UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO - CTC

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Rodrigo Szpak

Controle Supervisório de Sistemas de Manufatura sob Incertezas de Processamento e Restrições Temporais: Modelagem, Síntese e Implementação

Florianópolis

2021

Rodrigo Szpak

Controle Supervisório de Sistemas de Manufatura sob Incertezas de Processamento e Restrições Temporais: Modelagem, Síntese e Implementação

Proposta de Tese submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do Grau de Doutor em Engenharia de Automação e Sistemas.
Orientador: Prof. Max Hering de Queiroz, Dr.
Coorientador: Prof. José Eduardo Ribeiro Cury, Dr.

Florianópolis

2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Szpak, Rodrigo
Controle Supervisório de Sistemas de Manufatura sob Incertezas de Processamento e Restrições Temporais: Modelagem, Síntese e Implementação / Rodrigo Szpak ; orientador, Max Hering de Queiroz, coorientador, José Eduardo Ribeiro Cury, 2022.
165 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2022.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Sistemas a Eventos Discretos Temporizados. 3. Teoria de Controle Supervisório Temporizado. 4. Controle Modular Local Temporizado. 5. Controlador Lógico Programável. I. Queiroz, Max Hering de. II. Cury, José Eduardo Ribeiro. III. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Engenharia de Automação e Sistemas. IV. Título.

Rodrigo Szpak

Controle Supervisório de Sistemas de Manufatura sob Incertezas de Processamento e Restrições Temporais: Modelagem, Síntese e Implementação

O presente trabalho em nível de doutorado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Marcos Vicente de Brito Moreira, Dr.
Universidade Federal do Rio de Janeiro - UFRJ

Prof. André Bittencourt Leal, Dr.
Universidade do Estado de Santa Catarina - UDESC

Prof. Marcelo Ricardo Stemmer, Dr.
Universidade Federal de Santa Catarina - UFSC

Prof. Felipe Gomes de Oliveira Cabral, Dr.
Universidade Federal de Santa Catarina - UFSC

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Doutor em Engenharia de Automação e Sistemas.

Documento assinado digitalmente
Werner Kraus Junior
Data: 09/02/2022 12:04:00-0300
CPF: 531.085.239-53
Verifique as assinaturas em https://v.ufsc.br

Coordenação do Programa de
Pós-Graduação

Documento assinado digitalmente
Max Hering de Queiroz
Data: 09/02/2022 08:07:31-0300
CPF: 912.550.079-15
Verifique as assinaturas em https://v.ufsc.br

Prof. Max Hering de Queiroz, Dr.
Orientador

Florianópolis, 2021.

Com todo o meu amor, dedico este trabalho a Deus, aos meus pais Maria e Pedro (in memorian), a minha filha Isabella e a minha esposa Fernanda.

AGRADECIMENTOS

Aos Professores Max e Cury, agradeço profundamente pela compreensão durante essa etapa da minha vida. Agradeço, também, o comprometimento com este trabalho e os incontáveis e inestimáveis ensinamentos, contribuindo com ideias e críticas construtivas, além da primorosa revisão do presente documento.

RESUMO

Controladores lógicos programáveis (CLPs) são os principais dispositivos utilizados para o controle dos sistemas automatizados na indústria de manufatura. Portanto, a solução de um problema de automação em sistemas de manufatura passa pela obtenção de uma lógica de controle a ser implementada em CLPs. O projeto de uma lógica de controle para sistemas flexíveis com intervenções humanas e requisitos temporais é uma tarefa complexa, que justifica o uso de métodos formais, como a Teoria de Controle Supervisório (TCS) para Sistemas a Eventos Discretos Temporizados (SEDTs). Modelos temporizados permitem resolver problemas mais complexos, porém, ao custo de que os modelos gerados também sejam mais complexos. Esta Tese apresenta um método para modelagem (discretização dos intervalos de tempos contínuos) de sistemas de manufatura com tempos de processamentos incertos e restrições de tempo. Explora, também, a teoria do controle supervisório para sintetizar uma lógica de controle responsivo, que garante segurança e restrições de tempo em sistemas de manufatura, decorrentes da intervenção humana direta e operações variáveis. Apresenta-se, ainda, a abordagem modular para calcular um conjunto de supervisores temporizados que forçam as operações da máquina ou inibem as intervenções humanas, de uma forma não bloqueante e minimamente restritiva. A proposta de uma nova arquitetura para implementação estruturada de supervisores modulares temporizados em CLP, um código compatível com a norma internacional IEC61131-3 e uma análise sobre os problemas relativos à implementação de SEDT em CLP são apresentados. Cabe ressaltar que esses métodos foram aplicados a sistemas de manufatura do mundo real. Analisa-se o impacto da granularidade do período do relógio digital na complexidade dos modelos de síntese, no tamanho do código do CLP e na eficiência do sistema em malha fechada. O método de modelagem de SEDT proposto permitiu a discretização de forma minimante conservadora, dos intervalos de tempos contínuos para sistemas de manufatura com tempos de processamentos incertos. Com a utilização da arquitetura temporizada proposta, a implementação do controle modular local temporizado (CMLT) no Sistema Modular de Produção (MPS) resultou em um código estruturado de fácil interpretação, possibilitando a alteração do código (incluindo ou excluindo supervisores e subsistemas), e a implementação na memória do CLP.

Palavras-chaves: Sistemas a Eventos Discretos Temporizados. Teoria de Controle Supervisório Temporizado. Controlador Lógico Programável. Controle Modular Local Temporizado. Sistemas de Manufatura. Implementação.

ABSTRACT

Programmable Logic Controllers (PLCs) are the main devices used to control automated systems in the manufacturing industry. Therefore, the solution to an automation problem in manufacturing systems involves obtaining a control logic to be implemented in PLCs. The design of a control logic for flexible systems with human interventions and timing requirements is a complex task, which justifies the use of formal methods, such as the Supervisory Control Theory (SCT) for Timed Discrete Event Systems (TDESs). Timed models allow you to solve more complex problems, but at the cost that the generated models are also more complex. This Dissertation presents a method for modeling (continuous time interval discretization) of manufacturing systems with uncertain processing times and time constraints. It also explores the theory of supervisory control to synthesize a responsive control logic, which guarantees safety and time constraints in manufacturing systems, resulting from direct human intervention and variable operations. It also presents a modular approach to calculate a set of timed supervisors that force machine operations or inhibit human interventions, in a non-blocking and minimally restrictive way. The proposal of a new architecture for structured implementation of timed modular supervisors in PLC, a code compatible with the international standard IEC61131-3 and an analysis of the problems related to the implementation of SEDT in PLC are presented. It is noteworthy that these methods have been were applied to real-world manufacturing systems. The impact of digital clock period granularity on the complexity of synthesis models, on the PLC code size and on the efficiency of the closed-loop system, is analyzed. The proposed TDES modeling method allowed the minimally conservative discretization of continuous time intervals for manufacturing systems with uncertain processing times. Using the proposed timed architecture, the implementation of the local modular control timed (LMCT) in the Modular Production System (MPS) resulted in a structured code that is easy to interpret, enabling the code change (including or excluding supervisors and subsystems), and the implementation in memory of the PLC.

Key-words: Timed Supervisory Control Theory. Programmable Logic Controllers. Manufacturing systems. Timed Discrete Event Systems. Timed local Modular Control.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A complexidade dos sistemas automatizados na indústria de manufatura se intensificou extensivamente nas últimas décadas devido à expansão da demanda de produção e de melhorias na produtividade, qualidade e flexibilidade. O controlador lógico programável (CLP) é o principal dispositivo utilizado para o controle desses sistemas. Logo, a solução de um problema de automação em sistemas de manufatura passa pela obtenção de uma lógica de controle a ser implementada no mesmo. Na grande maioria das aplicações industriais a obtenção dessa lógica é feita sem um procedimento formal e, consequentemente, não se tem garantias do atendimento das especificações.

Um tema desafiador em sistemas de manufatura inteligentes é o controle em tempo real de sistemas de produção complexos que estão sujeitos a incertezas de processamentos, que podem ocorrer tanto pela intervenção humana direta quanto pelas variações nas durações dos processos (LAMNABHI-LAGARRIGUE *et al.*, 2017). A fim de garantir especificações de segurança e restrições de tempo, tais sistemas de controle devem ser capazes de responder em tempo hábil aos eventos decorrentes do comportamento humano não determinístico e operações variáveis da máquina. Quando os tempos de processamento são incertos devido à natureza dinâmica do sistema de manufatura, as abordagens convencionais, que assumem tempos de processamento constantes, podem não ser apropriadas para resolver problemas práticos. Por exemplo, quando uma restrição de intervalo de tempo restringe o tempo de espera máximo e mínimo entre duas operações variáveis consecutivas, a suposição de tempos de operação aproximados fixos pode levar a uma lógica de controle muito conservadora ou mesmo inviável. Além disso, uma estratégia de controle imperativa que impõe um comportamento rígido não é adequada para operadores humanos. Portanto, um controle reativo que responda com segurança e eficiência a eventos não determinísticos deve ser implementado.

Na literatura de modelagem de Sistemas a Eventos Discretos (SEDs) encontram-se diversas abordagens formais para a análise e validação sistemas de controle em CLP. Os métodos para validação classificam-se em modelagem e simulação, testes, verificação formal e *model checking* (GERGELY; COROIU; POPENTIU-VLADICESCU, 2011). A modelagem e simulação é baseada em um modelo que descreve o comportamento do sistema considerado. Esse modelo é, geralmente, executado por um software chamado simulador, que determina o comportamento do sistema em vários cenários. O teste é uma técnica operacional, pela qual pode ser verificado se uma implementação física do sistema cumpre os requisitos iniciais do projeto. Esses testes não sendo exaustivos podem mostrar apenas a presença de erros, porém não garantem sua au-

sência. Ao contrário dos testes, a verificação formal atua sobre modelos e não em implementações, fornecendo uma demonstração matemática da correção do problema. O *model checking* é uma técnica automatizada que, dado um modelo de estado finito e uma propriedade expressa em um formalismo lógico adequado, permite a verificação sistemática da propriedade. Dentre os principais métodos apresentados destacam-se os trabalhos de: Testes (THÉVENOD-FOSSE; WAESELYNCK; CROUZET, 1995); Verificação Formal (ANDERSON; TOURLAS, 1997), *Model Checking* (FARINES *et al.*, 2011; LJUNGKRANTZ *et al.*, 2010; MOKADEM *et al.*, 2010).

Existem, ainda, métodos formais de síntese, em que se representa o processo em malha aberta e as especificações através de modelos matemáticos, gerando uma lógica de controle correta por construção, a partir destes modelos. Dentre os principais métodos para síntese de controle de SED, destacam-se as Redes de Petri (HOLLOWAY; KROGH; GIUA, 1997; MOODY; ANTSAKLIS, 1998) e a Teoria de Controle Supervisório (RAMADGE; WONHAM, 1987).

As Redes de Petri (RP) fornecem uma técnica de modelagem visual onde fichas são utilizadas para simular atividades dinâmicas e concorrentes do sistema em estudo (MURATA, 1989). Essas redes possuem uma representação compacta tornando fácil visualizar o sistema modelado, possibilitam representar o tempo de forma compacta e, também, permitem a síntese baseada na estrutura em geral, sem precisar explicitar o espaço de estados. Porém, as Redes de Petri não permitem lidar com eventos não-controláveis, modelos temporizados e garantir a síntese com a máxima permissividade, de forma integrada. Além disso, os métodos de síntese baseados em RP são limitados, quando se levam em conta os eventos não-controláveis do processo.

Já Teoria de Controle Supervisório (TCS) iniciada por Ramadge e Wonham (1987) oferece um método formal baseado na teoria de autômatos e linguagens, para gerar um agente de controle chamado supervisor, que coordena o sistema de acordo com regras de controle, de forma que esses supervisores garantam especificações de segurança e não-bloqueio para o sistema de malha fechada, reagindo a eventos observados e interferindo na ocorrência de eventos de forma minimamente restritiva. A TCS tem a vantagem de permitir lidar com eventos não-controláveis, com modelos temporizados e garantir a síntese com a máxima permissividade, abordando assim, os três aspectos de forma unificada. Por esse motivo, a TCS será utilizada como base na proposta para esta pesquisa.

Esta Tese possui como temas centrais aspectos relativos à interação humana, incertezas de processamentos e restrições temporais. Para lidar com eventos gerados espontaneamente pelo processo, como na interação humana não determinística (onde o operador não é obrigado a realizar uma operação num tempo fixo - eventos são controláveis, mas não-forçáveis), o supervisor deve reagir a essas incertezas, tendo

um controle reativo. Essa é uma vantagem que justifica a utilização da TCS, já que outras abordagens não lidam tão bem com esse aspecto.

Com relação à questão de incertezas de processamentos, o principal ponto é que essa incerteza pode ser gerada pelos próprios tempos de processamentos das máquinas, assim ao invés de se lidar com um tempo fixo de uma duração, trabalha-se dentro de um intervalo mínimo e um máximo. Na literatura existem vários estudos que utilizam o escalonamento considerando o pior caso (maior tempo possível), porém, esse escalonamento se torna uma solução muito conservadora. Já com relação às restrições temporais muitas vezes essas dependem de eventos incertos, e a TCS temporizada resolve o problema, pois leva a uma solução minimamente restritiva.

Cabe salientar, também, que a TCS para SEDT torna-se viável, mesmo para os problemas em que não se tem restrições temporais, uma vez que é possível obter ganhos ao utilizar modelos mais detalhados que levam em conta os tempos de processamentos das máquinas, proporcionando maiores possibilidades de soluções, um controle eficiente e menos conservador.

O principal desafio da TCS é o fato de que a síntese de controladores para sistemas a eventos discretos é um procedimento computacional de complexidade polinomial, em função do número de estados dos modelos da planta e das especificações que, por sua vez, cresce exponencialmente com o aumento do número de componentes que integram o sistema. Essa limitação tem sido intensamente tratada na literatura para viabilizar a aplicação em problemas de grande porte, através da utilização de estruturas de controle modular e estruturas de dados otimizadas (CAI; WONHAM, 2010; CHEN; LIN, 2000; HOFFMANN; WONG-TOI, 1992). Segundo a abordagem de Controle Modular Local (CML) para autômatos não temporizados proposta por Queiroz e Cury (2002), o problema do crescimento exponencial dos modelos pode ser amenizado calculando-se um supervisor modular para cada especificação e levando em consideração, na síntese local, apenas os subsistemas afetados pela respectiva especificação. A abordagem de CML, além de diminuir a complexidade computacional da síntese, favorece a redução do tamanho dos supervisores. Essa característica costuma deixar a lógica de controle mais clara, compreensível e, portanto, mais confiável ao projetista, além de facilitar a implementação do sistema de controle.

Conforme Basile e Chiacchio (2007), a natureza assíncrona e indeterminada dos supervisores faz sua implementação ser uma tarefa complicada em equipamentos síncronos, como CLPs e PCs, pois alguns problemas podem trazer a falta de sincronismo entre a planta (sistema real) e o programa implementado no CLP. Segundo Fabian e Hellgren (1998), esses problemas de implementação para o caso não temporizado são divididos em quatro classes: sinais e eventos; causalidade; escolha; e sincronização inexata.

Cabe destacar, que esses problemas não são exclusivos da implementação baseada na TCS, mas são questões recorrentes na implementação de máquinas de estados, em que se assume que os eventos são sequenciais (que dois eventos não acontecem ao mesmo tempo). Esses problemas são recorrentes em qualquer implementação de SEDs em CLP, mesmo que seja por métodos não formais.

Fabian e Hellgren (1998) abordam o problema de implementação em CLPs criando mecanismos de conversão do supervisor descrito por uma máquina de estados finitos em *Ladder Diagram* (LD), segundo a norma IEC 61131-3. Hellgren, Lennartson e Fabian (2002) tratam do problema de implementação em CLP para a abordagem modular da TCS. Os autores utilizam modelos de Redes Petri (RP), pois usam a linguagem *Sequential Function Chart* (SFC) que fundamenta-se em RP. Assim, alguns problemas de implementação já são resolvidos, como por exemplo, o problema do efeito avalanche.

Em Basile e Chiacchio (2007) é proposto um método para conversão de autômatos, Redes de Petri (RP) e Redes de Petri Colorida (RPC) em linguagem de *Structured Text* (ST), segundo a norma IEC 61131-3, preservando a estrutura e o comportamento em malha fechada. No algoritmo proposto, os autores assumem que pode ocorrer somente uma mudança de estado para cada ciclo de varredura do CLP, que os supervisores tenham a propriedade *Interleaving sensitivity* e que não sejam afetados por problemas de conflito.

Existem na literatura diversos trabalhos que realizam a implementação de controladores de SEDs em CLPs, a partir dos modelos de Redes de Petri, como por exemplo: (MOREIRA; BASILIO, 2013; ZHOU; DICESARE, 2012; SUESUT *et al.*, 2004; LEE; ZANDONG; LEE, 2004; JIMENEZ; LOPEZ; RAMIREZ, 2001; UZAM; JONES, 1998; BASILE *et al.*, 2004; HOLLOWAY; KROGH, 1990).

Queiroz e Cury (2002) propõem uma estrutura de implementação de sistema de controle, baseada na arquitetura modular local para implementação em CLP, segundo a norma IEC 61131. Esse método baseia-se em uma arquitetura estruturada em três níveis (Supervisores Modulares, Sistema Produto e Sequências Operacionais). Nessa implementação, apenas um evento pode ser tratado por ciclo de varredura do CLP e todos os supervisores devem ser atualizados antes de realizar o tratamento de outro evento. A dissertação de mestrado de Constain (2011), mostrou que essa estrutura de controle traz vantagens, também, na integração do controle em CLP, com as diversas funcionalidades de Sistemas SCADA (*Supervisory Control and Data Aquisition*), como a apresentação de sinóticos, o monitoramento e a visualização de alarmes. Essa estrutura foi, posteriormente, estendida por Scotti (2015) integrando CLP, SCADA e o roteamento de tarefas.

Em Vieira *et al.* (2017) é proposta uma metodologia de implementação baseada

na arquitetura de Queiroz e Cury (2002), a qual pode ser aplicada de forma concentrada em um CLP ou distribuída em um conjunto de CLPs, utilizando a linguagem *Sequential Function Chart* (SFC), segundo a norma IEC 61131-3. Nessa abordagem, os autores propõem a implementação do programa do CLP, por meio de uma rotina principal, que invoca diversas *Functions* (FCs) e *Functions Blocks* (FBs), em ordem sequencial a cada ciclo de varredura do CLP. Leal, Cruz e Hounsell (2009) estendem essa proposta para que seja possível tratar mais de um evento por ciclo de varredura do CLP. O método permite processar o maior número de eventos possíveis por ciclo de varredura e evitar problemas de perda de informação. Em outro artigo mais recente, Leal, Cruz e Hounsell (2012) tratam o problema de escolha utilizando uma variável gerada de forma aleatória, de tal maneira que, a cada ciclo de varredura, essa variável assuma valores entre 0 e 1, evitando assim, que uma parte do código nunca seja executada. Para uma leitura de outros possíveis métodos de implementação em CLPs, considere o *survey* de Zaytoon e Riera (2017).

A abordagem original da TCS de Ramadge e Wonham (1989) não trata o tempo explicitamente e, portanto, não é capaz de lidar com especificações que envolvam requisitos temporais. A inclusão do tempo no modelo de um SED aumenta a capacidade de modelagem e é essencial em aplicações temporalmente críticas. O controle supervisório temporizado de Brandin e Wonham (1994) integra um relógio digital aos modelos de SEDs, para que um intervalo de tempo discreto possa ser atribuído a cada evento na planta. Essa abordagem permite lidar com problemas de restrições de tempo. O problema de complexidade do espaço de estados é ainda mais acentuado em SEDT, considerando nos modelos um evento discreto adicional para representar cada incremento do relógio digital. Para os quais, além de especificações de não-bloqueio e de ordenamento na ocorrência de eventos, são impostas restrições sobre o tempo entre a ocorrência de eventos discretos. Essa situação é típica de sistemas de manufatura controlados por CLPs. Na pesquisa de mestrado de Schafaschek (2014), a abordagem de síntese de supervisores, baseada em um modelo de autômatos temporizados, foi estendida para explorar arquiteturas de SEDT compostos, através da abordagem de CML, de forma a reduzir a complexidade do processo de síntese de supervisores para sistemas a eventos discretos de grande porte.

Esta Tese visa sistematizar um método para a aplicação da Teoria de Controle Supervisório Temporizado a sistemas de manufatura sujeitos a incertezas de operação e a requisitos temporais, abrangendo a modelagem do processo e de especificações, a síntese de controle modular local temporizado e a implementação de supervisores temporizados em controladores lógicos programáveis.

Normalmente, em sistemas de manufatura, os tempos de processamentos e as especificações de restrições temporais estão definidas em um domínio contínuo de

valores reais. Portanto, os intervalos de tempos contínuos associados aos modelos de SEDs de Brandin-Wonham precisam ser discretizados e expressos como múltiplos do período do relógio digital global. Dentre os trabalhos que realizam o controle temporizado em sistemas de manufatura (AFZALIAN; NOORBAKHSH; WONHAM, 2010; BRANDIN; WONHAM, 1993; LEDUC, 1996; LEDUC; WANG; AHMED, 2014; MIRZAEI; POUYAN; BIGLARI, 2020; WANG; LI, 2012; ZHANG *et al.*, 2013), os modelos já partem de tempos discretos e, no caso de Afzalian, Noorbakhsh e Wonham (2010), Brandin e Wonham (1993), Zhang *et al.* (2013), esse tempo é representado em números de *ticks*.

A primeira contribuição desta Tese, é a modelagem e síntese de um controle supervisório para um sistema de manufatura real com tempos de processamento incertos, devido à intervenção humana direta, operação flexível e restrições de tempo como *time-lags* e *delay*. Para isso, por um lado, deve-se considerar que o intervalo discreto obtido garante que os eventos ocorram sempre dentro do intervalo contínuo, originalmente definido e, por outro lado, que essa condição seja atendida de forma minimamente conservadora. Para um dado modelo de sistema estabelecido no mundo real, o problema que se quer resolver se resume em como traduzir esse intervalo de tempo para o mundo discretizado.

Para lidar com a complexidade computacional na síntese dos supervisores para SED temporizado que resulta da explosão do número de estados em sistemas compostos, utiliza-se a abordagem modular local de Schafaschek, Queiroz e Cury (2017) que explora a modularidade da planta e especificação.

Dentre os diversos artigos que abordam a implementação do controle baseado na TCS em CLP industrial, (CANTARELLI; ROUSSEL, 2008; FABIAN; HELLGREN, 1998; JIAO; GAN, 2013; LEAL; CRUZ; HOUNSELL, 2009; LEAL; CRUZ; HOUNSELL, 2012; LEDUC; WANG; AHMED, 2014; MONIRUZZAMAN; GOHARI, 2007; MUŜIĈ; MATKO, 2005; PRENZEL; PROVOST, 2018; QUEIROZ; CURY, 2002; ROUSSEL; GIUA, 2005; UZAM, 2012; Vieira *et al.*, 2017), o único trabalho que considera o controle SEDT é o de Leduc, Wang e Ahmed (2014), que propõe uma implementação para SEDT através de controladores de dados amostrados (SD), acionados por um relógio periódico, que considera o sistema como uma série de entradas e saídas que podem assumir os valores de verdadeiro ou falso. Eles apresentam, também, os problemas de simultaneidade e atraso relacionados à implementação do SEDT, e os resultados de controlabilidade, não bloqueio para controladores SD e um método para conversão do controlador em uma máquina de estado finito síncrona (FSM) de Mooore são expostos. No entanto, suas suposições sobre a classe de eventos e modelos têm sido fortes demais para lidar com a proposta de implementação do CMLT em CLPs. Por exemplo, eles assumem que o período do relógio digital é igual ao ciclo de varredura do CLP, o que pode levar a modelos temporizados muito grandes e, também, consideram

que todos os eventos controláveis correspondem a saídas forçadas do CLP, o que não é o caso da interação humana em sistemas de manufaturas inteligentes (*Smart Manufacturing systens*).

