pela necessidade de verificar se o evento é permitido por todos os supervisores, fez-se uma verificação desta condição para só então executar a transição de todas as máquinas com tal evento. Essa condição é mostrada no Trecho de Código 6 já com a implementação final.

```
1 if( ExecSM_Linha(data1, 1) > NOK && ExecSM_Conflito_Dir(data1, 1) >
     NOK  && ExecSM_Conflito_Esq(data1, 1) > NOK &&  ExecSM_Expedicao(
     data1,1) > NOK)
2 {
3     printf("Linha 1 -\t%d\n", ExecSM_Linha(data1,0));
4     printf("Conflito Dir -\t%d \n", ExecSM_Conflito_Dir(data1,0));
5     printf("Conflito Esq -\t%d \n", ExecSM_Conflito_Esq(data1,0));
6     printf("Expedicao -\t%d \n", ExecSM_Expedicao(data1,0));
7 }
```

Trecho de Código 6: Implementação da condição de evento habilitado em todos os supervisores

Neste trabalho, optou-se pela apresentação dos dados na tela com a função *printf* apenas por fins de simulação. Além disso, a entrada de dados, no caso os sensores e atuadores foram simulados utilizando a entrada padrão de dados com a função *scanf*.

3.3.4 GERADOR DE EVENTOS CONTROLÁVEIS

À nível de desenvolvimento prático, percebeu-se a necessidade de verificação do estado de cada atuador após a evolução para um novo estado, para que, quando um dos eventos habilitados no estado forem controláveis, estes devem evoluir

diretamente caso estejam ativados. Por exemplo, caso uma esteira esteja ligada e a máquina de estados evolua à um estado que habilite o evento *ligar_esteira* deve-se executar essa transição imediatamente. Como não houve implementação em nível de simulação física onde fosse possível fazer a leitura desse estado da esteira, decidiu-se pela criação de um gerador para esses eventos controláveis, tomando por base o gerador de códigos da ferramenta DESLAB (TORRICO, 2015). Porém, visando uma execução de simulação apenas como suporte, preferiu-se criar um processamento paralelo com *threads*. A função que executa em uma segunda *thread* apenas gera um número pseudo-aleatório n entre 0 e 4. Esse valor está contido no alfabeto do supervisor das linhas, logo não opera sobre os supervisores das outras etapas, por isso, em uma segunda variável, um segundo número é gerado entre 12 e 16. Isso permite executar apenas um teste condicional sobre o evento estar habilitado ou não. A utilização de uma computação concorrente gera a necessidade de proteção da região crítica, neste caso as máquinas de estado. Portanto, a utilização de um *mutex* foi necessária para evitar a interrupção de execução da *thread* principal pela secundária ou vice-versa. Assim como neste trecho de código, o bloqueio do *mutex* também foi adicionado na execução da função principal.

```
1  void *inc_x(void *x_void_ptr)
2  {
3      while(1)
4      {
5          int rand_num = rand()%4;
6          int rand_num2 = rand_num+8;
7          //printf("\n %d - %d \n", rand_num, rand_num2);
8          pthread_mutex_lock(&mtx);
9
10         if( ExecSM_Linha(rand_num, 1) > NOK && ExecSM_Conflito_Dir(
               rand_num2, 1) > NOK  && ExecSM_Conflito_Esq(rand_num2, 1)
                > NOK &&  ExecSM_Expedicao(rand_num2,1) > NOK)
11         {
12                 printf("INPUT: Random\n");
13                 printf("Linha 1 -\t%d\n", ExecSM_Linha(rand_num,0));
14                 printf("Conflito Dir -\t%d \n", ExecSM_Conflito_Dir(
                      rand_num2,0));
15                 printf("Conflito Esq -\t%d \n", ExecSM_Conflito_Esq(
                      rand_num2,0));
16                 printf("Expedicao -\t%d \n", ExecSM_Expedicao(
                      rand_num2,0));
17
18         }
```

```
19          pthread_mutex_unlock(&mtx);
20          sleep(1);
21      }
22      return NULL;
23  }
```

Trecho de Código 7: Implementação da função de threads para gerar eventos aleatórios

Sintetizando a implementação ocorreu sobre 4 supervisores. (i) O supervisor de apenas uma linha de produção com uma esteira central, visto a escalabilidade já citada;(ii) e (iii) Sobre às esteiras que escoam da região central para a expedição, um para o lado direito e um para o esquerdo; e (iv) Para o fluxo da expedição de produtos. Para cada máquina foi gerado um arquivo da linguagem C e todos se referenciam a um arquivo de cabeçalho único. A execução paralela é eficiente e independe de dispositivo, utilizando apenas funcionalidades padrões da linguagem. As bibliotecas *pthreads* e *stdio* e *stdlib* são utilizadas apenas pela execução no ambiente de testes, que não conta com os sensores, nem os atuadores. Foram aproximadamente 2250 linhas de código que compiladas com GCC versão 4.9.2 em sistema operacional Debian Jessie geraram um arquivo de 25KB portável a qualquer dispositivo que compile a linguagem e o padrão POSIX, para *threads*. Um código pequeno em relação a implementação de 4 máquinas de estados.

4 CONCLUSÃO

Nesse trabalho foi analisada a linha de produção da Atlas Eletrodomésticos com o objetivo de propor um melhor o escoamento da produção. Inicialmente o comportamento operacional das linhas foi analisado e dessa análise foram propostos modelos em máquinas de estados que mapeiam o comportamento para estruturas formais. Então foram propostas especificações de controle para ajustar esse comportamento dentro de um padrão esperado na prática. Por fim, a etapa de síntese obteve um conjunto de controladores modulares, que foram implementados usando a metodologia de máquina de estados com ponteiros de função na linguagem C. Como perspectivas futuras estima-se que a empresa vislumbre o ganho de processo obtido com a proposta deste trabalho frente ao controle sub-ótimo utilizado atualmente.