A segunda contribuição desta Tese, é a proposta de uma nova arquitetura para implementação de controle supervisório temporizado em CLP, que é aplicada a dois estudos de caso. Essa arquitetura estende o método de Vieira *et al.* (2017) para o caso SEDT, seguindo uma estrutura modular baseada na execução de máquinas de estados concorrentes. Algumas questões são levantadas, de forma a garantir o correto tratamento dos eventos. Assim, realizou-se um estudo sobre os possíveis problemas que podem ocorrer na implementação do controle modular local temporizado (CMLT) em CLPs, analisando como esses problemas se comportam na nova estrutura proposta.

Sumarizando, o presente trabalho tem como objetivo geral o desenvolvimento de contribuições que fundamentem a implementação do Controle Supervisório Modular Local Temporizado em Controladores Lógicos Programáveis. Para isso, propõe-se uma arquitetura de controle modular local temporizada para a implementação física do sistema de controle, em conjunto com um método para a modelagem (discretização dos intervalos de tempos contínuos) de sistemas de manufatura com tempos de processamentos incertos e restrições de tempo. Nesse sentido, é apresentada uma análise sobre os aspectos do problema de implementação de SEDT em CLPs e os resultados práticos de duas implementações reais bem-sucedidas, envolvendo um sistema modular de produção.

ORGANIZAÇÃO DO DOCUMENTO

Esta Tese está organizada como segue. O Capítulo 1 apresenta um panorama geral do tema, com a proposta de um método de implementação do CMLT em CLPs.

Realiza-se, no Capítulo 2, uma revisão dos principais conceitos da teoria de controle supervisório, tendo como enfoque principal a abordagem modular local. Apresenta-se, inicialmente, um problema real de controle que será utilizado como exemplo (Estação MPS Festo de Separação) para a ilustração dos conceitos e metodologias apresentadas.

O Capítulo 3 apresenta uma revisão dos principais conceitos e problemas sobre a implementação do sistema de controle não temporizado em CLPs, e a aplicação da metodologia ao exemplo do Capítulo 2.

No Capítulo 4, realiza-se uma revisão dos principais conceitos da teoria de controle supervisório de SEDT, tendo como enfoque principal a abordagem modular local temporizada. O problema real do Capítulo 2 é incorporado com restrições temporais e utilizado como exemplo para a aplicação dos conceitos e metodologias apresentadas.

A proposta de extensão da arquitetura de implementação do CML para sistemas temporizados é apresentada no Capítulo 5, juntamente com um estudo relativo aos problemas de implementação do CMLT em CLPs, e a aplicação dessa arquitetura no mesmo problema resolvido no Capítulo 4.

O Capítulo 6 traz uma proposta de modelagem de sistemas de manufaturas com tempos de processamento incertos e restrições de tempo, bem como sua aplicação a um pequeno sistema de manufatura. Realiza-se, também, um estudo sobre o impacto da granularidade do período do relógio digital.

No Capítulo 7 apresenta-se a modelagem, a síntese e implementação do CMLT aplicado a um sistema real de produção, sob incertezas de processamento e restrições de tempo.

E, por fim, o Capítulo 8 trata das considerações finais desta pesquisa de doutorado, com possíveis perspectivas futuras de estudo.

2 CONTROLE SUPERVISÓRIO DE SISTEMAS A EVENTOS DISCRETOS NÃO TEMPORIZADOS

Neste capítulo, apresenta-se, de forma resumida, os principais conceitos da Teoria de Controle Supervisório de sistemas a eventos discretos, proposta por Ramadge e Wonham (1987), tendo como foco principal o controle modular local proposto por Queiroz e Cury (2000).

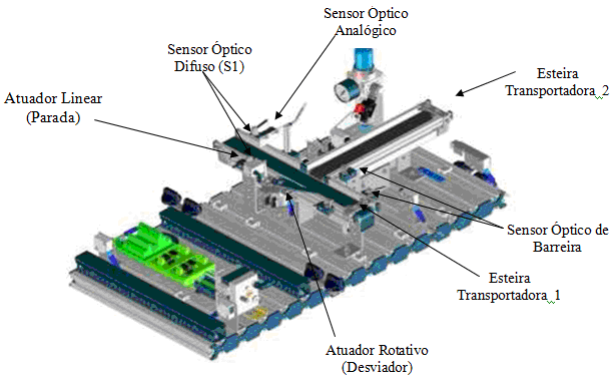
Na sequência é apresentado um problema real de controle, aplicado a um Sistema Modular de Produção (MPS), localizado no Laboratório de Automação Industrial (LAI) da UFSC. A resolução desse problema servirá para exemplificar os conceitos apresentados nas seções seguintes. Salienta-se, ainda, que a implementação desse sistema real foi apresentado no congresso *Fluid Power Net International* (FPNI), em outubro de 2016 (SZPAK; QUEIROZ, 2016).

Em uma segunda etapa, a solução será estendida e implementada para modelos temporizados, de acordo com a arquitetura de implementação do CMLT em CLPs, que será proposta e apresentada no Capítulo 5.

2.1 EXEMPLO MOTIVADOR: ESTAÇÃO DE SEPARAÇÃO

A planta MPS da Festo, localizada no LAI, é constituída por sete módulos, onde cada um é controlado individualmente por um CLP da marca Siemens da série S7-1200. Dentre os módulos que compõem a planta, está a Estação de Separação (*MPS Separating*) conforme ilustrada na Figura 1[1].

Figura 1 – Estação de Separação



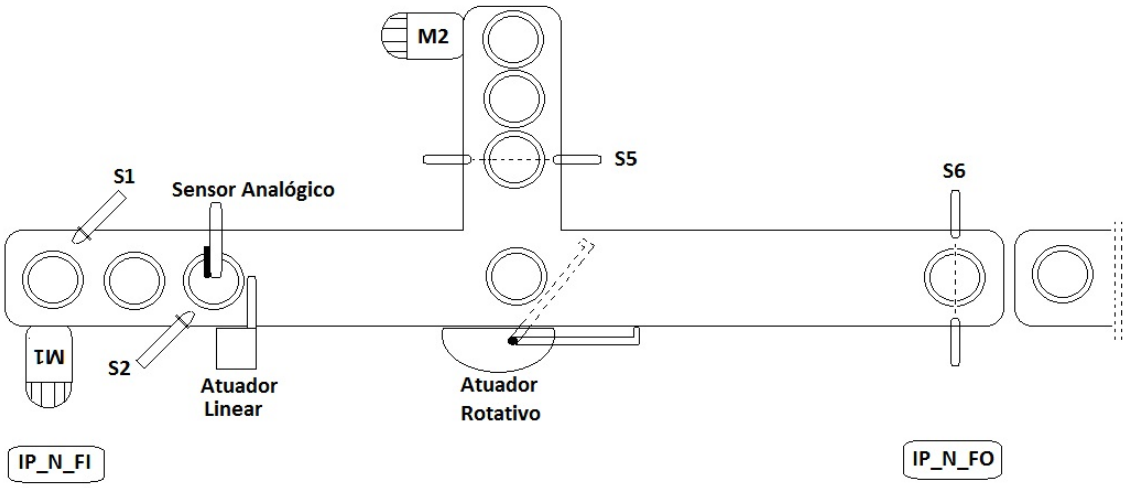
Fonte: Adaptado de Treinamento Festo Didactic (2012, informação verbal).

A Estação de Separação tem como objetivo principal detectar a profundidade do furo das peças, separando-as em duas direções diferentes do fluxo de material (Figura 2). Assim, as peças colocadas na esteira principal ($E1$) são transportadas até

[1] Curso realizado, quando do recebimento das bancadas didáticas para o Laboratório de Automação Industrial, do Instituto Federal de Santa Catarina (IFSC), Chapecó.

o ponto de medição, e por meio de um sensor óptico difuso, verifica-se a profundidade do furo e seleciona-se as peças de acordo com o resultado dessa medição. O atuador linear acionado bloqueia a passagem das peças até o final do teste e, dependendo do resultado desse teste, as peças *"boas"* são deslocadas até o fim da esteira principal e as peças *"rejeitadas"* são movidas para uma esteira secundária ($E2$), por meio do atuador pneumático rotativo.

Figura 2 – Esquema simplificado da Estação de Separação



Fonte: Elaborado pelo autor (2021).

O programa fornecido pelo fabricante dessa estação permite operar, em sequência, apenas uma peça por vez. Essa lógica pode ser facilmente implementada em controladores lógicos, porém, a solução é pouco eficiente. Logo, o objetivo desse estudo é sintetizar um novo programa de controle, através de uma metodologia formal que garanta o não bloqueio e uma maior eficiência da estação.

Assim, deseja-se resolver o problema por meio de uma lógica de controle a ser implementada no CLP. Esse problema de controle a ser tratado pode ser dividido em três objetivos, quais sejam:

- **Objetivo 1:** permitir o controle do maior número de peças concorrentes na estação, de forma a garantir uma produção contínua, minimamente restritiva e sem situações de bloqueio.
- **Objetivo 2:** tratar erros e falhas que podem ser gerados no sistema, como por exemplo: erros de deslocamento no fluxo de material; e falhas geradas pela intervenção humana, como a retirada de uma peça da estação.
- **Objetivo 3:** minimizar o tempo de produção, de modo que as peças classificadas como *"boas"* sejam deslocadas para a próxima estação, no menor tempo possível.

Para atingir os objetivo 1 e 2, busca-se: modelar e calcular uma lógica de controle utilizando a abordagem modular local da TCS; implementar a lógica de controle no CLP; e desenvolver um sistema SCADA para o acionamento, monitoramento e controle remoto. Para solucionar o problema 3, utiliza-se a abordagem modular local temporizada da TCS para modelar e realizar a síntese, e a arquitetura temporizada que será proposta no Capítulo 5, para implementar a lógica de controle no CLP. Elabora-se, também, um sistema SCADA para o acionamento, monitoramento e o controle remoto da planta.

2.2 MODELAGEM

Segundo Wonham e Cai (2019), o problema de controle supervisório para sistemas a eventos discretos é formulado pela modelagem da planta e das especificações como autômatos finitos, e para resolver o problema do controle supervisório é necessário projetar um supervisor que garanta que as especificações sejam atingidas. Nesta seção, são introduzidos os conceitos de base da teoria de Linguagens e Autômatos, e sua aplicação na modelagem da planta e das especificações aplicada à Estação de Separação.

2.2.1 Linguagens e Autômatos

Para a modelagem matemática de sistemas a eventos discretos, associam-se os eventos a símbolos. Seja, Σ o conjunto finito de símbolos representando os eventos que podem ocorrer no sistema a ser controlado (planta). Σ^* é o conjunto de todas as cadeias finitas de elementos do conjunto Σ, incluindo a cadeia vazia ε. Sejam s,u cadeias de Σ^*, uma cadeia u é dita ser prefixo de s se $\exists v \in \Sigma^*$, $s = uv$.

Segundo Ramadge e Wonham (1989), consideram-se que todos os eventos são gerados espontaneamente e que a ocorrência de alguns eventos pode ser evitada. Portanto, o alfabeto de eventos Σ é particionado em um conjunto de eventos controláveis Σ_c, que podem ser habilitados ou desabilitados por um agente externo (supervisor), e um conjunto de eventos não-controláveis Σ_u, que não podem ser inibidos de ocorrer no sistema. Logo $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$.

Uma linguagem (L) sobre o alfabeto Σ é um subconjunto de Σ^*. Portanto, o comportamento de um SED composto por um alfabeto Σ pode ser modelado por uma linguagem $L \subseteq \Sigma^*$, que representa todas as cadeias finitas de eventos que o sistema pode gerar. O prefixo-fechamento de uma linguagem L é definido por $\overline{L} := \{u \in \Sigma^* : \exists v \in \Sigma^* \wedge uv \in L\}$.

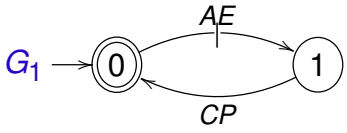
As linguagens podem ser associadas a autômatos, representados por uma quíntupla $\mathbf{G} = (Q,\Sigma,\delta,q_0,Q_m)$, no qual: Q é um conjunto finito de estados; $q_0 \in Q$

é o estado inicial; Σ é o alfabeto que representa um conjunto não vazio e finito de eventos; Q_m é um conjunto finito de estados marcados, no qual $Q_m \subseteq Q$; e a função de transição $\delta : Q \times \Sigma \rightarrow Q$, é uma função parcial, definida em cada estado de Q para um subconjunto de Σ. A função de transição é um mapeamento das transições $\delta(q,\sigma) = q'$, o qual gera o próximo estado q', após a ocorrência do evento σ, quando estiver no estado q. Pode-se associar o autômato $\mathbf{G}$ a uma função de eventos ativos $\Gamma : Q \rightarrow Pwr(\Sigma)$[2], que indica para cada estado $q \in Q$ o conjunto de eventos que podem ocorrer.

A linguagem gerada de $\mathbf{G}$, denotada por $L(\mathbf{G})$ é o conjunto de todas as sequências de eventos que são fisicamente possíveis de ocorrer na planta. Formalmente é definida por $L(\mathbf{G}) = \{s \in \Sigma^* | \hat{\delta}(q_o,s)!\}$ [3]. Diz-se que $L(\mathbf{G})$ é prefixo fechada, pois nenhuma sequência de eventos na planta pode ocorrer sem que o seu prefixo ocorra primeiro. A linguagem marcada de $\mathbf{G}$, denotada por $L_m(\mathbf{G})$, descreve o comportamento marcado do sistema, isto é, contém todas as cadeias que levam de q_0 a um estado de Q_m. $L_m(\mathbf{G})$ é uma sublinguagem de $L(\mathbf{G})$, ou seja, $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$. Define-se, formalmente, a linguagem marcada de $\mathbf{G}$ como $L_m(\mathbf{G}) = \{s \in L(\mathbf{G}) | \hat{\delta}(q_o,s) \in Q_m\}$.

Os autômatos podem ser ilustrados por grafos direcionados, em que os círculos simples representam os estados e os círculos duplos, os estados marcados. Os arcos representam a transição de um estado para outro, por meio da ocorrência de um evento, de tal modo que, os arcos com um traço representam os eventos controláveis e os arcos sem traço, os não-controláveis. Por exemplo, seja o autômato G_1 ilustrado na Figura 3, em que $\Sigma_{G_1} = \{AE, CP\}$, têm-se, que o comportamento gerado desse modelo é $L(\mathbf{G})$= { ε, AE, $AE.CP$, $AE.CP.AE$, ... } e o marcado $L_m(\mathbf{G})$={ ε, $AE.CP$, $AE.CP.AE.CP$, $AE.CP.AE.CP.AE.CP$, ... }, sendo o evento AE é controlável e o evento CP é não-controlável.

Figura 3 – Subsistema $\mathbf{G}_1$ - Entrada



Estados	
0	entrada proibida
1	entrada autorizada

Eventos		Descrição
AE	controlável	autoriza entrada de peças novas
CP	não-controlável	indica a chegada de uma nova peças

Fonte: Elaborado pelo autor (2021).

[2] Seja A um conjunto. $Pwr(A)$ é o conjunto de todos os subconjuntos de A (conjunto potência de A).

[3] Emprega-se a notação $\delta(q,s)!$ para especificar que a função é definida para o par ordenado (q,σ). A função de transição δ pode ser estendida para cadeias de eventos. Assim, defini-se-a função de transição estendida como a função $\hat{\delta} : Q \times \Sigma \rightarrow Q$ tal que, para $q \in Q$, $s \in \Sigma^*$ e $\sigma \in \Sigma$, $\hat{\delta}(\varepsilon,q) = q$ e $\hat{\delta}(s\sigma,q) = \delta(\sigma,\hat{\delta}(s,q))$ sempre que $q' = \hat{\delta}(s,q) =$ e $\delta(\sigma,q')$ estiverem ambas definidas.

2.2.2 Operações sobre linguagens e autômatos

Diz-se que L é prefixo-fechada se $L = \overline{L}$. Um estado $q \in Q$ é dito acessível (Ac) se $\exists s \in \Sigma^* | \delta(q_0,s) = q$, isto é, se existir uma cadeia aceita por $\mathbf{G}$ que partindo do estado inicial q_0 alcance o estado q. $Ac(\mathbf{G})$ denota a operação que contém a componente acessível do referido autômato. Um SED é dito ser não bloqueante se $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$. Isso significa que existe sempre uma sequência de eventos que leva a planta a partir de qualquer estado (acessível) para um estado desejado (marcado).

A composição de autômatos é realizada por um procedimento chamado de produto síncrono, representado por duas barras paralelas (||), que combina dois autômatos ($\mathbf{G_1}$ e $\mathbf{G_2}$) num único autômato $\mathbf{G} = \mathbf{G_1}||\mathbf{G_2}$. Dados dois autômatos $\mathbf{G_1} = (Q_1,\Sigma_1,\delta_1,q_{01},Q_{m1})$ e $\mathbf{G_2} = (Q_2,\Sigma_2,\delta_2,q_{02},Q_{m2})$, define-se a composição síncrona como: $\mathbf{G_1}||\mathbf{G_2} = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{1||2}, (q_{01},q_{02}), Q_{m1} \times Q_{m2})$ onde, $\delta_{1||2} : (Q_1 \times Q_2) \times (\Sigma_1 \cup \Sigma_2) \rightarrow (Q_1 \times Q_2)$, é definida por:

$$\delta_{1||2}((q_1,q_2),\sigma) = \begin{cases} (\delta_1(q_1,\sigma),\delta_2(q_2,\sigma)) \textit{se } \sigma \in \Sigma_1 \cap \Sigma_2 \textit{ e } \sigma \in \Sigma_1(q_1) \cap \Sigma_2(q_2); \\ (\delta_1(q_1,\sigma), q_2) \textit{se } \sigma \in \Sigma_1 \textit{ e } \sigma \notin \Sigma_2 \textit{ e } \sigma \in \Sigma_1(q_1); \\ (q_1,\delta_2(q_2,\sigma)) \textit{se } \sigma \in \Sigma_2 \textit{ e } \sigma \notin \Sigma_1 \textit{ e } \sigma \in \Sigma_2(q_2); \\ \textit{Indefinida, caso contrário.} \end{cases}$$

O produto síncrono define novos estados para $\mathbf{G}$. Um evento comum a Σ_1 e Σ_2 só pode ser executado sincronamente nos dois autômatos, os demais ocorrem assincronamente, isto é, de modo independente. Se os alfabetos são iguais $\Sigma_1 = \Sigma_2$, a composição é completamente síncrona e, caso contrário, se $\Sigma_1 \cap \Sigma_2 = \emptyset$, não existe nenhuma sincronização entre os eventos dos dois autômatos.

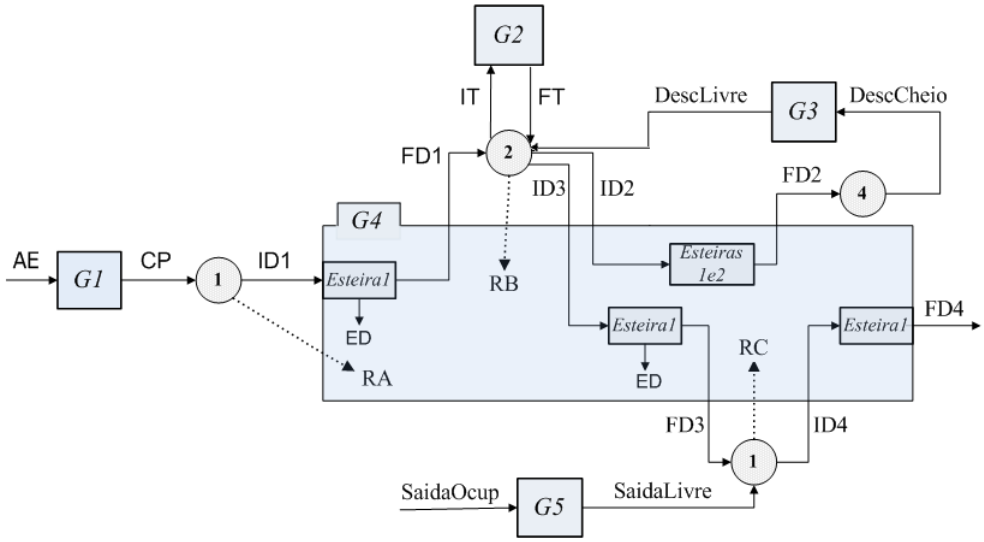
2.2.3 Modelagem dos autômatos da Estação de Separação

Cada módulo do MPS é composto por subsistemas concorrentes que interagem entre si para a realização das tarefas. O comportamento em malha aberta da estação a ser modelada pode ser representado por 5 subsistemas concorrentes:

- $\mathbf{G}_1$ - chegada de peças novas;
- $\mathbf{G}_2$ - teste de profundidade;
- $\mathbf{G}_3$ - descarte de peças rejeitadas;
- $\mathbf{G}_4$ - deslocamentos; e
- $\mathbf{G}_5$ - saída de peças boas.

A Figura 4 ilustra de forma mais abstrata o comportamento da estação a ser modelada, em que os subsistemas $\mathbf{G}_1$, $\mathbf{G}_2$, $\mathbf{G}_3$ e $\mathbf{G}_5$ são representados por quadrados; o subsistema $\mathbf{G}_4$ que modela os deslocamentos da planta é composto pelas esteiras *E*1 e *E*2 representadas por retângulos; os *buffer*s com suas respectivas capacidades são representados por círculos; e os eventos do sistema e suas interações com cada subsistema, por setas.

Figura 4 – Esquema da Estação de Separação



Fonte: Elaborado pelo autor (2021).

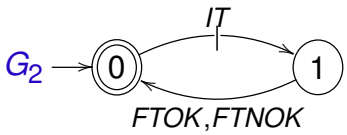
Na sequência é apresentada a modelagem dos 5 subsistemas, segundo a Teoria de Controle Supervisório.

$\mathbf{G}_1$ **– Entrada:** o autômato da Figura 3 modela a entrada de novas peças na estação. O estado inicial "0" indica entrada proibida de peças na estação e o estado "1" entrada autorizada. Quando ocorrer o evento controlável *AE*, será habilitada a entrada de peças novas na estação, o que é feito pela transição de um sinal ótico à estação precedente. O evento não-controlável *CP* acontecerá quando o sensor S1 identificar a chegada de peças novas.

$\mathbf{G}_2$ **- Teste:** o autômato da Figura 5 modela o teste de profundidade das peças, em que o estado inicial representa o sistema aguardando o seu início, e o estado "1" realizando o teste. Quando o evento controlável início do teste (*IT*) estiver ativo, retornará como resultado, peças boas (*FTOK*) ou peças rejeitadas (*FTNOK*).

$\mathbf{G}_3$ **– Descarte:** o autômato da Figura 6 modela o descarte de peças rejeitadas, em que o estado inicial representa *buffer D* livre e o estado "1", o *buffer D* cheio. Se o sensor *S*5 ficar ativo por mais de 1 segundo, um sinal será enviado ao sistema, informando que o *buffer* do descarte está cheio. Esse sinal é representado pelo evento não-controlável *Desc_Cheio*. Nesse estado, as peças rejeitadas devem ficar aguar-

Figura 5 – Subsistema $\mathbf{G}_2$ - Teste



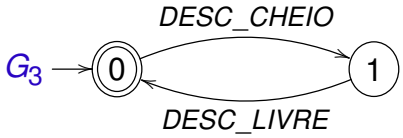
Estados	
0	não testando peça
1	testando peça

Eventos		Descrição
IT	controlável	início de teste
FTOK	não-controlável	fim de teste ok (peça boa)
FTNOK	não-controlável	fim de teste não ok (peça rejeitada)

Fonte: Elaborado pelo autor (2021).

dando a ocorrência do evento não-controlável *Desc_Livre* (desativação de *S*5) para serem deslocadas e as peças só podem ser deslocadas para o *buffer* de descarte se o mesmo estiver livre.

Figura 6 – Subsistema $\mathbf{G}_3$ - Descarte



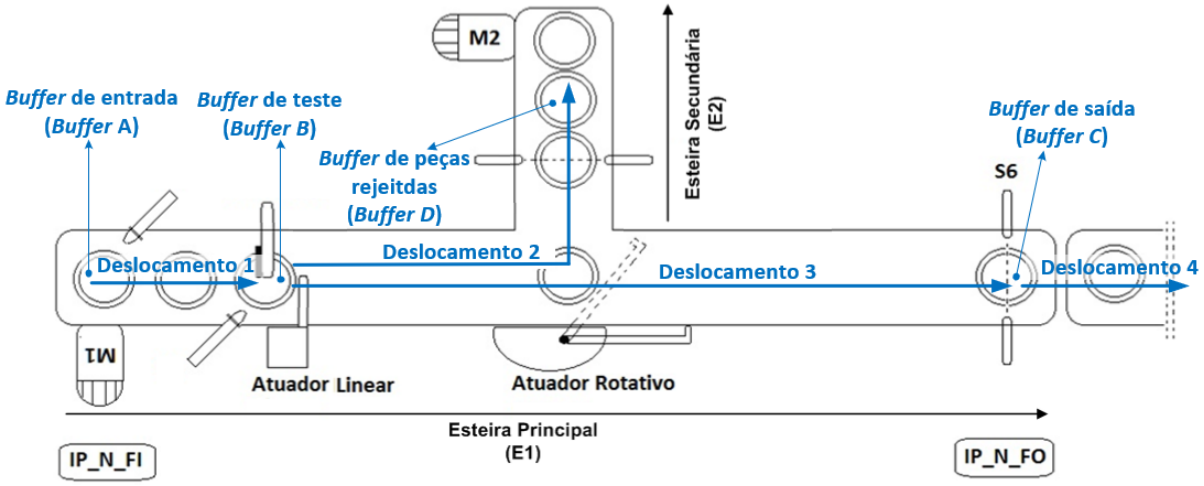
Estados	
0	*buffer* descarte vazio
1	*buffer* descarte cheio

Eventos		Descrição
DESC_LIVRE	não-controlável	*buffer* esteira descarte está vazio
DESC_CHEIO	não-controlável	*buffer* esteira descarte está cheio

Fonte: Elaborado pelo autor (2021).

$\mathbf{G}_4$ **– Deslocamentos:** conforme a Figura 7 a Estação de Separação é dividida em 4 deslocamentos possíveis.

Figura 7 – Deslocamentos Estação de Separação



Fonte: Elaborado pelo autor (2021).

Portanto, os eventos de deslocamento deste subsistema são nomeados da seguinte forma: os eventos de início de deslocamento têm nomes que começam com *IDi* e os eventos de fim de deslocamento começam com *FDi* para i = 1,2,3,4. A Tabela 1 mostra, de forma resumida, os eventos de início e fim de deslocamento com suas respectivas combinações, e apresenta, também, o erro de deslocamento.

Tabela 1 – Lista de eventos subsistema G_4

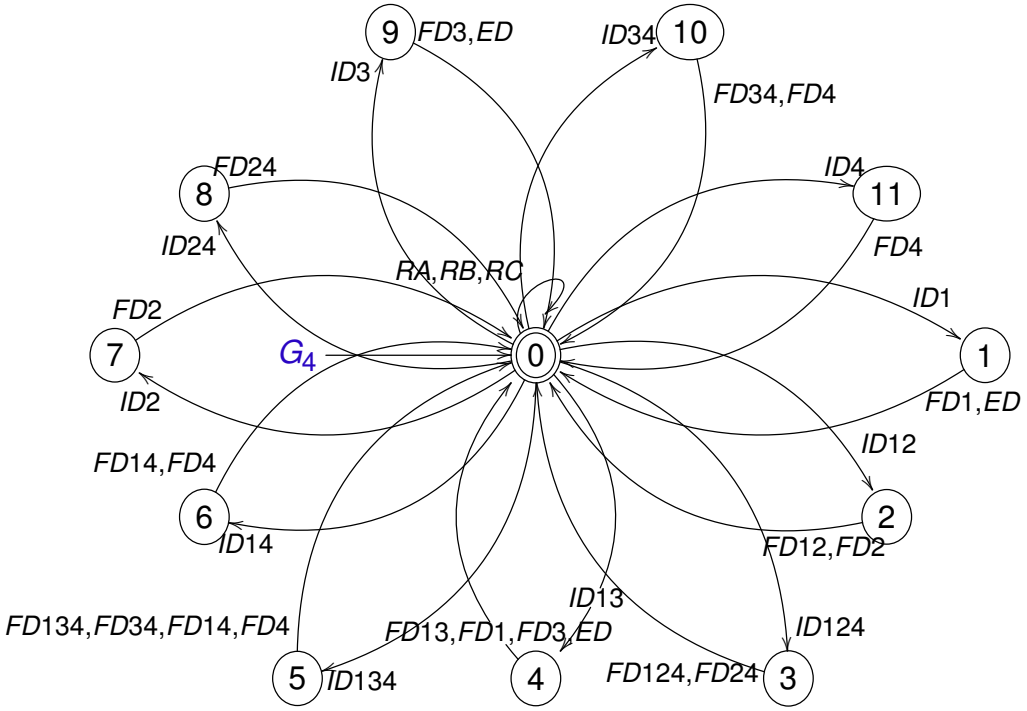
Eventos		**Descrição**
ID1	controlável	comando de início de deslocamento 1, que consiste em levar uma peça do *buffer* A (BA) para o *buffer* B (BB)
ID2	controlável	comando de início de deslocamento 2, que consiste em levar uma peça testada e rejeitada do BB para o *buffer* D (BD)
ID3	controlável	comando de início de deslocamento 3, que consiste em levar uma peça testada e boa do BB para o *buffer* C (BC)
ID4	controlável	comando de início de deslocamento 4, que consiste em retirar uma peça do BC quando a saída da estação é autorizada
ID12	controlável	início do deslocamento 12, que consiste em levar uma peça de BA para BB ao mesmo tempo que transporta de BB para BD.
ID13	controlável	início do deslocamento 13, que consiste em levar uma peça de BA para BB ao mesmo tempo que transporta de BB para BC.
ID14	controlável	início do deslocamento 14, que consiste em levar uma peça de BA para BB ao mesmo tempo que retira uma peça do BC
ID24	controlável	início do deslocamento 24, que consiste em levar uma peça de BB para BD ao mesmo tempo que retira uma peça do BC
ID34	controlável	início do deslocamento 34, que consiste em levar uma peça de BB para BC ao mesmo tempo que retira uma peça do BC
ID124	controlável	início do deslocamento 124, que consiste em levar uma peça do BA para o BB, ao mesmo tempo que do BB para o BD, retirando uma peça do BC
ID134	controlável	início do deslocamento 134, consiste em levar uma peça do BA para BB, ao mesmo tempo que do BB para o BC, retirando uma peça do BC
FD1	não-controlável	fim do deslocamento 1, chagada de uma peça no BB (decorrente da ativação de S2)
FD2	não-controlável	fim do deslocamento 2, chagada de uma peça no BD
FD3	não-controlável	fim do deslocamento 3, chagada de uma peça no BC (decorrente da ativação de S6)
FD4	não-controlável	fim do deslocamento 4, chagada de uma peça na estação seguinte
FD12	não-controlável	fim do deslocamento 12, chagada de peças no BB e no BD
FD13	não-controlável	fim do deslocamento 13, chagada de peças no BB e no BC
FD14	não-controlável	fim do deslocamento 14, chagada de peças no BB e na estação seguinte
FD24	não-controlável	fim do deslocamento 24, chagada de peças no BD e na estação seguinte
FD34	não-controlável	fim do deslocamento 34, chagada de peças no BC e na estação seguinte
FD124	não-controlável	fim do deslocamento 124, chagada de peças no BB, BD e na estação seguinte
FD134	não-controlável	fim do deslocamento 134, chagada de peças no BB, BC e na estação seguinte
ED	não-controlável	erro de deslocamento, ocorre quando uma peça não chega no BB ou no BC, após um determinado tempo

Fonte: Elaborado pelo autor (2021).

O autômato da Figura 8 modela o comportamento dos deslocamentos das es-

teiras, a detecção de erros de deslocamentos e a retirada de peças nos *buffer*s de entrada, teste e saída. O movimento do sistema se dá pela esteira principal ($E1$) e secundária ($E2$), e seus inícios de deslocamento podem ser combinados, em função das condições do próprio sistema, gerando um conjunto de 11 possibilidades de deslocamentos.

Figura 8 – Subsistema $\mathbf{G_4}$ - Deslocamentos



Estados	
0	esteiras desligadas
1, 4, 5, 6, 9, 10, 11	esteira 1 ligada com desviador recuado
2, 3, 7, 8	esteiras 1 e 2 ligadas com desviador avançado

Eventos		**Descrição**
IDi	controlável	início do deslocamentos i
FDi	não-controlável	fim do deslocamentos i
RA	não-controlável	peça do *buffer* A foi retirada
RB	não-controlável	peça do *buffer* B foi retirada
RC	não-controlável	peça do *buffer* C foi retirada
ED	não-controlável	erro de deslocamento

Fonte: Elaborado pelo autor (2021).

Os eventos $ID1$, $ID3$, $ID4$, e suas combinações $ID13$, $ID14$, $ID34$, $ID134$, levam o sistema, aos estados “1”, “4”, “5”, “6”, “9”, “10” e “11” que representam a esteira principal em movimento, ao passo que a esteira secundária está parada. O evento $ID2$, leva o sistema ao estado “7” que representa as esteiras $E1$ e $E2$ em movimento. Os eventos $ID12$, $ID124$, $ID24$, levam o sistema aos estados “2”, “3”, “8” que representam as esteiras $E1$ e $E2$ em movimento. Todos eventos FDi, ou seja, $FD1$, $FD2$, $FD3$, $FD4$, $FD12$, $FD13$,...,$FD134$, retornam o sistema ao seu estado inicial “0”. O evento ED detecta a ocorrência de erro nos finais de deslocamento $FD1$ e $FD3$, isto é, se a peça após um determinado tempo não chegar ao *buffer* de teste ou ao *buffer* de saída,

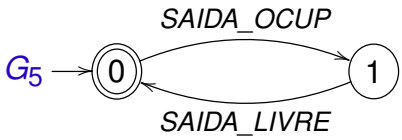
ocorrerá um erro no deslocamento, levando o sistema ao seu estado inicial “0”.

Conforme ilustrado na Figura 8, o estado inicial “0” representa o sistema em repouso, aguardando algum *IDi*. Logo, quando o evento *ID*1 ocorrer (for acionada a esteira E1), devido à chegada de uma nova peça no *buffer* de entrada, o autômato transitará para o estado “1”, que representa a esteira principal em movimento. Assim, a esteira *E*1 é acionada até que a peça alcance o *buffer* de teste e o evento *FD*1 seja gerado, com o respectivo desligamento da esteira E1. Caso não ocorra *FD*1, após determinado período de tempo, um erro de deslocamento irá ocorrer e o evento *ED* será gerado, fazendo com que o sistema retorne ao seu estado inicial. Da mesma forma, se o evento *ID*12 está habilitado, significa que tem uma peça nova no *buffer* de entrada e uma peça testada e rejeitada no *buffer* de teste. Portanto, o sistema transitará para o estado “2”, que representa as esteiras *E*1 e *E*2 em movimento. Logo, ambas as esteiras são acionadas levando uma peça para o *buffer* de teste e a outra para o *buffer* de peças rejeitadas, habilitando, assim, o evento *FD*12 que irá fazer com que o autômato retorne ao seu estado inicial. Essa lógica segue para todas as 11 possíveis combinações de início e fim de deslocamento.

Outra forma de falha no fluxo de material de sistemas de automação eletropneumáticos, em decorrência da intervenção humana, é a retirada de peças do sistema. Nesse exemplo, as peças podem ser retiradas dos *buffer*s A, B e C a qualquer instante, quando o sistema estiver em repouso. Nesse caso, quando o sistema estiver no estado inicial, os eventos RA, RB e RC poderão ocorrer.

$\mathbf{G_5}$ **– Saída:** o autômato da Figura 9 modela o comportamento da saída de peças da estação. O estado inicial representa saída livre e, o estado “1”, saída ocupada. Quando o sensor *IP_N_FO* enviar um sinal ao sistema informando que a estação está ocupada, o evento *SAIDA_OCUP* será ativado e a saída ficará bloqueada. Nesse estado, as peças devem ficar aguardando a ocorrência do evento não-controlável *SAIDA_LIVRE* para serem deslocadas até a próxima estação.

Figura 9 – Subsistema $\mathbf{G_5}$ - Saída



Estados	
0	saída livre
1	saída ocupada

Eventos		Descrição
SAIDA_LIVRE	não-controlável	estação 4 livre
SAIDA_OCUP	não-controlável	estação 4 ocupada

Fonte: Elaborado pelo autor (2021).

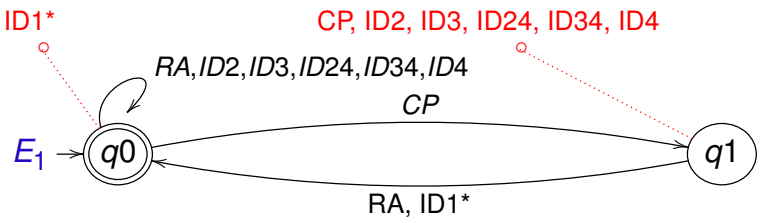
No caso do MPS, a planta global é obtida pelo produto síncrono dos autômatos que representam o comportamento concorrente dos seus subsistemas: $\mathbf{G} = \mathbf{G_1}||\ \mathbf{G_2}||\ \mathbf{G_3}||\ \mathbf{G_4}||\ \mathbf{G_5}$. O autômato ($\mathbf{G}$) gerado através da ferramenta computacional TCT, que representa o comportamento da planta global em malha aberta (sem ação de controle), possui 192 estados e 1456 transições.

2.2.4 Modelagem das especificações da Estação de Separação

Na Teoria de Controle Supervisório, o comportamento de sistemas a eventos discretos em malha aberta pode incluir uma série de cadeias de eventos indesejáveis, resultantes da ação não coordenada dos subsistemas sem ação de um controlador, o que pode levar o sistema a estados indesejados ou proibidos. As especificações são representadas como uma sublinguagem da planta, que pode ser modelada como um autômato. As especificações modelam o comportamento desejado para a planta, mediante a construção de autômatos para cada restrição de coordenação do sistema a ser controlado. Para o estudo de caso apresentado, serão modeladas 7 especificações, as quais resolvem os problemas de interação entre os 5 subsistemas apresentados na seção anterior.

E_1 **– *Buffer* Entrada:** conforme o autômato da Figura 10, essa restrição impede qualquer início de deslocamento *ID*1*, isto é, *ID*1, *ID*12, *ID*13, *ID*124, *ID*134, *ID*14 se não houver peça na entrada, e evita também o *overflow* do *buffer* A, ou seja, que tenha mais de uma peça na entrada. Assim, sempre que uma nova peça chegar na estação, deve-se proibir que o evento *CP* ocorra antes do início de algum movimento *ID*1*. Portanto, assume-se neste trabalho que *IDi** para *i* = 1,2,3,4 representa todos os inícios de deslocamentos que contenham *i*. A fim de destacar os eventos que devem ser desabilitados e, assim, impedidos de ocorrer num determinado estado, adotou-se a simbologia através de uma linha pontilhada, como pode ser observado na figura. Logo, no estado *q*0, os eventos que devem ser desabilitados são todos os inícios de deslocamentos *ID*1* e, no estado *q*1, os eventos *CP*, *ID*2, *ID*3, *ID*24, *ID*34, *ID*4. Essa simbologia será utilizada em todas as restrições apresentadas neste trabalho.

Figura 10 – Especificação E_1 - *Buffer* Entrada

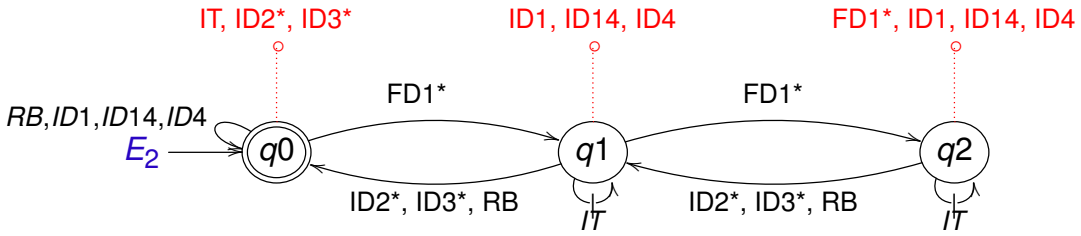


Fonte: Elaborado pelo autor (2021).

E_2 **– *Buffer* Teste:** conforme o autômato da Figura 11, esta restrição trata o

problema de *underflow* (impede o início de teste sem peça no *buffer* B) e *overflow* (impede que o *buffer* B tenha a sua capacidade máxima ultrapassada - duas peças). O início de teste (*IT*) e o início de qualquer deslocamento *ID*2*(*ID*12, *ID*124, *ID*2, *ID*24) ou *ID*3*(*ID*13, *ID*123, *ID*3, *ID*34) não podem acontecer sem que antes tenha ocorrido algum fim de deslocamento *FD*1*(*FD*1, *FD*12, *FD*13, *FD*124, *FD*134, *F*14).

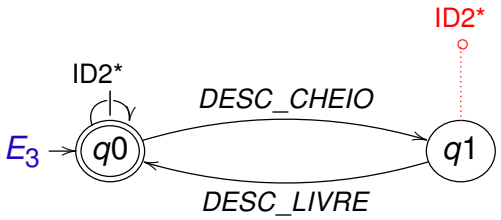
Figura 11 – Especificação E_2 - *Buffer* Teste



Fonte: Elaborado pelo autor (2021).

E_3 **– *Buffer* Descarte:** o autômato da Figura 12, ilustra a restrição para o controle do início de deslocamento da esteira *E*2. Logo, quando o evento não-controlável *DESC_CHEIO* ocorrer, o início de deslocamento *ID*2* será impedido até que o *buffer* de descarte fique livre, isto é, ocorra o evento não-controlável *DESC_LIVRE*.

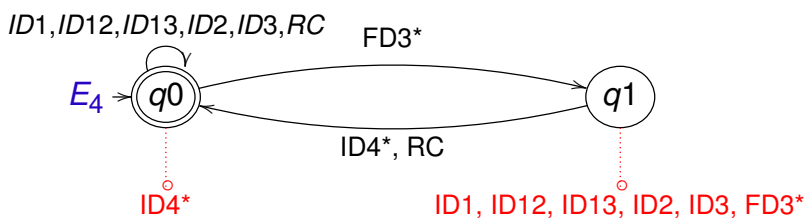
Figura 12 – Especificação E_3 - *Buffer* Descarte



Fonte: Elaborado pelo autor (2021).

E_4 **– *Buffer* Saída:** o autômato da Figura 13, ilustra a restrição de *underflow* e *overflow* para o *buffer* C, que impede o início de qualquer deslocamento *ID*4*(*ID*124, *ID*134, *ID*14, *ID*4), sem que antes tenha ocorrido algum fim de deslocamento *FD*3*(*FD*13, *FD*134, *FD*3, *FD*34).

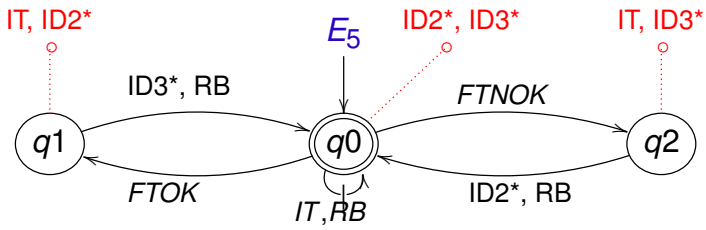
Figura 13 – Especificação E_4 - *Buffer* Saída



Fonte: Elaborado pelo autor (2021).

E_5 **– Fim Teste Desloc_2_3:** conforme o autômato da Figura 14, esta restrição impede o início do teste (*IT*) sem peça no *buffer* B, e controla o início de deslocamento das esteiras $E1$ e $E2$, em função do resultado do teste. Assim, quando o resultado do teste for *FTOK* (peças boas), podem ocorrer os inícios de deslocamentos $ID3^*$ ou a retirada da peça *RB* e, quando for *FTNOK* (peças rejeitadas), podem ocorrer os inícios de deslocamentos $ID2^*$ ou *RB*.

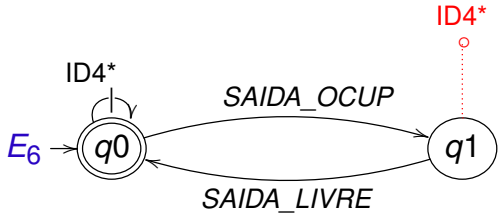
Figura 14 – Especificação E_5 - Deslocamento Fim Teste



Fonte: Elaborado pelo autor (2021).

E_6 **– Deslocamento Saída:** o autômato da Figura 15, ilustra a restrição da saída da estação, impede que a esteira $E1$ seja iniciada sempre que a saída estiver ocupada, evitando a sobreposição das peças. O estado $q1$ indica saída ocupada, impedindo os inícios de deslocamentos $ID4^*$ ($ID14$, $ID124$, $ID134$, $ID24$, $ID34$, $ID4$). Já o estado $q0$ indica saída livre, permitindo o início desses deslocamentos.

Figura 15 – Especificação E_6 - Deslocamento Saída

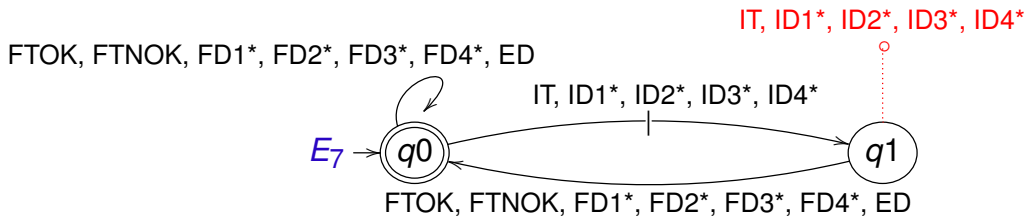


Fonte: Elaborado pelo autor (2021).