Dado o bom desempenho da implementação genérica de máquinas de estados obtida neste trabalho, é possível estender este modelo para um gerador automático do código a ser implementado. Este programa poderá automatizar a geração, garantindo maior confiabilidade e inibindo erros inerentes a implementação manual.

REFERÊNCIAS

AKESSON, K et al. **Supremica**. [S.l.], 2013. Disponível em: <http://www.supremica.org/>.

ATLAS, INDÚSTRIA DE ELETRODOMÉSTICO. **Sobre a Atlas**. 2016. ATLAS. Disponível em: <http://www.atlas.ind.br/pt/sobre>.

CAI, Kai; WONHAM, W. Murray. Supervisor localization: A top-down approach to distributed control of discrete-event systems. **IEEE Transactions Automatic Control**, v. 55, n. 3, p. 605–618, 2010.

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to Discrete Event Systems**. 2 ed. NY: Springer Science, 2008.

CHEN, Y. L.; LAFORTUNE, S.; LIN, F. Modular supervisory control with priorities for discrete event systems. **34th IEEE Conference on Decision and Control**, p. 409–415, 1995.

CURY, J E R. Teoria de controle supervisório de sistemas a eventos discretos. **V Simpósio Brasileiro de Automação Inteligente**, 2001.

FLORDAL, Hugo; MALIK, Robi. Compositional verification in supervisory control. **SIAMJCO**, v. 48, n. 3, p. 1914–1938, 2009.

GUTIERREZ, Regina Maria Vinhais; PAN, Simon Shi Koo. Complexo eletrônico: automação do controle industrial. **BNDES Setorial, Rio de Janeiro**, n. 28, p. 189–231, 2008.

HOPCROFT, J. E.; ULLMAN, J. D.; MOTWANI, R. **Introduction to Automata Theory, Languages and Computation**. 2 ed. [S.l.]: Addison Wesley, 2001.

LEWIS, Harry R; PAPADIMITRIOU, Christos H. **Elements of the Theory of Computation**. [S.l.]: Prentice Hall PTR, 1997.

MENEZES, Paulo Blauth. **Linguagens formais e autômatos**. [S.l.]: Sagra-Dcluzzato, 1998.

MOHAJERANI, S. et al. Compositional synthesis of discrete event systems using synthesis abstraction. In: **Chinese Control and Decision Conference (CCDC'11)**. [S.l.: s.n.], 2011. p. 1549–1554.

MONTGOMERY, E. **Introdução aos Sistemas a Eventos Discretos e à Teoria do Controle Supervisório**. 1 ed. [S.l.]: Alta Books, 2004.

NISE, Norman S. **CONTROL SYSTEMS ENGINEERING, (With CD)**. [S.l.]: John Wiley & Sons, 2007.

OGATA, Katsuhiko; YANG, Yanjuan. Modern control engineering. Prentice-Hall Englewood Cliffs, 1970.

PENA, P. N.; CURY, J. E. R.; LAFORTUNE, S. Verification of nonconflict of supervisors using abstractions. **IEEE Transactions on Automatic Control**, v. 54, n. 12, p. 2803–2815, 2009.

QUEIROZ, Max H. De; CURY, J. E. R. Modular control of composed systems. In: **American Control Conference**. Illinois, USA: [s.n.], 2000. v. 6, p. 4051–4055.

QUEIROZ, Max H. De; CURY, Jose E. R. Modular supervisory control of large scale discrete event systems. In: **Discrete Event Systems: Analysis and Control. Proc. WODES'00**. [S.l.]: Kluwer Academic, 2000. p. 103–110.

RAMADGE, P J; WONHAM, W M. Modular feedback logic for discrete event systems. **SIAM Journal of Control and Optimization**, v. 25, n. 5, p. 1202–1218, 1987.

RAMADGE, P J; WONHAM, W M. Supervisory control of a class of discrete event process. **SIAM Journal of Control and Optimization**, v. 25, n. 1, p. 206–230, 1987.

RAMADGE, P J; WONHAM, W M. The control of discrete event systems. **IEEE Special Issue on Discrete Event Dynamic Systems**, IEEE, v. 77, p. 81–98, 1989.

SANTOS, JÉSSICA KS et al. Simulação de sistemas a eventos discretos sob o controle supervisório modular local. 2013.

TEIXEIRA, M. **Explorando o uso de distinguidores e de Autômatos Finitos Estendidos na Teoria do Controle Supervisório de Sistemas a Eventos Discretos**. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2013.

TEIXEIRA, M. et al. Variable abstraction and approximations in supervisory control synthesis. **2013 American Control Conference, ACC'13**, Washington, USA, p. 120–125, Jun. 2013.

TORRICO, César Claure. **Controle Discreto**. 2015. DESLAB - Ferramenta para autômatos finitos. Disponível em: <https://sites.google.com/site/controlediscreto9/instaladores>.

WONG, K C; WONHAM, W M. Modular control and coordination of discrete event systems. **Discrete Event Dynamic Systems**, v. 8, n. 3, p. 241–273, 1998.

WONHAM, W.; RAMADGE, P. Modular supervisory control of discrete-event systems. **Mathematics of Control, Signals, and Systems (MCSS)**, Springer London, v. 1, p. 13–30, 1988.