E_7 **– Exclusão Mútua:** conforme a Figura 16, essa especificação tem a finalidade de gerar uma exclusão mútua entre o início de deslocamento ($ID1^*$, $ID2^*$, $ID3^*$, $ID4^*$) e o início de teste (*IT*), isto é, o teste não pode ser iniciado se a esteira $E1$ estiver em movimento e vice-versa. O estado $q0$ representa a esteira $E1$ parada e o teste parado; e o estado $q1$ representa a esteira $E1$ em movimento ou com peça sendo testada.

A especificação global é obtida fazendo-se a composição síncrona de todas as especificações do sistema ($E = E_1||E_2||E_3||E_4||E_5||E_6||E_7$), assim, o autômato resultante ($\mathbf{E}$) possui 208 estados e 1924 transições. O comportamento desejado do sistema pode ser expresso pela linguagem alvo K, obtida por $K = L_m(\mathbf{E}||\mathbf{G})$, dessa

Figura 16 – Especificação E_7 - Exclusão Mútua



Fonte: Elaborado pelo autor (2021).

forma, o autômato resultante ($\mathbf{K}$) possui 528 estados e 2992 transições, representando o conjunto de sequências de $\mathbf{G}$ que satisfazem as especificações.

A partir dos modelos obtidos (planta e especificações), realiza-se a síntese dos supervisores. Na seção a seguir apresenta-se a metodologia de síntese de supervisores baseada na TCS e ilustra-se sua aplicação na Estação de Separação.

2.3 SÍNTESE DE CONTROLE SUPERVISÓRIO

Ramadge e Wonham (1989) consideram que todos os eventos são gerados espontaneamente na planta e que a ocorrência de alguns eventos pode ser evitada pelo supervisor. Portanto, para associar a estrutura de controle a uma planta $\mathbf{G}$, o alfabeto de eventos Σ é particionado em um conjunto de eventos controláveis Σ_c, que podem ser habilitados ou desabilitados por um agente externo (supervisor); e um conjunto de eventos não-controláveis Σ_u, que não podem ser inibidos de ocorrer no sistema. Logo $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$.

A planta $\mathbf{G}$ em um SED deve ser controlada com base em certas especificações, isto é, no comportamento lógico requerido. O desempenho desejado da planta a ser controlada poderá ser especificado pela afirmação de que sua linguagem marcada deve estar contida em alguma linguagem da especificação. O conjunto de restrições de coordenação define uma especificação E a ser obedecida. As linguagens gerada $L(\mathbf{G})$ e marcada $L_m(\mathbf{G})$ podem conter cadeias indesejáveis de eventos, por violarem alguma condição que se deseja impor ao sistema, como estados proibidos em $\mathbf{G}$. Assim, de modo a fazer com que os subsistemas atuem de forma coordenada, introduz-se um agente denominado supervisor, que pode ser definido formalmente por uma função $\mathcal{S} : L(\mathbf{G}) \rightarrow Pwr(\Sigma)$, que associa um conjunto de eventos habilitados a cada cadeia de eventos observados na planta. Os eventos a serem desabilitados pela estrutura de controle podem ser calculados como os eventos que podem ocorrer na planta e não estão habilitados por $\mathcal{S}$. Dessa forma, os eventos desabilitados após a ocorrência de $s \in L(\mathbf{G})$ são dados por $\Gamma(\delta(q_0,s)) - \mathcal{S}(s)$, onde Γ é a função de eventos ativos num determinado estado da planta. O supervisor $\mathcal{S}$ é admissível se não implicar a

desabilitação de eventos não-controláveis, isto é, $\forall s \in L(\mathbf{G}), \Sigma_u \cap \Gamma(\delta(q_0,s)) \subseteq \mathcal{S}(s)$. O par $(\mathcal{S},\mathbf{G})$ é escrito $\mathcal{S}/\mathbf{G}$ e representa a planta $\mathbf{G}$ sob supervisão de $\mathcal{S}$. A linguagem gerada por $\mathcal{S}/\mathbf{G}$ é dada por:

$$\begin{cases} 1.\ \varepsilon \in L(\mathcal{S}/\mathbf{G}); \\ 2.\ s\sigma \in L(\mathcal{S}/\mathbf{G}) \Leftrightarrow (s \in L(\mathcal{S}/\mathbf{G})) \wedge (s\sigma \in L(\mathbf{G})) \wedge (\sigma \in \mathcal{S}(s)). \end{cases}$$

A linguagem marcada $L_m(\mathcal{S}/\mathbf{G})$ que representa o comportamento marcado do sistema sob supervisão é definida como $L_m(\mathcal{S}/\mathbf{G}) = M \cap L(\mathcal{S}/\mathbf{G})$, onde M é a linguagem definida como marcada pelo supervisor $\mathcal{S}$.

A planta, juntamente com o supervisor, forma um sistema em malha fechada, onde as mudanças de estados são ditadas pela ocorrência de eventos na planta $\mathbf{G}$ e a ação de controle $\mathcal{S}$, cuja função é desabilitar em $\mathbf{G}$ os eventos que não podem ocorrer em $\mathcal{S}$, após uma cadeia de eventos observada. O supervisor é dito ser não bloqueante para $\mathbf{G}$ se garantir o não bloqueio do sistema em malha fechada, ou seja, $\overline{L_m(\mathcal{S}/\mathbf{G})} = L(\mathcal{S}/\mathbf{G})$. Um supervisor $\mathcal{S}$ pode ser representado por um autômato $\mathbf{S}$ e uma função que associa cada estado de $\mathbf{S}$ a um conjunto de eventos desabilitados, de modo que $L(\mathcal{S}/\mathbf{G}) = L(\mathbf{S}||\mathbf{G})$ e $L_m(\mathcal{S}/\mathbf{G}) = L_m(\mathbf{S} \,||\, \mathbf{G})$.

O comportamento desejado é especificado na forma de uma linguagem-alvo K. Uma linguagem $K \subseteq L_m(\mathbf{G})$ é controlável em relação a $\mathbf{G}$ se $\overline{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K}$. Isso significa que a ocorrência de um evento não-controlável e fisicamente possível, após qualquer cadeia de $\overline{K}$, mantém a sequência no conjunto $\overline{K}$. Pode-se provar que existe um supervisor não bloqueante $\mathcal{S}$ tal que $L_m(\mathcal{S}/\mathbf{G}) = K$ se e somente se K é controlável. $\mathcal{C}(K,\mathbf{G})$ é o conjunto de todas as sublinguagens de K controláveis em relação a $\mathbf{G}$. Existe um elemento supremo, chamado $Sup\mathcal{C}(K,\mathbf{G})$, que representa a máxima sublinguagem de K que é controlável em relação a $\mathbf{G}$. Quando K não é controlável, $Sup\mathcal{C}(K,\mathbf{G})$ representa o comportamento menos restritivo possível de ser imposto por supervisão sobre $\mathbf{G}$ de modo a respeitar K. No caso da restrição da linguagem $Sup\mathcal{C}(K,\mathbf{G})$ não ser aceitável, diz-se que o problema de controle não tem solução.

Geralmente os supervisores são representados por autômatos com um número grande de estados, o que pode tornar inviável a sua implementação em CLPs. Um supervisor reduzido deve ter exatamente a mesma ação de controle que o supervisor original, mas estruturalmente menor. Pode-se dizer que dois supervisores $\mathcal{S}_1$ e $\mathcal{S}_2$ são equivalentes se suas ações de controle sobre a planta global $\mathbf{G}$, produzirem o mesmo comportamento. Os supervisores $\mathcal{S}_1$ e $\mathcal{S}_2$ podem ser representados pelos autômatos $\mathbf{S}_1$ e $\mathbf{S}_2$, a equivalência é verificada quando $L(\mathbf{S}_1/\mathbf{G}) = L(\mathbf{S}_2/\mathbf{G})$ e $L_m(\mathbf{S}_1/\mathbf{G}) = L_m(\mathbf{S}_2/\mathbf{G})$. Su e Wonham (2004) apresentam um algoritmo para a redução dos supervisores que

é implementado na ferramenta computacional TCT que foi utilizada neste trabalho.

Nesse exemplo da Estação de Separação, a especificação K não é controlável e $Sup\mathcal{C}(K,\mathbf{G})$ é modelado por um autômato ($\mathbf{S}$) de 396 estados e 2205 transições, que consiste em um supervisor não bloqueante e minimamente restritivo, de forma a garantir o cumprimento das especificações de controle. A redução do supervisor monolítico é realizada através do comando *"Supreduce"* do TCT, gerando um supervisor reduzido ($\mathbf{S}_{red}$) de 128 estados e 1050 transições.

2.3.1 Controle modular local

Na abordagem modular local, ao invés de projetar um único supervisor que satisfaça todas as especificações, procura-se construir um supervisor local para cada uma delas, de forma que, atuando em conjuntos, os supervisores locais satisfaçam a especificação global. A proposta de controle modular de Wonham e Ramadge (1988) é estendida por Queiroz e Cury (2000) de forma que seja possível implementar os supervisores, a partir de modelos de menor tamanho. Essa abordagem permite explorar além da modularidade das especificações, a modularidade da planta, de forma a diminuir a complexidade computacional da síntese de supervisores e o tamanho das soluções.

Queiroz e Cury (2000) definem que cada subsistema é representado por um autômato $\mathbf{G}_i = (Q_i,\Sigma_i,\delta_i,q_{0i},Q_{mi})$, $i \in I = \{1,...,n\}$, sendo n o número de subsistemas. Uma especificação genérica representa uma restrição de forma independente da planta e cada uma delas pode ser representada por um autômato $\mathbf{E}_{gen,j}$ e definida sobre $\Sigma_{gen,j} \subseteq \Sigma$ para $j \in J = \{1,...,m\}$, sendo m o número de especificações. A planta local é composta pelos subsistemas $\mathbf{G}_i$ que estão sendo restringidos pela especificação $\mathbf{E}_{gen,j}$. Cada planta local para $\{j = 1,...,m\}$ é representada por $\mathbf{G}_{loc,j} = ||_{i \in I_{loc,j}} \mathbf{G}_i$ com $I_{loc,j} = \{k \in I | \Sigma_k \cap \Sigma_{gen,j} \neq \emptyset\}$. Logo, as plantas locais referentes a cada especificação genérica são obtidas realizando o produto síncrono de todos os autômatos que compartilham algum evento em comum com a especificação em questão, isto é, $\Sigma_{gen,j} \cap \Sigma_i \neq \emptyset$. Dessa forma, o comportamento desejado para cada subsistema da planta global pode ser expresso por um conjunto de restrições locais $K_{loc,j}$, obtidas pela composição síncrona de cada especificação genérica com a sua planta local correspondente $K_{loc,j} = L_m(\mathbf{E}_{gen,j}||\mathbf{G}_{loc,j})$.

Os supervisores modulares locais $\mathcal{S}_{loc,j}$, $\{j = 1,..m\}$ são obtidos calculando as máximas linguagens controláveis contida na restrição local $Sup\mathcal{C}(K_{loc,j},\mathbf{G}_{loc,j})$, sendo m o número de supervisores locais, os quais atuando em conjunto, coordenam o comportamento do sistema. Cada supervisor é, então, projetado com uma visão parcial da planta e, também, com a restrição local, e não se tem informação das outras restrições implementadas pelos outros supervisores. Portanto, pode ser o caso das restrições

implementarem ações conflitantes, e para garantir que o sistema sob ação do conjuntos dos supervisores locais não apresente bloqueio, deve-se realizar a verificação da modularidade local (teste de conflito), que consiste em realizar a composição síncrona de todos os supervisores modulares locais, chamado de teste de modularidade. Caso o autômato resultante dessa composição não tenha estados bloqueantes (seja trim), então o Teorema 7 apresentado em Queiroz e Cury (2000) afirma que os supervisores locais são ótimos e têm o mesmo desempenho em relação ao controle monolítico. Caso contrário, há conflito entre os supervisores e, como consequência, não é mais garantida a propriedade de não bloqueio do sistema, sob ação dos supervisores modulares locais.

Os passos para projetar os supervisores através da abordagem modular local, de um conjunto de subsistemas $\{G_i,\ i = 1,...,n\}$ e um conjunto de especificações genéricas $\{E_{gen,j},\ j = 1,..,m\}$ são resumidos a seguir:

1. obter as plantas locais $\mathbf{G}_{loc,j}$, para $\{j = 1,...,m\}$;
2. obter as restrições locais $K_{loc,j}$, para $\{j = 1,...,m\}$;
3. obter a máxima linguagem controlável $Sup\mathcal{C}(K_{loc,j},\mathbf{G}_{loc,j})$, para $\{j = 1,...,m\}$;
4. verificar o comportamento em conjunto dos supervisores locais $\mathcal{S}_{Loc,j}$ (teste de conflito); e
5. se os supervisores locais forem modulares, implementar um supervisor local para cada linguagem controlável. Caso contrário, resolver o problema de conflito.

A seguir, será descrito cada um dos passos apresentados anteriormente no contexto da aplicação da Estação de Separação.

2.3.2 Síntese de supervisores para a Estação de Separação

Para a realização da síntese dos supervisores não temporizados, foram utilizadas as ferramentas de computação IDES (RUDIE, 2006) e TCT (FENG; WONHAM, 2006).

Plantas locais: primeiramente, obtêm-se as plantas locais $\mathbf{G}_{loc,j}$, por meio da composição dos autômatos dos subsistemas que são afetados pelas especificações genéricas $E_{gen,j}$, ou seja, é realizada a composição síncrona dos subsistemas que compartilham, ao menos, um evento com a especificação. As plantas locais geradas para cada especificação estão demonstradas na Tabela 2.

Como se pode observar na Tabela 2, as plantas locais $\mathbf{G}_{loc,2}$, $\mathbf{G}_{loc,5}$ e $\mathbf{G}_{loc,7}$ são iguais e representam a composição síncrona dos subsistemas de entrada $\mathbf{G}_2$ e deslocamento $\mathbf{G}_4$.

Tabela 2 – Plantas Locais $\mathbf{G}_{loc,j}$

Especificação	Planta Local	Subsistemas
E_1	$\mathbf{G}_{loc,1}$	$\mathbf{G}_1 \| \mathbf{G}_4$
E_2	$\mathbf{G}_{loc,2}$	$\mathbf{G}_2 \| \mathbf{G}_4$
E_3	$\mathbf{G}_{loc,3}$	$\mathbf{G}_3 \| \mathbf{G}_4$
E_4	$\mathbf{G}_{loc,4}$	$\mathbf{G}_4$
E_5	$\mathbf{G}_{loc,5}$	$\mathbf{G}_2 \| \mathbf{G}_4$
E_6	$\mathbf{G}_{loc,6}$	$\mathbf{G}_5 \| \mathbf{G}_4$
E_7	$\mathbf{G}_{loc,7}$	$\mathbf{G}_2 \| \mathbf{G}_4$

Fonte: Elaborado pelo autor (2021).

Restrições locais: no segundo passo, obtém-se uma restrição $K_{loc,j}$ para cada planta local, onde é realizada a composição síncrona de cada uma delas, com a especificação genérica em comum ($K_{loc,j} = L_m(\mathbf{G}_{loc,j} \| \mathbf{E}_{gen,j})$).

Máxima linguagem controlável: no terceiro passo são calculadas as máximas linguagens controláveis $Sup\mathcal{C}(K_{loc,j}, \mathbf{G}_{loc,j})$ para cada planta local. Para cada especificação $E_{gen,j}$ pode-se, então, construir um supervisor não-bloqueante $\mathcal{S}_{loc,j} = Sup\mathcal{C}(K_{loc,j}, \mathbf{G}_{loc,j})$ de forma que, ao desabilitar eventos controláveis da respectiva planta local, gera-se, em malha fechada, a máxima linguagem controlável obtida. Por meio do comando *"Supcon"* do TCT foram gerados 7 supervisores locais de 13 a 48 estados, sendo todos não bloqueantes.

A Tabela 3 mostra, de forma resumida, o número de todos os estados dos autômatos gerados para a solução do problema.

Teste de não conflito dos supervisores: Para garantir que não haja conflito entre as ações dos supervisores locais em conjunto, verifica-se o comportamento desses, a fim de garantir que a ação do conjunto de todos os supervisores seja controlável e não-bloqueante. Assim, após a síntese dos 7 supervisores locais, realizou-se o teste da modularidade local por meio da composição síncrona de todos os supervisores locais, resultando em um autômato *trim* (livre de bloqueios) e idêntico ao supervisor global $\mathbf{S} = \mathbf{S}_{loc,1} \| \mathbf{S}_{loc,2} \| \mathbf{S}_{loc,3} \| \mathbf{S}_{loc,4} \| \mathbf{S}_{loc,5} \| \mathbf{S}_{loc,6} \| \mathbf{S}_{loc,7}$ representado por um autômato de 396 estados e 2205 transições. Logo, os supervisores locais são ótimos (equivalentes ao monolítico).

Supervisores reduzidos: pode-se observar na Tabela 3 que, para os casos onde a restrição local $K_{loc,j}$ é igual ao supervisor local $\mathcal{S}_{loc,j}$, pode ser usada a própria especificação genérica $E_{gen,j}$ como o supervisor local reduzido $\mathcal{S}_{red,j}$, pois terá a mesma ação de controle sobre a planta, mas com um número consideravelmente menor de estados. Assim, os autômatos das especificações 2,3,4,5,6,7 podem ser diretamente adotados como supervisores locais reduzidos, sem a necessidade de usar

Tabela 3 – Número de estados dos autômatos - síntese do CML

j	$E_{gen,j}$	$G_{loc,j}$	$K_{loc,j}$	$S_{loc,j}$	$S_{red,j}$
1	2	24	48	36	2
2	3	24	26	26	3
3	2	24	24	24	2
4	2	12	13	13	2
5	3	24	48	48	3
6	2	24	24	24	2
7	2	24	13	13	2

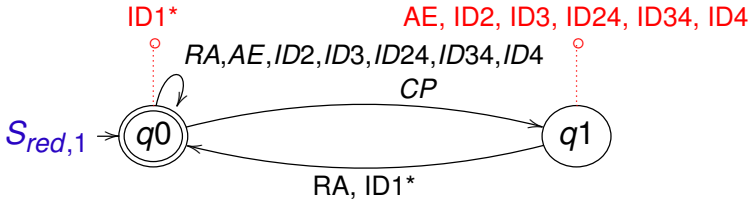
Fonte: Elaborado pelo autor (2021).

Emque :
$K_{loc,j}$ = restrição local;
$E_{gen,j}$ = especificação genérica;
$G_{loc,j}$ = planta local;
$S_{loc,j}$ = supervisor local;
$S_{red,j}$ = supervisor local reduzido.

um algoritmo de redução. Para a especificação $E_{gen,1}$, tem-se que $K_{loc,1} \neq S_{loc,1}$, logo esta especificação não pode ser usada como supervisor reduzido e fez-se, assim, necessário o uso do algoritmo de redução através do comando "*SupRed*"do TCT.

Da mesma forma que para as especificações, adota-se a simbologia por meio de uma linha pontilhada para ilustrar os eventos que devem ser desabilitados e, assim, impedidos de ocorrer num determinado estado do supervisor. Como pode ser observado na Figura 17, para o supervisor reduzido 1, no estado $q0$, os eventos que devem ser desabilitados são todos os inícios de deslocamentos $ID1^*$ e, no estado $q1$, os eventos AE, $ID2$, $ID3$, $ID24$, $ID34$, $ID4$.

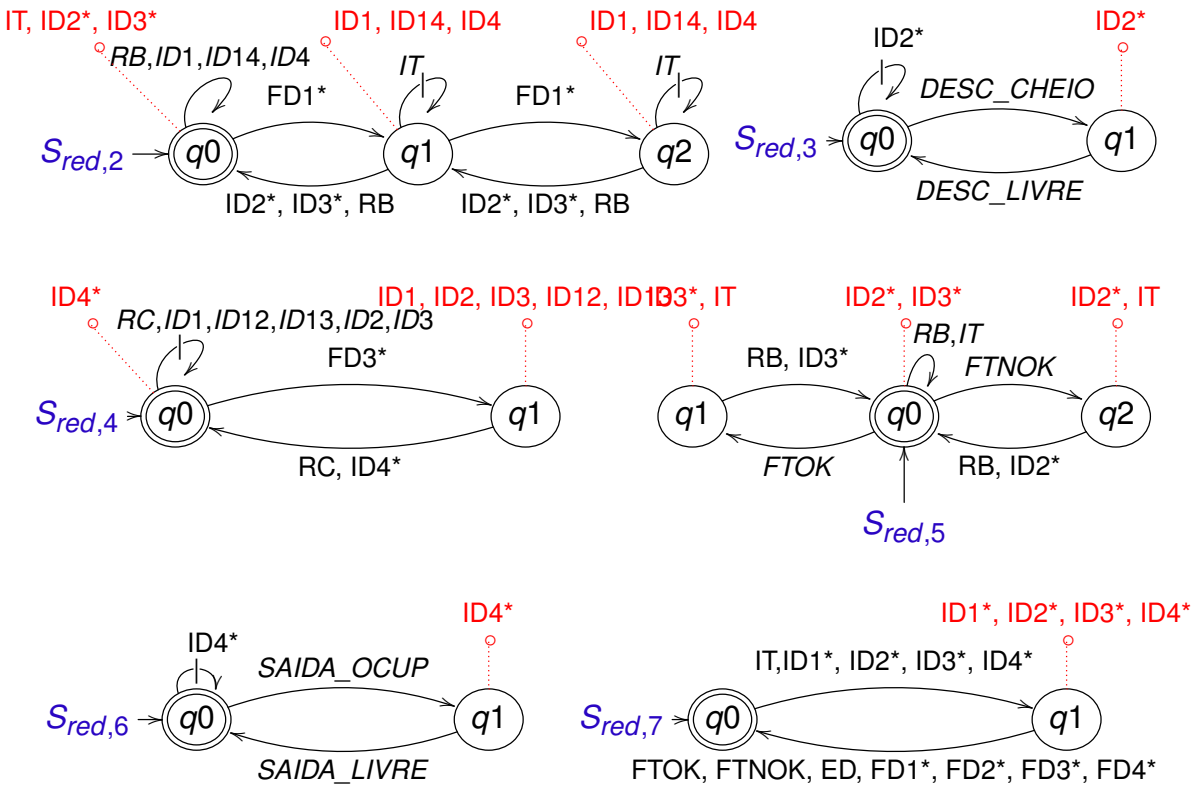
Figura 17 – Supervisor Local Reduzido $S_{red,1}$



Fonte: Elaborado pelo autor (2021).

A Figura 18 apresenta os autômatos empregados na representação reduzida dos supervisores locais restantes.

Figura 18 – Supervisores Locais Reduzidos



Fonte: Elaborado pelo autor (2021).

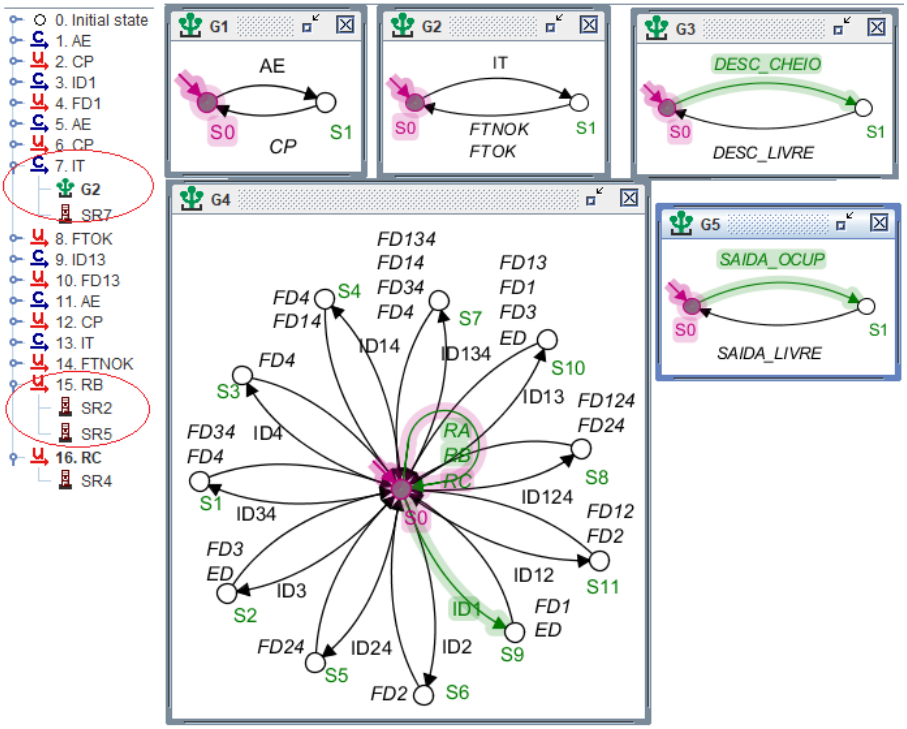
2.3.3 Simulação da Estação de Separação

A utilização de uma ferramenta de emulação permite verificar o comportamento do sistema em vários cenários possíveis e, portanto, identificar os prováveis erros de modelagem. O modelo encontrado foi testado e emulado por meio da ferramenta computacional "Supremica"(AKESSON, 2002) e modificado de forma a ficar em conformidade com a lógica proposta no projeto informacional. Para o problema proposto, foi necessário modificar e inserir novas especificações, como por exemplo, criou-se uma nova especificação de exclusão mútua entre os eventos de início (*IT*) e os inícios de deslocamento (*IDi*), uma vez que foi possível verificar na simulação que isso poderia levar o sistema a um comportamento indesejado.

A Figura 19 apresenta a emulação da lógica de controle supervisório, sendo executada em um cenário onde o sistema opera com três peças em paralelo, ou seja, com uma peça na entrada, uma peça testada e outra peça na saída, como pode ser observado pela sequência de eventos gerada, em que os passos “2”, “6”, “12” indicam a chegada de peças no sistema e os passos “8” e ”14”, o fim de teste. No passo “14”, o evento *ID*124 está habilitado. Após a ocorrência de duas falhas, em decorrência da intervenção humana (retirada de peças dos *buffer*s B e C) indicadas pelos passos

"15" e "16", o controlador atua desabilitando o evento *ID*124 e habilitando o evento *ID*1, mantendo em conformidade a lógica proposta no projeto informacional. Pode ser observado ainda no passo "7", a exclusão mútua entre o *IT* e o *IDi* realizada pela atuação do supervisor 7. Logo, o sistema não pode iniciar nenhum deslocamento, enquanto o teste está sendo realizado e vice-versa.

Figura 19 – Simulação Cenário 1 – Sistema operando com três peças em paralelo



Fonte: Elaborado pelo autor (2021).

2.4 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram apresentados de forma reduzida os principais conceitos da TCS e da abordagem do CML. Esses conceitos foram aplicados a um problema real de controle e a resolução do exemplo foi utilizada para a construção de um artigo, o qual foi apresentado no Congresso Internacional *Fluid Power Net International* (SZPAK; QUEIROZ, 2016).

Tem-se como uma primeira contribuição desta pesquisa de doutorado, a modelagem de um problema real, em que são colocadas questões relevantes para sistemas eletropneumáticos, uma vez que não existem muitos trabalhos de modelagem e implementação do CML em sistemas reais. Mostrou-se, nesta modelagem, que muitas vezes um problema que pode ser trivial, como o sequenciamento de tarefas, passa a não ser, quando se considera o tratamento de falhas e tarefas concorrentes e, nesse contexto, o controle supervisório encontra uma lógica que trata o problema. O supervisor monolítico encontrado é representado por um autômato com 396 estados e 2.205

transições e a lógica de controle modular equivalente é modelada por um conjunto de 12 autômatos, com 2 a 12 estados, o que torna essa solução de fácil compreensão e mais flexível para mudanças. O tratamento de falhas com eventos observáveis e não-controláveis, embora apresente modelos simples, ainda permite o projeto de uma lógica capaz de lidar com ocorrências esporádicas de falhas, de forma minimamente restritiva. No próximo capítulo, esses modelos serão implementados em um CLP para realizar o controle do sistema real.

3 IMPLEMENTAÇÃO DO SISTEMA DE CONTROLE PARA SISTEMAS A EVENTOS DISCRETOS NÃO TEMPORIZADOS

No capítulo 2 a TCS foi introduzida e aplicada ao exemplo motivador da Estação de Separação. Este capítulo aborda a implementação da TCS para SEDs em CLPs e apresenta uma estrutura de controle que permite realizar a implementação em um sistema real. Assim, a seguir encontra-se, de forma resumida, uma revisão bibliográfica, referente aos problemas que podem ocorrer na implementação, bem como seus tratamentos. Na Seção 3.2 discorre-se sobre a estrutura de implementação de supervisores modulares definida por Queiroz e Cury (2002); a proposta para o desenvolvimento integrado de sistemas *SCADA* de Portilla, Queiroz e Cury (2014); e, por fim, a estrutura de inicialização de Vieira (2007). Na Seção 3.3 mostra-se como essas estruturas são aplicadas com sucesso ao controle supervisório da Estação de Separação obtido no Capítulo 2. E, na sequência, a Seção 3.4 traz a conclusão deste capítulo.

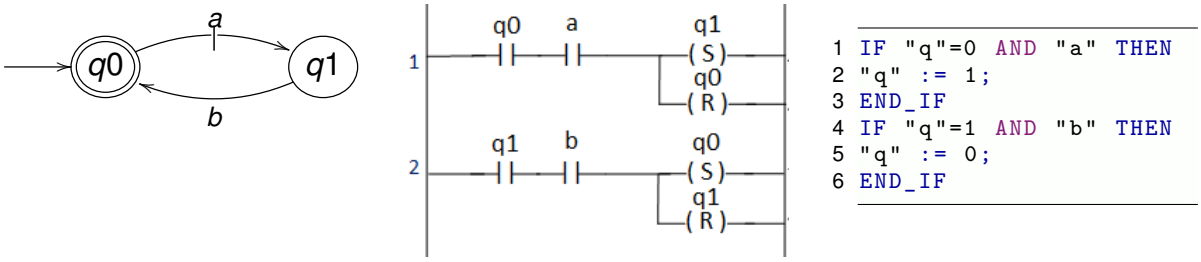
3.1 ASPECTOS RELATIVOS À IMPLEMENTAÇÃO DA TCS

É possível observar na literatura que autores, como Hellgren, Fabian e Lennartson (2005), Skoldstam, Akesson e Fabian (2007), Zaytoon e Riera (2017), Prenzel e Provost (2018), compartilham da mesma opinião, que a TCS, apesar de ter recebido uma grande aceitação no meio acadêmico, carece de aplicações industriais. Hellgren, Fabian e Lennartson (2005) argumentam que um dos principais obstáculos para isso é o problema da implementação física e salientam que existem poucas referências de como implementar os supervisores obtidos através da aplicação da TCS. No caso especial de sistemas de manufatura, onde o controle via CLP é de grande importância, existe uma lacuna entre o mundo de autômatos baseados em eventos e o mundo de CLPs baseados em sinais.

Conforme Fabian e Hellgren (1998), um supervisor é normalmente descrito por uma máquina de estados finito, de modo que sua implementação é uma questão de fazer com que o CLP se comporte como uma máquina de estados. Segundo os autores, uma maneira simples de implementar uma máquina de estados em *Ladder* (LD) é representar por meio de variáveis booleanas internas para cada estado e cada evento e, também, representar as transições por dois contatos em série (*AND*), entre as variáveis de estado e evento. Quando uma transição ocorre, o próximo estado é setado e o estado anterior é resetado. A Figura 20 ilustra o código de programação em *Ladder* (LD) e *Texto Estruturado* (ST) para a implementação de uma máquina de estados, conforme definido por Fabian e Hellgren (1998). Para os autores, os eventos controláveis representam comandos e o eventos não-controláveis as respostas do sistema. Dessa forma, a ocorrência de um evento não-controlável (*b*), é detectada por uma variável booleana de entrada (*b*). Já a geração de um evento controlável (*a*)

resulta na ativação de uma variável booleana de saída (*a*) e cada estado é, também, associado a uma variável booleana interna (*q*0, *q*1). Para a evolução dinâmica dessa máquina de estados, uma transição ocorre sempre que os contatos das variáveis que representam o estado anterior e um evento interno (controlável) ou externo (não-controlável) estiverem ativos. Na sequência, o estado posterior será setado e o estado anterior resetado. Assim, é possível observar na Figura 20, que, se o sistema estiver no estado inicial (variável *q*0 := 1) e o evento *a* ocorrer (variável *a* := 1), a transição *a* da máquina de estado ocorrerá, setando o estado *q*1 e resetando o estado *q*0.

Figura 20 – Implementação em diagrama Ladder e Texto Estruturado para um autômato de 2 estados



Fonte: Adaptado de Fabian e Hellgren (1998).

Alguns dos principais problemas que surgem na implementação de arquiteturas de controle supervisório e que serão explorados a seguir, são definidos por Fabian e Hellgren (1998) e divididos em quatro classes: sinais e eventos; causalidade; escolha; e sincronização inexata.

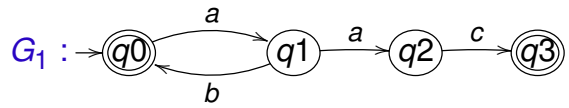
3.1.1 Sinais e eventos

A TCS considera que os eventos possuem valores simbólicos e estes ocorrem assincronamente em intervalos de tempo discreto. Já os CLPs lidam com sinais booleanos, cujos valores são analisados de forma síncrona, em que diversos eventos podem ocorrer simultaneamente (num mesmo ciclo de varredura do CLP) e violando, assim, um pressuposto fundamental da TCS (Fabian e Hellgren (1998)). Durante cada ciclo de varredura, o CLP executa sequencialmente três tarefas: leitura e armazenamento dos valores lógicos das variáveis de entrada em sua memória interna; execução do programa; e atualização das saídas. O tempo que o CLP leva para executar essas três tarefas é denominado tempo de varredura ou *"Scan Time"* que, geralmente, é da ordem de milissegundos (até microssegundos em CLPs de última geração). Conforme Vieira (2007), definem-se, nesta categoria, os problemas de efeito avalanche e a incapacidade de reconhecer a ordem de eventos.

Efeito avalanche: este problema pode acontecer quando uma única ocorrência de um evento é registrada mais de uma vez no mesmo ciclo de varredura do CLP,

em que a ocorrência de um único evento pode gerar várias transições de estados do sistema, o que resulta na quebra de sincronia entre a planta (sistema real) e o supervisor implementado. Assim, as ações de controle subsequentes, possivelmente, não serão corretas. Isso pode ocorrer, principalmente, se um evento específico for usado para acionar sucessivas transições de estados. A Figura 21 ilustra o autômato de uma planta em que é fácil observar que: quando o estado $q0$ está ativo, uma única ocorrência do evento *a* pode levar o sistema de controle a transitar do estado $q0$ para o estado $q2$ no mesmo ciclo de varredura, gerando o efeito avalanche. Porém, como o evento *a* ocorreu uma única vez, o processo (sistema real) estará ativo no estado $q1$. Assim, para essa planta, se o efeito avalanche ocorrer, a ação associada ao estado $q1$ não será executada.

Figura 21 – Autômato sujeito ao problema de efeito avalanche



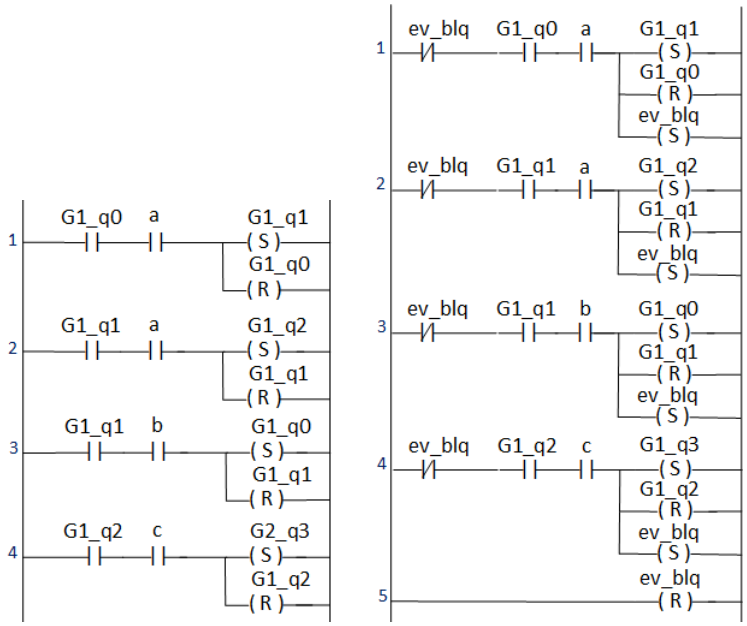
Fonte: Elaborado pelo autor (2021).

Para ilustrar esse problema, a Figura 22a apresenta um código que pode levar ao problema do efeito avalanche, em que é possível observar que a ocorrência do evento *a* faz com que o programa execute, no mesmo ciclo de varredura, os degraus 1 e 2, e por consequência, ative a variável *G*1_*q*2 que representa o estado $q2$.

Fabian e Hellgren (1998) propõem como solução uma estrutura em que os estados que possuem a transição com o mesmo evento devem ser implementados em ordem reversa. No caso da Figura 22a, a ordem de execução dos degraus deveria ser 2, 1, 3, 4. Porém, segundo Hasdemir, Kurtulan e Gören (2008), essa solução pode não funcionar para alguns autômatos e, nesse caso, deve ser aplicada apenas para aqueles que não apresentem ciclos na estrutura de transição. Os autores propõem uma metodologia de implementação que resolve o problema de efeito avalanche, em que as funções SET e RESET do CLP não são utilizadas, reduzindo, em muitos casos, a memória do programa.

Em Queiroz e Cury (2002) propõe-se como solução que duas transições seguidas não podem ocorrer na planta, sem que os supervisores tenham sido atualizados. Já Vieira (2007) propõe como solução, associar uma variável do CLP de tal modo que essa variável seja ativada sempre que ocorrer uma transição de estado e desativada no final do código. A Figura 22b ilustra a solução para essa proposta, em que a variável *Ev_Blq* permite a ocorrência de apenas uma transição a cada ciclo de varredura do CLP. No final do código essa variável deve ser resetada. Portanto, é fácil observar que a ocorrência do evento *a*, a partir do estado $q0$, leva o sistema para o estado $q1$, eliminando o problema do efeito avalanche.

Figura 22 – Problema de Efeito Avalanche -Implementação Autômato *G*



(a) Sujeito ao Efeito Avalanche (b) Imune ao Efeito Avalanche

Fonte: Adaptado de Vieira (2007)

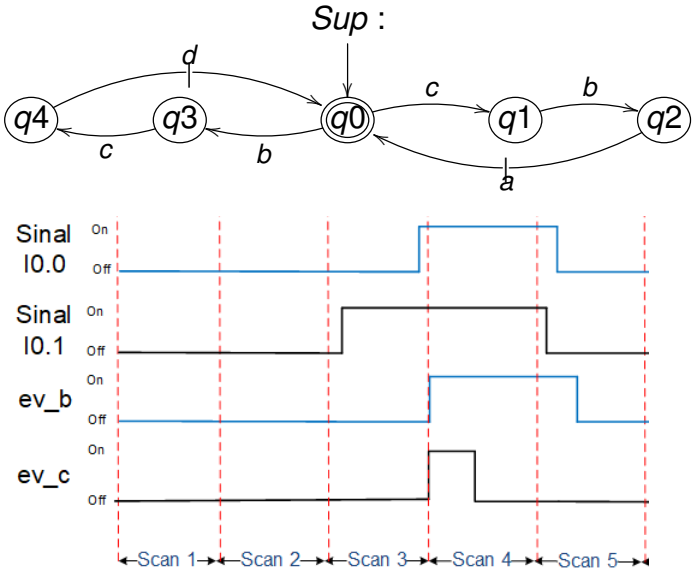
Leal, Cruz e Hounsell (2012) propõem outra solução, que trata o problema do efeito avalanche para a ocorrência de mais de um evento por ciclo de varredura do CLP. O método consiste em tratar todos os eventos não-controláveis, criando dois grupos de memórias auxiliares para o tratamento de cada evento, em que o primeiro grupo é usado para armazenar os eventos gerados pela planta, e o segundo para habilitar as transições de estados do sistema produto e dos supervisores.

Incapacidade de conhecer a ordem de eventos: a TCS define que dois eventos não podem ocorrer simultaneamente, porém, os sinais do CLP podem se sobrepor no tempo. Esse problema ocorre devido a forma que o CLP faz a leitura dos seus sinais de entrada, que é realizada apenas no início de varredura de cada ciclo. Portanto, se ocorrer uma mudança de mais de um sinal de entrada no mesmo ciclo de varredura, o CLP é incapaz de reconhecer qual mudança ocorreu primeiro.

A insensibilidade ao entrelaçamento proposta em Fabian e Hellgren (1998) é uma propriedade que evita que um supervisor adote uma ação de controle incorreta, em função de não reconhecer a ordem de ocorrência dos eventos não-controláveis ativos, no mesmo ciclo de varredura do CLP. Essa propriedade verifica se todas as sequências de eventos não-controláveis consecutivos, levam a mesma ação de controle, independentemente da ordem de observação. Segundo o autor, para que esse problema não ocorra, o sistema deve ter a propriedade da insensibilidade ao entrelaçamento.

A Figura 23, ilustra o problema apresentado, em que a ordem dos eventos não-controláveis *b* e *c* podem levar a diferentes estados com diferentes ações de controle. Pode-se observar que o sinal físico *I*0.1, referente ao evento *c* ocorre primeiro (*scan*3). Dessa forma, o supervisor deveria seguir o caminho *q*1-*q*2 e ativar o evento *a*. Porém, como esses dois eventos estarão ativos, ao mesmo tempo, no início do *scan*4 e como o CLP não tem como reconhecer essa ordem, isso poderia gerar um erro de controle, onde ele executaria primeiro o evento *b* (caminho *q*3-*q*4) e ativaria o evento *d*.

Figura 23 – Autômato sujeito ao Problema de *Interleaving Insentivy*



Fonte: Elaborado pelo autor (2021).

Essa propriedade é formalmente declarada, conforme Fabian e Hellgren (1998). Seja Σ^* o conjunto de todas as sequências finitas (cadeias) sobre Σ, o entrelaçamento entre duas cadeias $s_1, s_2 \in \Sigma^*$, é um conjunto de cadeias obtido pela concatenação de um prefixo de uma das cadeias com um prefixo da outra, concatenada com o entrelaçamento dos sufixos remanescentes. Toda a intercalação de s_1, s_2, é uma linguagem denotada por $s_1|||s_2$ e $s_1, s_2 \in s_1|||s_2$.

Exemplo 3.1: O conjunto de sequências denotado por $s_1 \;|||\; s_2$ é a linguagem obtida pelos possíveis entrelaçamentos de s_1 e s_2. Sejam $s_1 = abc$ e $s_2 = xy$ duas sequências de eventos, então $s_1|||s_2 = \{abcxy, abxyc, abxcy, axybc, axbyc, axbcy, xyabc, xaybc, xabyc, xabcy\}$.

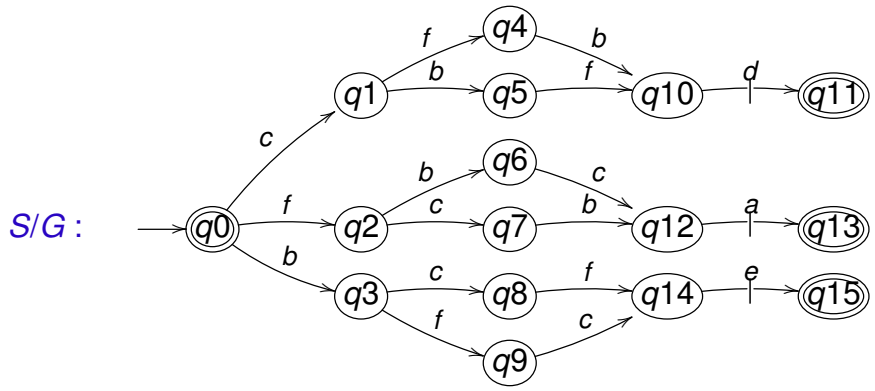
Para definir formalmente o problema levantado anteriormente, tem-se em Fabian e Hellgren (1998) a propriedade 1, em que um supervisor $\mathcal{S}$ é insensível ao entrelaçamento com relação a uma planta **G** (definida sobre um alfabeto $\Sigma = \Sigma_u \cup \Sigma_c$) e um subalfabeto Σ_c, se para $s \in \Sigma^*$, $s_1, s_2 \in \Sigma_u^*$ e $\sigma \in \Sigma_c$:

$$s.s_1.s_2.\sigma \in L(\mathcal{S}/\mathbf{G}) \Rightarrow s.(s1|||s2).\sigma \in L(\mathcal{S}/\mathbf{G}) \quad (1)$$

Analisando essa propriedade para o exemplo da Figura 23, em que $s = \varepsilon \in \Sigma^*$, $s_1 = b, s_2 = c \in \Sigma_u^*$, $\sigma = d \in \Sigma_c$, tem-se que: $s.s_1.s_2.\sigma = b.c.d \in L(\mathcal{S}/\mathbf{G})$, e $s.(s1|||s2).\sigma = \{b.c.d, c.b.d\}$. Assim, como $c.b.d \notin L(\mathcal{S}/\mathbf{G})$, então, o supervisor não é insensível ao entrelaçamento e sua implementação em CLP está sujeita a inconsistências na ação de controle sobre o evento d pela incapacidade de reconhecer a ordem dos eventos não-controláveis b e c.

Exemplo 3.2: Considerando a Figura 24, e para $s = \varepsilon \in \Sigma^*$, $s_1 = c, s_2 = f, s_3 = b \in \Sigma_u^*$, $\sigma = d \in \Sigma_c^*$. Tem-se que: $s.s_1.s_2.s_3.\sigma = c.f.b.d \in L(\mathcal{S}/\mathbf{G})$ e $s.(s_1|||s_2|||s_3)\sigma = \{c.f.b.d, c.b.f.d, b.c.f.d, b.f.c.d, f.c.b.d, f.b.c.d\}$.

Figura 24 – Supervisor sujeito ao problema de Interleaving Insensitivity



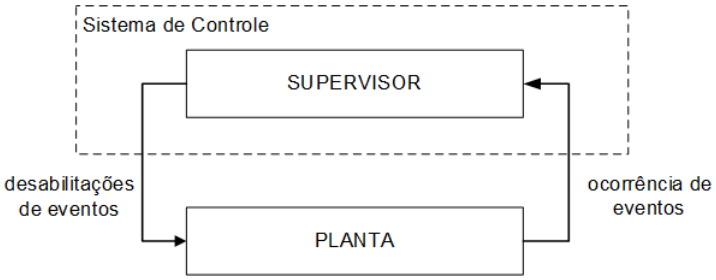
Fonte: Elaborado pelo autor (2021).

Portanto, o supervisor não é insensível ao entrelaçamento e sua ação de controle em CLP, também não é confiável, pois $f.c.b.d \notin L(\mathcal{S}/\mathbf{G})$.

3.1.2 Causalidade

A teoria de controle supervisório definida por RW assume que a planta gera todos os eventos e que o supervisor desabilita dinamicamente eventos que a planta poderia ter gerado. Entretanto, conforme apresentado em Balemi (1994), Malik (2002), essa suposição não é verdadeira para a maioria dos sistemas industriais e os eventos, geralmente, não ocorrem de forma espontânea, mas apenas como respostas a comandos do CLP. Os atuadores são ativados por comandos e correspondem aos eventos controláveis, enquanto as respostas correspondem aos eventos não-controláveis, representando as mudanças do sistema, detectadas por sensores. A Figura 25 ilustra a estrutura de controle proposta pela TCS, onde a planta é a responsável pela geração de todos os eventos (Σ_u e Σ_c), enquanto o supervisor tem a função de desabilitar os eventos controláveis para satisfazer aos requisitos de controle.

Figura 25 – Arquitetura de controle - Supervisor controlando a planta



Fonte: Adaptado de Malik (2002).

Assim conforme Fabian e Hellgren (1998), o conceito de causalidade não pode ser evitado na implementação e a seguinte pergunta deve ser respondida: Quem gera o quê?

Em Balemi (1994), uma abordagem entrada/saída é definida para tratar esse problema, em que é necessário que uma planta (**P**) seja completa para um supervisor ($\mathcal{S}$), de tal forma que a planta sempre seja capaz de aceitar os eventos controláveis gerados pelo supervisor. Essa condição é satisfeita se e somente se: $L(\mathbf{P}).\Sigma_c \cap L(\mathcal{S}) \subseteq L(\mathbf{P})$, isto é, se todo evento controlável que for habilitado pelo supervisor puder de fato ocorrer a partir daquele estado da planta. Cabe ressaltar que, para o caso geral, o autômato que representa um supervisor reduzido não satisfaz tal propriedade.

Conforme Hellgren, Lennartson e Fabian (2002), para implementações monolíticas, a divisão do conjunto em entradas e saídas é uma resposta suficiente. Entretanto, na implementação do controle supervisório modular é necessário determinar qual supervisor gerou o evento. Os autores dizem que não há uma solução direta para esse problema, como pode ser visto em artigo publicado anteriormente por Hellgren, Fabian e Lennartson (2001), em que também é sugerido que outros agentes, chamados de monitores de eventos, devem criar os eventos de saídas gerados pelos supervisores.

Uma solução introduzida por Queiroz e Cury (2002) para o problema da causalidade e adotada em diversos trabalhos subsequentes, como em Queiroz (2004); Texeira e Leal (2008); Leal, Cruz e Hounsell (2009); Pinotti (2012); Vieira *et al.* (2017), é dada pela utilização de uma interface entre supervisores e planta física.

Atendendo fielmente a hipótese de RW de que os eventos são espontaneamente gerados pela planta e de que o supervisor apenas desabilita eventos controláveis, Queiroz e Cury (2002) propõem que o Sistema Produto (SP) deve ser implementado no CLP. Nessa estrutura, os autômatos que representam os subsistemas assíncronos da planta são responsáveis pela geração dos eventos controláveis. Assim, todos os eventos são gerados no sistema produto e os supervisores mantêm a função original de apenas estabelecer as desabilitações. Essa arquitetura resolve o problema de

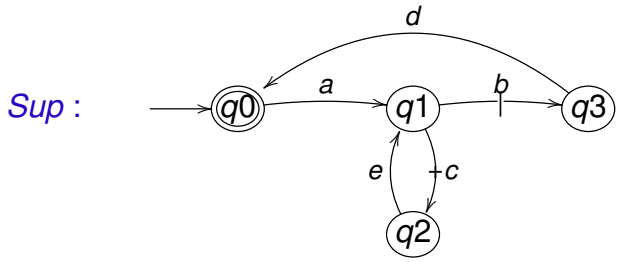
causalidade e será melhor detalhada na Seção 3.2.

3.1.3 Escolha

O problema de escolha ocorre quando se tem dois ou mais eventos controláveis e habilitados num dado estado. Conforme Fabian e Hellgren (1998), o supervisor obtido por intermédio da aplicação da TCS deve ser minimamente restritivo, de forma que permita uma maior liberdade possível na planta, de tal forma que as especificações ainda sejam satisfeitas, e caso mais de um evento controlável esteja habilitado num determinado estado do supervisor, apenas um evento deve ser escolhido e executado. Logo, dependendo da ordem de implementação interna do CLP, diferentes escolhas poderão ser feitas explicitamente pelo programador. Caso contrário, a execução sequencial do programa dentro do CLP irá fazê-la.

Em Hellgren, Lennartson e Fabian (2002) para que o problema de escolha não aconteça, os supervisores modulares devem ser determinísticos. Malik (2002) diz que, se a escolha de um evento for priorizada em relação a outro evento, parte do sistema poderá ser prejudicada e nunca ocorrer, de modo a poder ocorrer bloqueio no sistema. A Figura 26 ilustra esse problema. Pode-se observar que, sempre que o evento *b* for priorizado, o evento *c* nunca ocorrerá e, caso o evento *c* for priorizado, o sistema nunca alcançará um estado marcado, gerando o bloqueio do sistema.

Figura 26 – Autômato sujeito ao problema da escolha

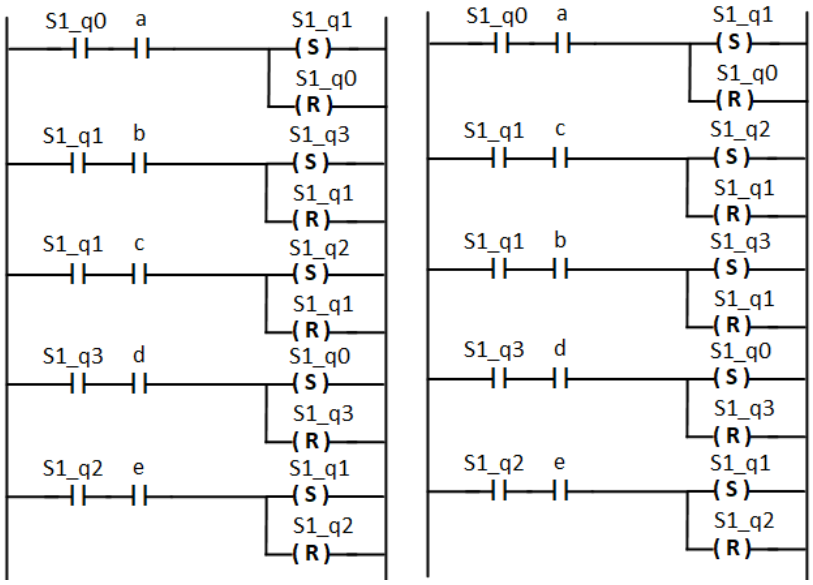


Fonte: Elaborado pelo autor (2021).

A Figura 27 mostra a implementação correspondente a um autômato com o problema da escolha. Pode-se observar que apenas alterando a posição dos degraus 2 e 3 tem-se uma prioridade diferente implementada. Assim, na Figura 27a, a implementação prioriza o evento *b* e, portanto, os eventos *c* e *e* nunca irão ocorrer. Na Figura 27b, a implementação prioriza o evento *c*, logo, tem-se o problema de bloqueio no sistema, uma vez que essa solução não leva a um estado marcado.

Hellgren, Lennartson e Fabian (2002) propõem a utilização de agentes denominados monitores de eventos, que oferece uma avaliação parcial em resposta ao problema de escolha, em que os monitores são calculados para que os eventos conflitantes de saída (gerados pelos supervisores) nunca sejam executados simultane-

Figura 27 – Implementação Problema de Escolha



(a) Prioridade evento *b* (b) Prioridade evento *c*

Fonte: Elaborado pelo autor (2021).

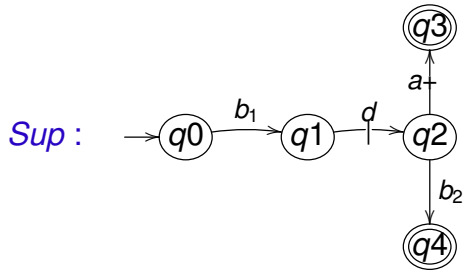
amente, mesmo que sejam ativados de forma síncrona. Cruz (2011), Leal, Cruz e Hounsell (2012) propõem uma solução que consiste em alternar uma variável booleana entre valores falso e verdadeiro para cada ciclo de varredura do CLP, permitindo que a variável gere, ora um evento, ora outro, de tal forma que a probabilidade de escolha, para cada um deles, seja igual e aleatória. Portilla, Queiroz e Cury (2014) propõem definir a escolha de eventos controláveis através do sistema *SCADA*, em que é possível fazer a escolha de eventos de forma que os caminhos habilitados não levem a bloqueio, resolvendo o problema de conflito em supervisores modulares. Já Scotti (2015) propõe realizar a escolha de comandos do controle supervisório, a partir de um Sistema de Roteamento de Tarefas (SRT), responsável pelas decisões de alocação e liberação de recursos do processo.

3.1.4 Sincronização inexata

Conforme Fabian e Hellgren (1998), a interação entre a planta e o supervisor pode ser prejudicada pela falta de sincronização entre a ocorrência do evento e o processamento da respectiva informação no CLP. Atrasos devido à atualização periódica da entrada de sinais podem significar que um supervisor implementado no CLP não seja executado em sincronia exata com a planta. Uma mudança de sinal pode ocorrer na planta, enquanto o programa está sendo executado, o que pode invalidar a escolha feita, uma vez que o supervisor pode ter conhecimento somente sobre os comandos e respostas emitidos até o momento da última atualização das entradas e saídas do CLP.

A Figura 28 mostra o autômato de um supervisor sujeito à sincronização inexata. Supondo que o estado ativo do supervisor seja $q2$, a ação de controle consiste na habilitação do evento a, porém, durante o tempo necessário para realizar essa ação de controle é possível que o evento não-controlável b_2 tenha ocorrido na planta. Como essa informação não será comunicada imediatamente ao supervisor, o evento a terá sido gerado de forma indevida, resultando na quebra de sincronia entre a planta e o controlador implementado no CLP.

Figura 28 – Supervisor Sujeito ao Problema da Sincronização Inexata



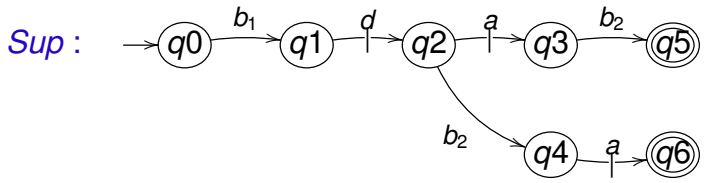
Fonte: Elaborado pelo autor (2021).

Para garantir que a linguagem gerada pelo autômato que modela o supervisor tenha a propriedade de ser insensível ao atraso, os mesmos devem satisfazer a *definição 3.9* (*delay insensitive language*) definida por Balemi Balemi (1992), como segue: uma linguagem K é dita ser insensível ao atraso, se para: $s \in \overline{K}$, $\sigma_c \in \Sigma_c$, $e\, \sigma_u \in \Sigma_u$.

$$\textit{quando } s.\sigma_c,\ s.\sigma_u \in \overline{K} \Rightarrow s.\sigma_c.\sigma_u, s.\sigma_u.\sigma_c \in \overline{K} \tag{2}$$

Logo, de acordo com essa propriedade, se nenhuma escolha feita pelo supervisor for invalidada pela ocorrência de um evento não-controlável, pode-se afirmar que o supervisor é insensível ao atraso (Figura 29), uma vez que, independentemente da ordem dos eventos, o supervisor pode seguir as cadeias formadas por $s.\sigma_c.\sigma_u$ e $s.\sigma_u.\sigma_c$, pois ambas são mapeadas pelo supervisor.

Figura 29 – Supervisor Insensível ao Atraso



Fonte: Elaborado pelo autor (2021).

Exemplo 3.3: Considerando os autômatos das Figuras 28 e 29 que representam uma linguagem sob supervisão para $\Sigma_c = a,d$, $\Sigma_u = b_1, b_2$, pode-se verificar que:

O autômato da Figura 28 não possui a propriedade de insensibilidade ao atraso, como segue.

Sendo $\overline{K} = \{\varepsilon, b_1, b_1.d, b_1.d.a, b_1.d.b_2\}$, e tomando $s = b_1.d$, $\sigma_c = a$ e $\sigma_u = b_2$, tem-se que: $s\sigma_c = b_1.d.a \in \overline{K}$; $s\sigma_u = b_1.d.b_2 \in \overline{K}$. Mas, $s.\sigma_u.\sigma_c = b_1.d.b_2.a \notin \overline{K}$; $s\sigma_c\sigma_u = b_1.d.a.b_2 \notin \overline{K}$. Logo a linguagem $K = \{b_1.d.b_2, b_1.d.a\}$ não é insensível ao atraso.

Já para o autômato da Figura 29, é possível verificar que o mesmo possui a propriedade insensibilidade ao atraso, como segue.

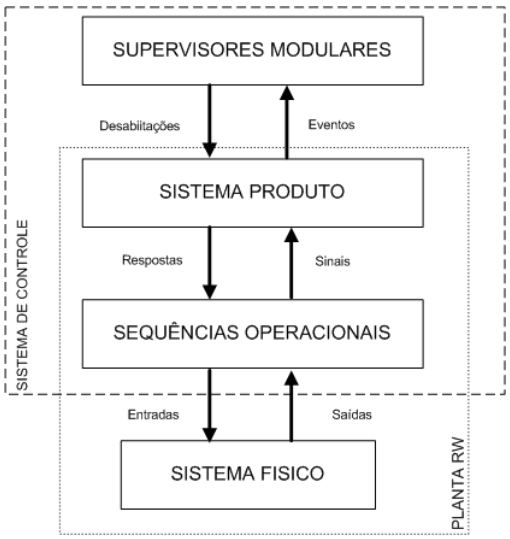
Sendo $\overline{K} = \{\varepsilon, b_1, b_1.d, b_1.d.a, b_1.d.b_2, b_1.d.a.b_2, b_1.d.b_2.a\}$, e tomando $s = b_1.d$, $\sigma_c = a$ e $\sigma_u = b_2$, tem-se que: $s\sigma_c = b_1.d.a \in \overline{K}$; $s\sigma_u = b_1.d.b_2 \in \overline{K}$; $s.\sigma_u.\sigma_c = b_1.d.b_2.a \in \overline{K}$; e $s\sigma_c\sigma_u = b_1.d.a.b_2 \in \overline{K}$. Logo a linguagem $K = \{b_1.d.b_2, b_1.d.a\}$ é insensível ao atraso.

Uma vez apresentados os problemas de implementação de sistemas de controle a SED, será aplicada, na próxima seção, a arquitetura de controle modular local para a implementação em CLPs do estudo de caso da Estação de Separação.

3.2 UM ARQUITETURA DE IMPLEMENTAÇÃO DO CONTROLE SUPERVISÓRIO

Nesta seção será apresentada, de forma resumida, a arquitetura de controle supervisório proposta por Queiroz e Cury (2002) e detalhada por Vieira *et al.* (2017), composta por três níveis, como pode visto na Figura 30. Essa arquitetura de controle trata de alguns dos problemas de implementação apresentados na Seção 3.1 e baseia-se em uma interface de controle entre o sistema físico e os supervisores modulares, calculados pelos procedimentos de síntese.

Figura 30 – Arquitetura de Controle Supervisório



Fonte: Queiroz e Cury (2002).

Segundo Queiroz e Cury (2002), essa estrutura é proposta buscando-se implementar uma interface entre o sistema real e os supervisores modulares, obtidos a partir de um modelo abstrato do sistema, divididos em: Supervisores Modulares (SM); Sistema Produto (SP); e Sequências Operacionais (SO). A função é fazer com que o sistema real se comporte de acordo com a planta RW.

Supervisores Modulares: neste nível são implementados os supervisores modulares reduzidos. Têm-se como entrada os eventos controláveis gerados pela planta e, como saída, a desabilitação de eventos controláveis. Um mapa de retroalimentação associa os estados ativos a um conjunto de desabilitações de eventos controláveis. Nesse nível, os eventos desabilitados pelos supervisores determinam o funcionamento da planta abstrata, definida pela representação por Sistema-Produto. As transições dos supervisores que entram e saem no mesmo estado (*'auto-laços'*), não são implementadas, pois não alteram os estados ativos do supervisor e, portanto, não alteram a ação de controle do sistema.

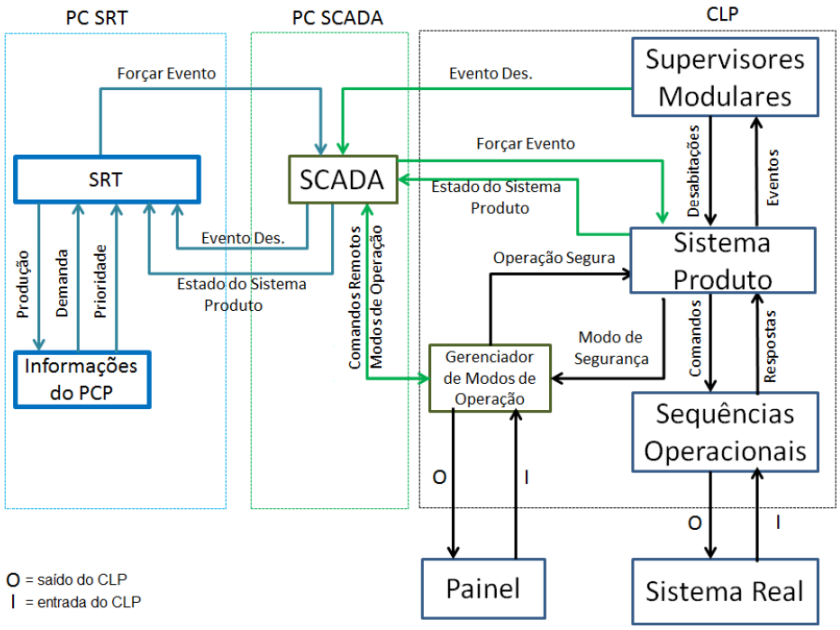
Sistema Produto: neste nível, implementam-se os modelos dos subsistemas que compõem a planta na forma de máquinas de estado concorrentes. As transições controláveis são responsáveis por gerar comandos para a SO, enquanto as transições não controláveis sinalizam respostas das SO. Conforme Queiroz e Cury (2002), alguns cuidados devem ser tomados para garantir a correta operação da estrutura de controle proposta: 1) os supervisores devem estar atualizados para a ocorrência de uma transição controlável no SP; 2) as transições não controláveis do SP devem ser tratadas com maior prioridade que as transições controláveis, devido a sua imprevisibilidade; e 3) a sinalização da ocorrência de eventos controláveis para o SM deve ser feita de forma sequencial.

Sequências Operacionais: este nível faz a comunicação entre o SP e o sistema real. Os comandos vindos do SP, isto é, os eventos controláveis que não foram desabilitados pelos supervisores modulares, são associados a uma sequência operacional. Conforme Queiroz e Cury (2002), é neste nível que o programa interpreta os comandos abstratos do SP, como sequências lógicas que guiam a operação de cada subsistema, em que são gerados os sinais de saída (correspondentes aos eventos controláveis) do sistema de controle e são lidos os sinais de entradas (correspondentes aos eventos não-controláveis) vindos do sistema real.

Essa estrutura foi estendida por Scotti (2015) que integrou: CLP; SCADA; Gerenciador de Modos de Operação (GMO); e o Sistema de Roteamento de Tarefas (SRT) (Figura 31). Nessa arquitetura, o GMO define o modo de operação a ser realizado pelo SP e caso seja executada alguma ação que leve a lógica a um estado de bloqueio, o GMO recebe a informação de lógica não segura do SP e entra em modo de segurança. O sistema SCADA, por sua vez, troca informações sobre os modos de operação com o

GMO; recebe informações sobre o estado atual do SP para realizar as representações gráficas; recebe dos supervisores as informações de quais eventos estão desabilitados; e permite ao operador habilitar os eventos quando em modo manual. Na sequência apresentam-se o GMO e o SCADA, utilizados na Seção 3.3, para a implementação da Estação de Separação.

Figura 31 – Arquitetura de Sistema do Controle Supervisório Integrado



Fonte: Scotti (2015).

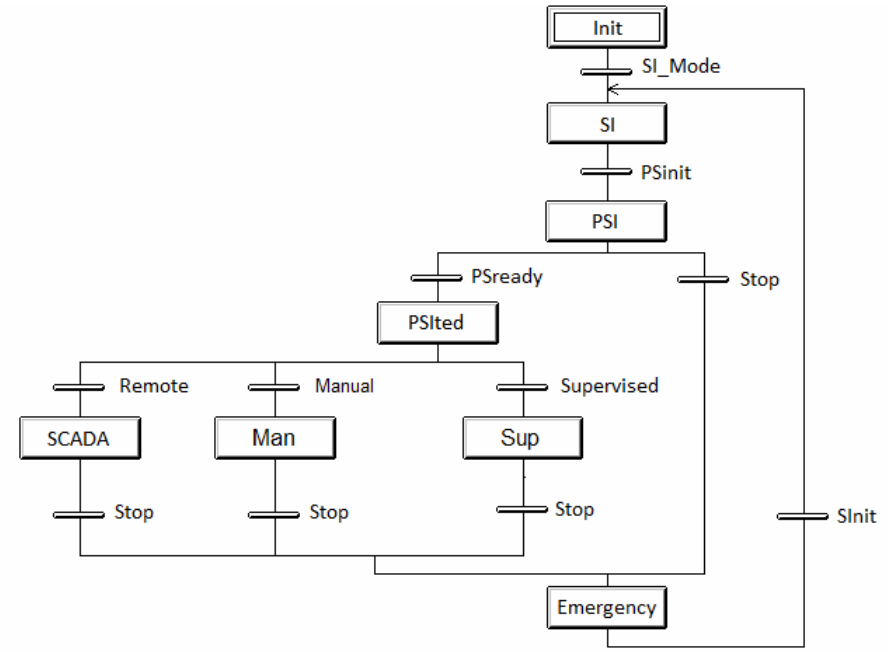
Gerenciador Modos de Operação (GMO):

Em Vieira (2007) é proposta uma metodologia para a implementação da inicialização do sistema. Essa metodologia estabelece um método formal para implementar a arquitetura de controle supervisório, visando superar algumas limitações, como: conduzir o sistema para o estado inicial por meio de uma sequência de tarefas pré-estabelecidas; reação em caso de emergência; interrupção da geração de eventos controláveis; ou geração seletiva de eventos controláveis. Os modos de operação são: *software inicialization* (SI); *physical system initialization* (PSI); *supervised* (Sup); *manual* (Man); *emergency* (Emergency). Em Portilla, Queiroz e Cury (2014) é proposta uma extensão desse método ao incluir o modo de operação *remote* (SCADA), como pode ser observado na Figura 32.

Integração com sistema SCADA:

Em Constain (2011) é proposta uma metodologia para o desenvolvimento integrado de sistemas SCADA com a programação de controle supervisório em CLP para sistemas de manufatura. Essa metodologia é constituída de oito fases: projeto

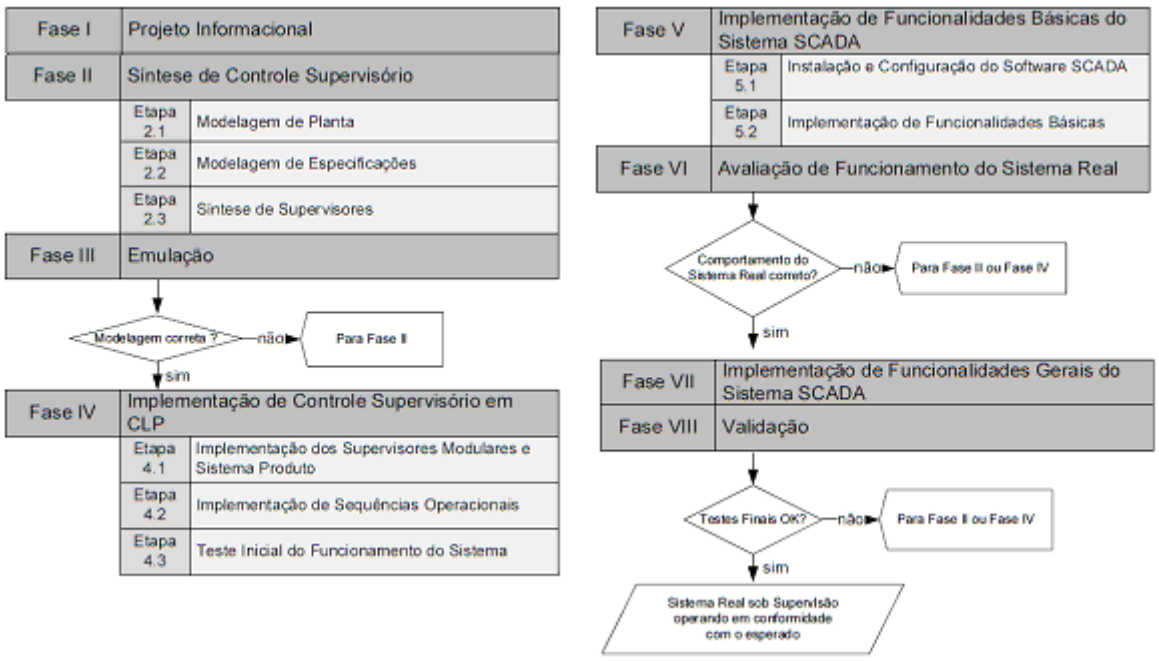
Figura 32 – Gerenciador modos de operação.



Fonte: Adaptado de Vieira (2007).

informacional; síntese de controle supervisório; emulação; implementação de controle supervisório em CLP; implementação de funcionalidades básicas do SCADA; avaliação de funcionamento do sistema real; implementação de funcionalidades gerais do sistema SCADA; e validação, como pode ser visto na Figura 33.

Figura 33 – Metodologia para integração de Controle Supervisório a um Sistema SCADA



Fonte: (Portilha, Queiroz e Cury (2014)

A seguir apresenta-se de forma resumida cada uma dessas fases:

- **Fase I: Projeto Informacional.** Nesta fase é realizado um levantamento das informações técnicas referente à planta, para assim começar a modelar o sistema.
- **Fase II: Síntese de Controle Supervisório.** Esta fase é dividida em três partes: modelagem dos subsistemas; modelagem das especificações; e síntese de supervisores.
  - *Modelagem dos subsistemas:* esta etapa identifica o conjunto de subsistemas envolvidos no problema e divide a planta em subsistemas concorrentes que interagem entre si para a execução das tarefas.
  - *Modelagem das especificações:* nesta etapa é modelado o comportamento desejado para a planta definido na Fase I. Para isso, são modeladas cada restrição de coordenação do sistema a ser controlado, de forma a resolver os problemas de interação entre os subsistemas que compõem a planta.
  - *Síntese de Supervisores:* nesta etapa são sintetizados os supervisores locais para cada subsistema afetado por uma restrição, dessa forma, cada supervisor local coordenará uma parte do sistema (planta). A síntese desses supervisores segue a abordagem modular local proposta por Queiroz e Cury (2000), onde cada um dos supervisores locais representa a máxima linguagem controlável para cada planta local.
- **Fase III: Emulação.** Nesta fase, são realizados testes da lógica de controle proposta. Por meio da simulação é possível visualizar a atuação dos supervisores sobre a planta, possibilitando identificar erros ocorridos na modelagem. Caso ocorram erros de modelagem, deve-se retornar à Fase II para realizar as mudanças de projeto necessárias. Caso contrário, os supervisores locais e os subsistemas podem ser implementados.
- **Fase IV: Implementação do Controle Supervisório em CLP.** Nesta fase, cada nível proposto por Queiroz e Cury (2002) é implementado no CLP.
- **Fase V: Implementação das funcionalidades básicas do Sistema *SCADA*.** Nesta fase são implementadas as funcionalidades, tais como: o sinótico; o envio de comandos; o histórico de eventos; a geração de alarmes críticos, entre outros.
- **Fase VI: Avaliação de funcionamento do sistema real.** Esta fase tem o propósito de avaliar se o comportamento do sistema real está de acordo com o inicialmente planejado e ainda são realizados testes integrando os dispositivos do sistema real ao CLP.

- **Fase VII: Implementação das funcionalidades gerais do Sistema *SCADA*.** Esta fase tem a finalidade de implementar funções como a geração de alarmes, relatórios e geração de informação para níveis gerenciais, entre outros.
- **Fase VIII: Validação.** Esta fase tem a finalidade de realizar os testes finais que permitam validar e certificar o funcionamento ótimo do sistema real e da aplicação *SCADA* para o sistema de manufatura.

Para a implementação no estudo de caso que integra a Seção 3.3, serão utilizadas a arquitetura de controle modular composta por três níveis, a integração com o sistema SCADA e o GMO, analisados nesta seção.

3.3 IMPLEMENTAÇÃO DO CML NA ESTAÇÃO DE SEPARAÇÃO

A metodologia utilizada segue as fases de implementação proposta por Constain (2011), e depois apresentadas no trabalho de Scotti (2015), conforme visto anteriormente. A partir da obtenção da fase II (os modelos dos subsistemas, das restrições e dos supervisores modulares locais reduzidos), é possível implementar as fases IV, V, VI, VII e VIII, que serão descritas nesta subseção. O código utilizado para essa implementação é gerado automaticamente, por meio da ferramenta computacional *IDES2ST*[1], tendo como padrão, a linguagem de Texto Estruturado (ST), de acordo a norma IEC-61131-3.

Para um melhor entendimento do código gerado, apresentam-se, na sequência, as principais variáveis criadas pelo IDES2ST, conforme o padrão de Vieira (2007).

- *e_* < *evento* >: Variável padrão em que os eventos são criados. Exemplo: *e_IT*;
- *Ae_* < *evento* >: Variável que interpreta os sinais de resposta da planta. Ativa os eventos não-controláveis da lógica. Exemplo: *Ae_Saida_Ocup* (ativa o evento *e_Saida_Ocup*);
- *De_* < *evento_c* >: Variável que desabilita os eventos controláveis. Exemplo: *De_IT* (desabilita o início do teste);
- *p_* < *nome_do_subsistema_* > *St*: Variável que contém o valor do estado atual do sistema produto correspondente. Exemplo: *p_G*1*_St* (indica o estado de *G*1);
- *s_* < *nome_do_supervisor_reduzido_* > *St*: Variável que contém o valor do estado atual do supervisor correspondente. Exemplo: *s_Sred*1*_St*; (indica o estado do supervisor reduzido 1).

[1] https://sourceforge.net/projects/ides2st/

No nível SM, os supervisores $S_{red,j}$ $j \in \{1,2,3,4,6,7\}$ são implementados como máquina de estados concorrentes, e as desabilitações dos eventos controláveis *De_* < *evento_c* > são sinalizadas de acordo com os estados ativos. O código gerado automaticamente pelo *IDES2ST* possui uma *Function Block* (FB Supervisores) que divide a implementação em duas partes: a primeira implementa as máquinas de estados; e a segunda, as desabilitações.

A Figura 34 ilustra a implementação dessas duas etapas, em ST, para o supervisor $S_{red,5}$. A ocorrência dos eventos não-controláveis *FTOK*, *FTNOK* e *RB* é detectada através das variáveis de entrada *e_FTOK*, *e_FTNOK* e *e_RB*. A geração dos eventos controláveis resulta na ativação, durante um ciclo de atualização, das variáveis de saída (ex: evento *ID*12 e variável *e_ID*12). Associa-se, também, a cada estado do supervisor ($q0$, $q1$, $q2$), uma variável booleana (*s_Sred*5_*St* = 0), (*s_Sred*5_*St* = 1) e (*s_Sred*5_*St* = 2). Assim, pode-se observar que o supervisor transitará do estado $q0$ para o estado $q1$, quando a variável *s_Sred*5_*St*=0 e o evento não-controlável *FTNOK* ocorrer, resultando na ativação da variável *s_Sred*5_*St* := 1). No entanto, se o supervisor estiver no estado $q1$ e ocorrer o evento não-controlável *e_FTOK*, o mesmo passará para o estado $q2$ (*s_Sred*5_*St* := 2). Para tratar o problema do efeito avalanche é incluída a variável *evt_blk_SU*. Já a segunda parte do código gerado, que desabilita os eventos controláveis, pode-se observar nesta Figura, por exemplo, que o evento *ID*13 é desabilitado (*De_ID*13 = 1) quando o *s_Sred*5_*St* estiver no estado $q0$ ou $q1$, e o evento *ID*12 quando estiver no estado 0 ou $q2$.

Já os subsistemas G_i $i \in \{1,2,3,4,5\}$ são implementados no SP como máquinas de estados assíncronas e as transições associadas aos eventos não-controláveis são disparadas por sinais vindos da SO, como por exemplo: os eventos final do teste (*FTOK* e *FTNOK* e as respectivas variáveis *Ae_FTOK* e *Ae_FNTOK*), e os eventos de final de deslocamentos do sistema (*FD*12, *FD*123 ... e as respectivas variáveis *Ae_FD*12, *Ae_FD*123, ...). Os eventos controláveis são gerados sempre que não forem desabilitados pelos supervisores, como por exemplo: a variável início do teste (*e_IT*) é gerada sempre que o subsistema $G_{loc,2}$ estiver no estado *qo* e não estiver sendo desabilitada por nenhum supervisor (*De_IT*), como pode ser visto na Figura 35. Cada transição deve ativar um sinal de ocorrência do respectivo evento (*e_* < *event* >) que será sinalizada para o nível SM.

Da mesma forma, neste nível, o código do SP gerado pela ferramenta IDES2ST possui uma *Function Block* (FB Sistema Produto). Neste caso, o código gerado implementa primeiro a lógica dos eventos não-controláveis, devido a sua imprevisibilidade e, depois, a lógica dos eventos controláveis. Antes que qualquer evento controlável possa ocorrer, o programa verifica os eventos não-controláveis e atualiza o sistema caso ocorra alguma mudança.

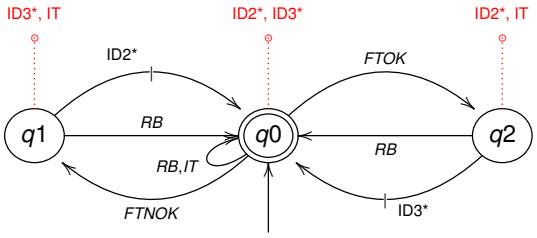
Figura 34 – Código de Implementação Supervisor $S_{red,5}$

Máquina de Estados

```
(*Uncontrollable events of s_Sred5*)
"evt_blkSU" := 0;
IF ((NOT "evt_blkSU") AND "s_Sred5_St"=0 AND "e_FTNOK") THEN
    "evt_blkSU" := 1;  "s_Sred5_St" := 1;
END_IF
IF ((NOT "evt_blkSU") AND "s_Sred5_St"=0 AND "e_FTOK") THEN
    "evt_blkSU" := 1;  "s_Sred5_St" := 2;
END_IF
IF ((NOT "evt_blkSU") AND "s_Sred5_St"=1 AND "e_RB") THEN            "
    evt_blkSU" := 1;  "s_Sred5_St" := 0;
END_IF
IF ((NOT "evt_blkSU") AND "s_Sred5_St"=2 AND "e_RB") THEN
    "evt_blkSU" := 1;  "s_Sred5_St" := 0;
END_IF
(*Controllable events of s_Sred5*)
IF ((NOT "evt_blkSU") AND "s_Sred5_St"=1 AND "e_ID12" AND "e_ID124"
    AND "e_ID2" AND "e_ID24") THEN
    "evt_blkSU" := 1;  "s_s_Sred5_St" := 0;
END_IF
IF ((NOT "evt_blkSU") AND "s_Sred5_St"=2 AND "e_ID13" AND "e_ID134"
    AND "e_ID3" AND "e_ID34") THEN
    "evt_blkSU" := 1;  "s_s_Sred5_St" := 0;
END_IF
```

Desabilitações

```
(*e_ID13*) De_ID13:= 0;
IF (s_Sred5_St=0 OR s_Sred5_St=1) THEN
    De_ID13:= 1;
END_IF
(*e_ID3*) De_ID3:= 0;
IF (s_Sred5_St=0 OR s_Sred5_St=1) THEN
    De_ID3:= 1;
END_IF
(*e_ID34*) De_ID34:= 0;
IF (s_Sred5_St=0 OR s_Sred5_St=1) THEN
    De_ID34:= 1;
END_IF
  ⋮
(*e_ID24*) De_ID24:= 0;
IF (s_Sred5_St=0 OR s_Sred5_St=2) THEN
    De_ID24:= 1;
END_IF
(*e_ID124*) De_ID124:= 0;
IF (s_Sred5_St=0 OR s_Sred5_St=2) THEN
    De_ID124:= 1;
END_IF
(*e_IT*) De_IT:= 0;
IF (s_Sred5_St=1 OR s_Sred5_St=2) THEN
    De_IT:=1;
END_IF
```



Fonte: Elaborado pelo autor (2021).

A Figura 35 ilustra o código gerado para o subsistema G_2.

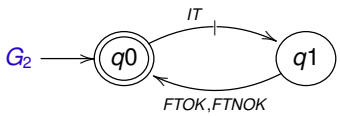
Figura 35 – Código de Implementação Subsistema G_2

Máquina de Estados - (Σu)

```
(*Uncontrollable events of p_SP2*)
IF ((NOT "evt_blkSU") AND "p_SP2_St"=1 AND "Ae_FTOK") THEN
    "Ae_FTOK" := 0;
    "e_FTOK" := 1;
    "evt_blkSU" := 1;
    "s_SP2_St" := 0;
END_IF
IF ((NOT "evt_blkSU") AND "p_SP2_St"=1 AND "Ae_FTNOK") THEN
    "Ae_FTNOK" := 0;
    "e_FTNOK" := 1;
    "evt_blkSU" := 1;
    "s_SP2_St" := 0;
END_IF
```

Máquina de Estados - (Σc)

```
(*Controllable events of p_SP2*)
IF ((NOT "evt_blkSU") AND "p_SP2_St"=0 AND ("NOT De_IT")) THEN
    "e_IT" := 1;
    "evt_blkSU" := 1;
    "s_SP2_St" := 1;
END_IF
```



Fonte: Elaborado pelo autor (2021).

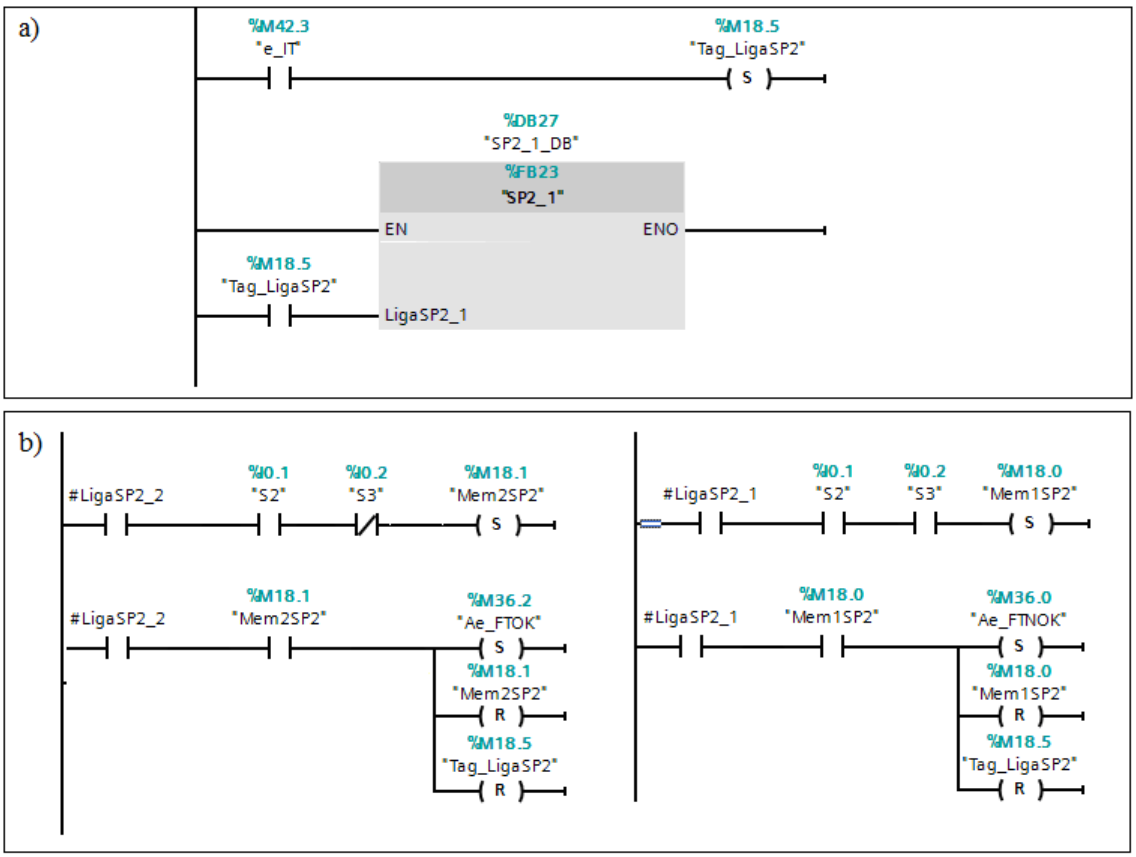
A Figura 36a ilustra a sequência operacional do subsistema G_2. O Sistema Produto atualiza as informações sobre os estados desse subsistema e o supervisor desabilita os eventos que não podem ocorrer. Caso a variável *De_IT* esteja desabilitada e o início do teste possa ocorrer, a variável *e_IT* é ativada e a sub-rotina SP2 será executada. Quando o programa entrar na sub-rotina SP2, será realizado o teste na peça. Portanto, se os sensores S2 e S3 estiverem ativos, a peça testada deverá ser rejeitada e a variável *Ae_NTOK* será habilitada, caso contrário, a peça testada será "boa" e a variável *Ae_TOK* será habilitada, conforme Figura 36b.

Seguindo a metodologia proposta por Vieira (2007), foram implementados os modos de operação: software *inicialization* (*SI*); *physical system initialization* (*PSI*); *supervised* (*Sup*); *Remote* (*SCADA*); e *emergency* (*Emergency*). A Figura 37 ilustra o código do modo de inicialização implementado no CLP.

Na Figura 37a é apresenta a implementação do modo *SI*. Nesse modo, a variável *Init* será ativada sempre na primeira execução do programa, o que faz setar automaticamente a variável *SI_Mode*, ativando a FB de inicialização, que é responsável por zerar o valor de todas as variáveis internas do CLP.

A Figura 37b mostra a implementação do modo *PSI*, que representa a inicia-

Figura 36 – Sequência Operacional do subsistema G_2

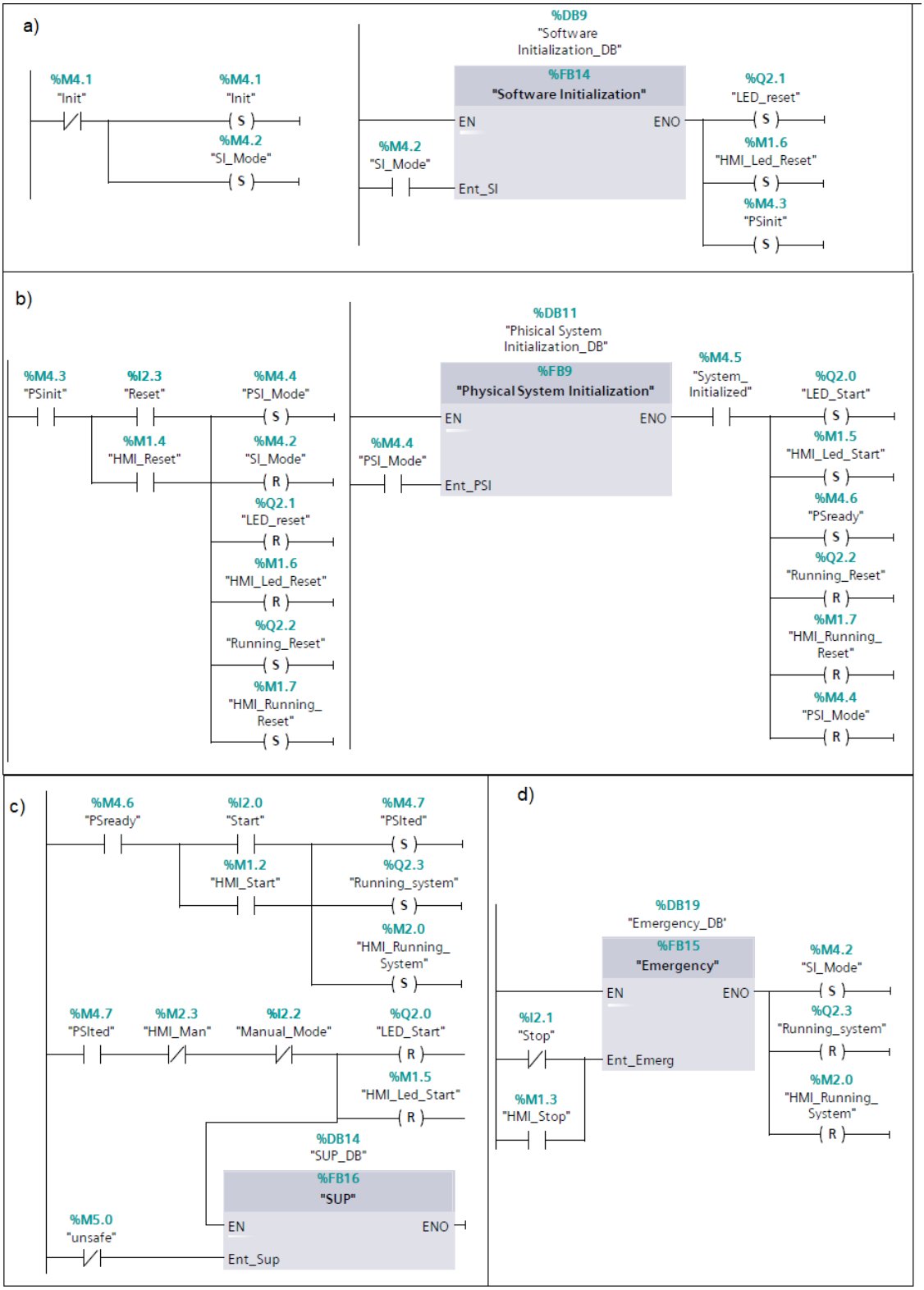


Fonte: Elaborado pelo autor (2021).

lização física do sistema, executada logo após o modo *SI*. Nesse modo, o sistema ficará aguardando o acionamento do botão *Reset*, para ser iniciado, em que todas as peças que estiverem na estação serão descartadas, de forma a permitir que o sistema possa reiniciar sem peças e retornar ao seu estado inicial. Concluída a inicialização do sistema, a variável interna *PSready* é ativada e o sistema pode entrar no modo Supervisionado (SUP), como pode ser visto na Figura 37c. A Figura 37d apresenta a implementação do modo de emergência (Emergency), em que é ativado quando pressionado o botão *Stop*, onde todas as atividades são suspensas imediatamente e, na sequência, o sistema retorna para o modo SI, realizando novamente todo o processo de inicialização do sistema.

Para a construção do sinótico, foram analisados os estados necessários e suficientes para visualizar corretamente o funcionamento da planta. Na sequência, foram criadas as variáveis como tags no software *SCADA*, de forma a serem inseridas na tela do sinótico. Portanto, podem-se monitorar os estados dos supervisores e das principais variáveis do sistema, como por exemplo: o resultado do teste; buffer do descarte cheio/livre; e saída ocupada/livre. Além disso, é possível comandar remotamente o sistema, por meio dos comandos iniciar, parar e reiniciar.

Figura 37 – Código CLP - implementação do sistema de inicialização

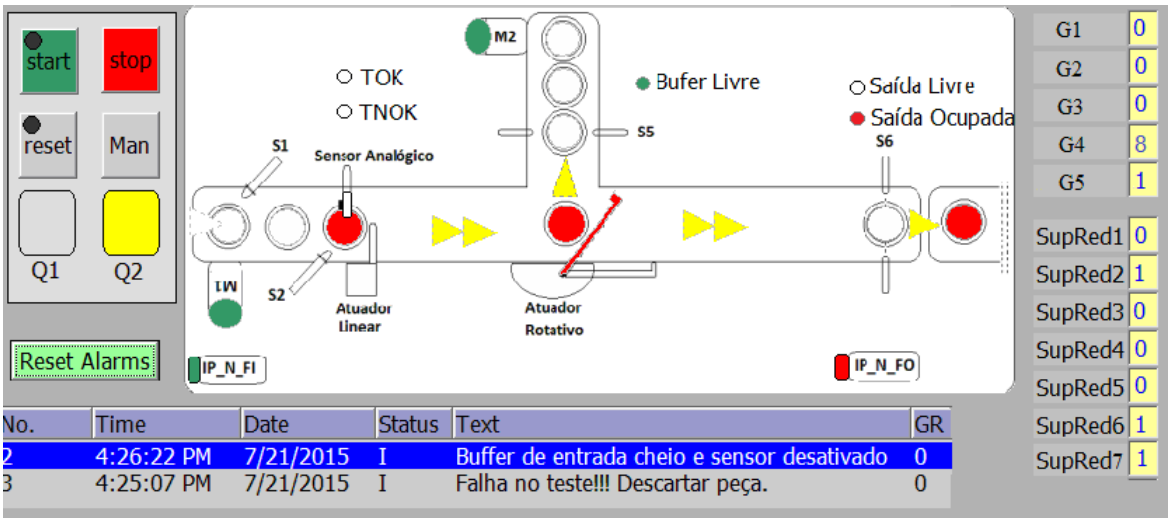


Fonte: Elaborado pelo autor (2021).

Foram aplicadas, no sinótico, as informações dos estados do subsistema *G*4, o qual é responsável pelo início e fim de todos os deslocamentos possíveis nas esteiras e pela detecção de falhas no fluxo de material. O subsistema *G*4 dividiu os deslocamen-

tos em 11 movimentos possíveis, permitindo visualizar, no sinótico, os deslocamentos ativos, a ocorrência de falhas no sistema e as posições das peças, de forma indireta, sem o uso de sensores. A Figura 38 mostra, como exemplo, o estado da variável *p_G4_St* que se encontra no estado 8, onde ocorre o evento controlável *e_ID24*.

Figura 38 – Sinótico Estação MPS Festo de Separação com alarmes



Fonte: Elaborado pelo autor (2021).

Geração de Alarmes:

Os alarmes fornecem a indicação global do estado do sistema e chamam a atenção do operador para alguma modificação do estado do processo. Esses alarmes combinam a informação dos sensores com o estado lógico dos modelos dos geradores. Logo, foram implementados alarmes no sistema para informar ao usuário, por meio de uma tabela, a descrição dos eventos proibidos que ocorreram, de modo que, as falhas identificadas possam ser corrigidas e as decisões operacionais apropriadas possam ser adotadas. Na Figura 38 visualiza-se o disparo de dois alarmes: o primeiro, indica que o buffer de entrada está cheio, porém, o sensor S1 está desativado; e, o segundo, informa a ocorrência de um erro no teste (descartar peça).

3.4 CONCLUSÃO DO CAPÍTULO

Com os resultados observados na implementação dessa arquitetura de controle supervisório na bancada didática do LAI, verificou-se que o comportamento do sistema funcionou de acordo com o esperado. Assim, pode-se dizer que a implementação da lógica de controle obtida para a estação de separação foi bem-sucedida. Tem-se com uma contribuição parcial, a implementação em um sistema real, uma vez que existem poucas implementações da lógica de CML por ser uma teoria nova. Os resultados obtidos demonstraram que a metodologia aplicada mostrou-se eficiente na sistematização e elaboração da solução do sistema, tornando a solução do projeto mais completa,

segura, flexível e minimamente restritiva, além de permitir estruturação do programa do CLP e uma maior sistematização para a integração do sistema SCADA. A implementação da lógica de CML torna possível a implementação dos supervisores no CLP, uma vez que foram implementados 8 supervisores modulares de 2 a 4 estados, resultando em uma lógica implementável na memória do CLP. No próximo capítulo serão apresentados os principais conceitos da TCS para SEDT e os modelos temporizados da planta e das especificações para a Estação MPS Festo de Separação .

4 CONTROLE SUPERVISÓRIO DE SISTEMAS A EVENTOS DISCRETOS TEMPORIZADOS

Neste Capítulo tratam-se, de forma resumida, os principais conceitos da Teoria de Controle Supervisório de sistemas a eventos discretos temporizados (SEDT), iniciada por Brandin e Wonham (1994), tendo como foco principal a abordagem modular local temporizada proposta por Schafaschek, Queiroz e Cury (2015).

Na sequência, apresenta-se a solução do problema real de controle aplicado à Estação de Separação do Sistema Modular de Produção (MPS), localizado no Laboratório de Automação Industrial (LAI) da UFSC. A resolução deste problema servirá para exemplificar os conceitos deste capítulo e também será usada no Capítulo 5 para aplicação da arquitetura de implementação temporizada.

4.1 MODELAGEM

Nesta seção apresenta-se o modelo de Brandin e Wonham para SEDs temporizados, iniciando-se com uma breve apresentação de autômatos temporizados e operações sobre linguagens e autômatos temporizados. A título de exemplo de aplicação, são apresentados os modelos temporizados da planta e das especificações para a Estação de Separação.

4.1.1 Autômatos temporizados

Conforme Brandin e Wonham (1994), para definir um sistema a eventos discretos temporizado $\mathbf{G}$, parte-se de um autômato não temporizado $\mathbf{G}_{\text{act}} = (A,\Sigma_{\text{act}},\delta_{\text{act}},a_0,A_m)$, onde A é o conjunto de atividades executadas pelo sistema; Σ_{act} corresponde a um conjunto finito de eventos do sistema; a função de transição atividade é uma função parcial $\delta_{\text{act}} : A \times \Sigma_{\text{act}} \rightarrow A$, que descreve como o sistema realiza a transição entre as atividades, após a ocorrência dos eventos em Σ_{act}; a_0 e A_m são, respectivamente, a atividade inicial e as atividades marcadas do sistema.

O tempo é formalmente introduzido em $\mathbf{G}_{\text{act}}$ por meio de um novo evento *tick* que representa uma unidade de tempo, medida por um relógio digital global, o qual assume, portanto, valores temporais discretos, determinados pelo número de ocorrências de *tick*. O conjunto de eventos que descreve um SEDT é, consequentemente, dado por $\Sigma = \Sigma_{\text{act}} \cup \{tick\}$, onde cada evento $\sigma \in \Sigma_{\text{act}}$ tem um *Limite Inferior* $l_\sigma \in \mathbb{N}$ e um *Limite Superior* $u_\sigma \in \mathbb{N} \cup \{\infty\}$, que representam, respectivamente, o número mínimo e máximo de *ticks* que podem ser observados antes da ocorrência de σ, sendo $l_\sigma \leq u_\sigma$. Σ_{act} é dividido em dois subconjuntos: o de eventos remotos que podem ser atrasados indefinidamente, $\Sigma_{\text{rem}} = \{\sigma \in \Sigma_{\text{act}} | u_\sigma = \infty\}$; e o de eventos esperados, que uma vez ha-

bilitados, devem ocorrer em uma quantidade finita de tempo, $\Sigma_{esp} = \{\sigma \in \Sigma_{act} | u_\sigma < \infty\}$. Tem-se portanto $\Sigma_{act} = \Sigma_{rem} \dot{\cup} \Sigma_{esp}$.

A forma pela qual os limites temporais restringem a ocorrência de evento é definida pelo número de ocorrências do *tick*. Assim é usado um *cronômetro* τ_σ para cada $\sigma \in \Sigma_{act}$, que assume valores em um intervalo temporizado $T_\sigma \subseteq N$, definido como $T_\sigma = [0, l_\sigma]$ se $\sigma \in \Sigma_{rem}$ e $T_\sigma = [0, u_\sigma]$ se $\sigma \in \Sigma_{esp}$. O valor inicial do *cronômetro* associado a cada elemento $\sigma \in \Sigma_{act}$ é dado por: $\tau_{\sigma 0} = l_\sigma$ *se* $\sigma \in \Sigma_{rem}$; e $\tau_{\sigma 0} = u_\sigma$ *se* $\sigma \in \Sigma_{esp}$.

O cronômetro inicia uma contagem regressiva a partir do seu valor inicial $\tau_{\sigma 0}$, quando um evento estiver habilitado em $\mathbf{G}_{act}$, onde a cada ocorrência do *tick* será decrementada em uma unidade, enquanto o evento continuar habilitado. O cronômetro retorna ao seu valor inicial, na desabilitação ou na ocorrência de um evento relacionando a este cronômetro. Para $\sigma \in \Sigma_{rem}$, o evento passa a estar elegível quando $\tau_\sigma = 0$ e permanece elegível indefinidamente, pois quando o cronômetro atingir o valor nulo, a sua contagem é interrompida e o seu valor não mudará, enquanto o evento estiver habilitado e não ocorrer. Já o $\sigma \in \Sigma_{esp}$ torna-se elegível quando $\tau_\sigma = u_\sigma - l_\sigma$ e torna-se *iminente* caso permaneça habilitado pelo tempo máximo de unidade de tempo (u_σ), ou seja, quando $\tau_\sigma = 0$ (o cronômetro atingiu valor nulo). Quando estiver iminente, σ deverá acontecer antes do próximo *tick* do relógio, a menos que seja desativado nesse intervalo de tempo por outro evento de Σ_{act}.

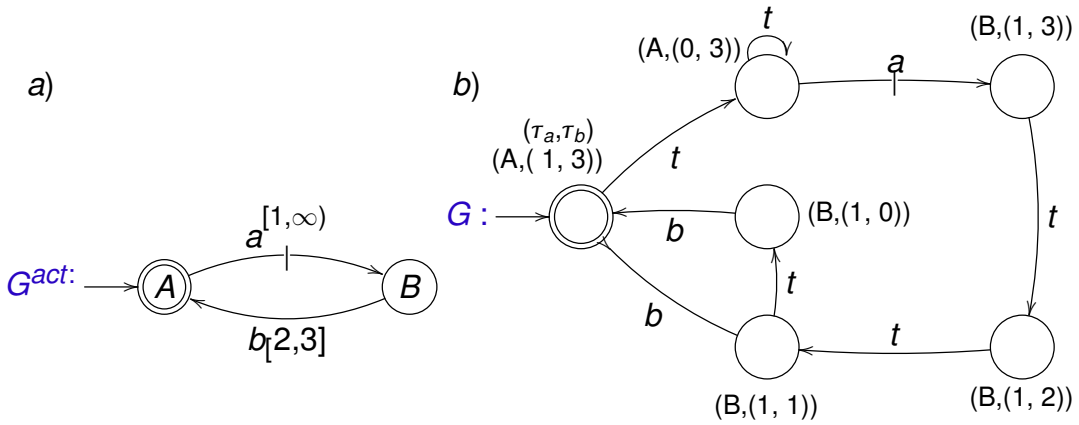
Então, um SEDT pode ser representado por um autômato $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, que incorpora o evento *tick* em uma função de transição $\delta : Q \times \Sigma \to Q$. A linguagem gerada e marcada de $\mathbf{G}$ são definidas, respectivamente, por $L(\mathbf{G}) = \{s \in \Sigma^* | \delta(q_0, s)!\}$ e $L_m(\mathbf{G}) = \{s \in \Sigma^* | \delta(q_0, s) \in Q_m\}$. Para $s \in L(\mathbf{G})$, $\Sigma_G(s) = \{\sigma \in \Sigma | \delta(q_0, s\sigma)!\}$. Um SEDT é não bloqueante se $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$.

Um SEDT pode ser expresso graficamente em dois tipos de estruturas: por um *Grafo de Transição de Atividades* (GTA) que corresponde a um $\mathbf{G}_{act}$, com as transições marcadas por eventos e seus intervalos de tempo correspondentes, ou *Grafo de Transição Temporizado* (TTG) que corresponde ao grafo de transição de $\mathbf{G}$, que introduz a noção do tempo, explicitamente na estrutura de transição do autômato, por meio da adição do evento *tick*, que modela o avanço de uma unidade do tempo discreto.

Na Figura 39a é apresentado um exemplo de um autômato GTA, onde o intervalo associado ao evento *a* remoto é $[1, \infty)$, significando que, após uma unidade de tempo (uma ocorrência do evento *tick*) a partir da atividade inicial (*A*), o evento *a* fica habilitado indefinidamente. Para o evento esperado *b*, o intervalo de tempo associado é $[2, 3]$, que representa a necessidade de, no mínimo, duas ocorrências do evento *tick* na atividade (*B*) para que esteja habilitado e, no máximo, três ocorrências

do evento *tick* para que ocorra de forma mandatória, antes da ocorrência de outro evento *tick* do relógio digital. A Figura 39b mostra o GTT do autômato descrito, onde o evento *tick* é representado pela letra *t*. Supondo que o sistema esteja na atividade inicial *A*, onde o valor do cronômetro do evento *a* (τ_a) é igual a 1 e do evento *b* é igual a 3 ((*A*,(1,3))), após a ocorrência de um *tick* o valor de τ_a é decrementado e o evento *a* torna-se elegível ($a \in \Sigma_{rem}$ e $\tau_a = 0$). O cronômetro do relógio permanece com o valor nulo (Atividade (*A*,(0,3))) até que o evento *a* ocorra, e é reiniciado após a ocorrência do mesmo (Atividade (*B*,(1,3))). O evento *b* está elegível na atividade (*B*,(1,1)), quando $\tau_b = u_\sigma - l_\sigma = 1$, e iminente na atividade (*B*,(1,0)), quando $\tau_b = 0$. Cabe ressaltar que para esse exemplo, o critério de modelagem para marcação do GTT não é o cronômetro em zero e, por isso, o estado equivalente a atividade (*A*,(0,3)) do GTT não é marcado. Deve-se ressaltar que a escolha do valor de duração de um *tick* é uma decisão importante da etapa de modelagem de um SEDT pois, como veremos posteriormente, esta escolha afeta o nível de exatidão e a complexidade dos modelos envolvidos, e consequentemente, a complexidade da síntese de supervisores e desempenho do sistema controlado resultante.

Figura 39 – a) GTA b) GTT



Fonte: Elaborado pelo autor (2021).

Operações sobre linguagens e autômatos:

Brandin e Wonham (1994) definem a composição de um SEDT como $\mathbf{G}_{12} = comp(\mathbf{G}_1,\mathbf{G}_2)$. A função *comp* representa o comportamento síncrono dos autômatos $\mathbf{G}_{act1}$ e $\mathbf{G}_{act2}$ com a composição dos intervalos de tempo $[u_\sigma,l_\sigma]$ determinadas pelas seguintes regras: se $\sigma \in (\Sigma_{act1} - \Sigma_{act2}) \cup (\Sigma_{act2} - \Sigma_{act1})$, o intervalo $[u_\sigma,l_\sigma]$ permanece inalterado; se $\sigma \in (\Sigma_{act1} \cap \Sigma_{act2})$, o intervalo $[u_{12_\sigma},l_{12_\sigma}]$ é definido como $[max(l_{1_\sigma},l_{2_\sigma}), min(u_{1_\sigma},u_{2_\sigma})]$ desde que $l_{12_\sigma} \leq u_{12_\sigma}$. Cabe ressaltar que o resultado da composição de dois autômatos GTAs, realizada pela operação *comp*, pode ser diferente do resultado obtido na composição síncrona (||) dos seus respectivos autômatos GTTs. O produto síncrono de plantas com eventos em comum e que não seja

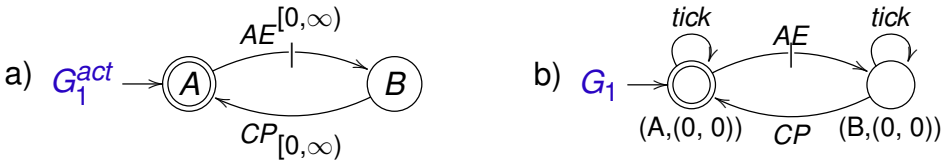
o *tick*, isto é, ($\Sigma_1 \cap \Sigma_2 \neq \{tick\}$) pode acarretar em um bloqueio e um travamento do relógio. Dados dois autômatos assíncronos $\mathbf{G}_1$ e $\mathbf{G}_2$ que compartilham um relógio global, se $\Sigma_1 \cap \Sigma_2 = \{tick\}$, então conforme Wonham e Cai (2019) pode-se afirmar que $comp(\mathbf{G}_1,\mathbf{G}_2) \approx \mathbf{G}_1||\mathbf{G}_2$, onde $\approx$ denota que as linguagens geradas e marcadas coincidem.

4.1.2 Modelagem dos autômatos temporizados da Estação de Separação

Na sequência é apresentada a modelagem dos cinco subsistemas que compõem a Estação de Separação, segundo a abordagem do controle supervisório modular local temporizado. Na Seção 2.2 já foram apresentadas as descrições de cada subsistema e, para o caso aqui proposto, assumem-se as mesmas descrições já mencionadas anteriormente. A seguir serão apresentados os grafos de atividades (GTA) de cada subsistema e seus respectivos grafos de transições (GTT). Para a modelagem da planta, o valor do *tick* escolhido foi de 1 segundo.

G1 – Entrada: no grafo de atividades (GTA) da Figura 40, o intervalo [0,∞), associado aos eventos *AE* e *CP*, indica que a ocorrência do evento *AE* a partir da atividade inicial, está habilitada em qualquer instante de tempo do relógio digital global e o evento *CP* pode ocorrer imediatamente após o evento *AE*, pois o mesmo está também habilitado em qualquer instante de tempo.

Figura 40 – G_1: Entrada a) GTA b) GTT



Atividades	
A	entrada proibida
B	entrada livre

Eventos		Descrição
AE	controlável e forçável	autoriza entrada de peças novas
CP	não-controlável	indica a chegada de uma nova peças

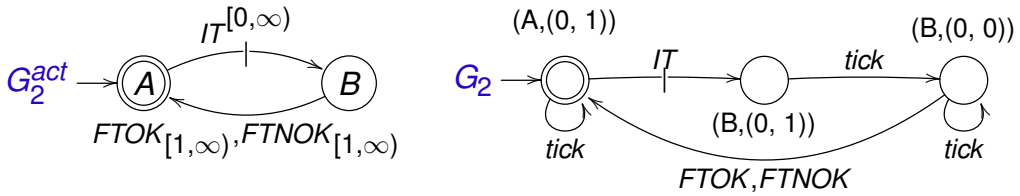
Fonte: Elaborado pelo autor (2021).

G2 - Teste: conforme o GTA da Figura 41, o intervalo de tempo associado ao evento *IT* é [0,∞), o que significa que este evento está habilitado em qualquer instante, a partir da atividade inicial. Os eventos *FTOK* e *FTNOK* têm o intervalo de tempo [1,∞) associado e estarão elegíveis em uma unidade de tempo do relógio digital (ocorrência de um *tick*), a partir do início de teste ou em qualquer instante depois.

G3 – Descarte: conforme o GTA da Figura 42, os eventos deste subsistema têm o intervalo [1,∞) associado. O evento *DESC_CHEIO* estará elegível em uma unidade de tempo, a partir da atividade inicial ou em qualquer instante depois. E, o evento

Figura 41 – $\mathbf{G}_2$: Teste a) GTA b) GTT

a) b)



Atividades	
A	não testando peça
B	testando peça

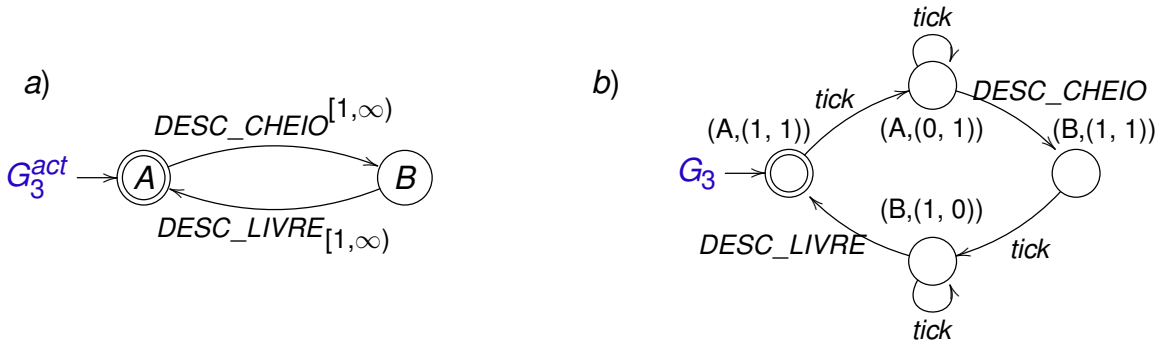
Eventos		Descrição
IT	controlável e forçável	início de teste
FTOK	não-controlável	fim de teste ok (peça boa)
FTNOK	não-controlável	fim de teste não ok (peça rejeitada)

Fonte: Elaborado pelo autor (2021).

DESC_LIVRE torna-se elegível uma unidade de tempo depois do *DESC_CHEIO* ocorrer ou em qualquer instante após.

Figura 42 – $\mathbf{G}_3$: Descarte de peças rejeitadas a) GTA b) GTT

a) *b)*



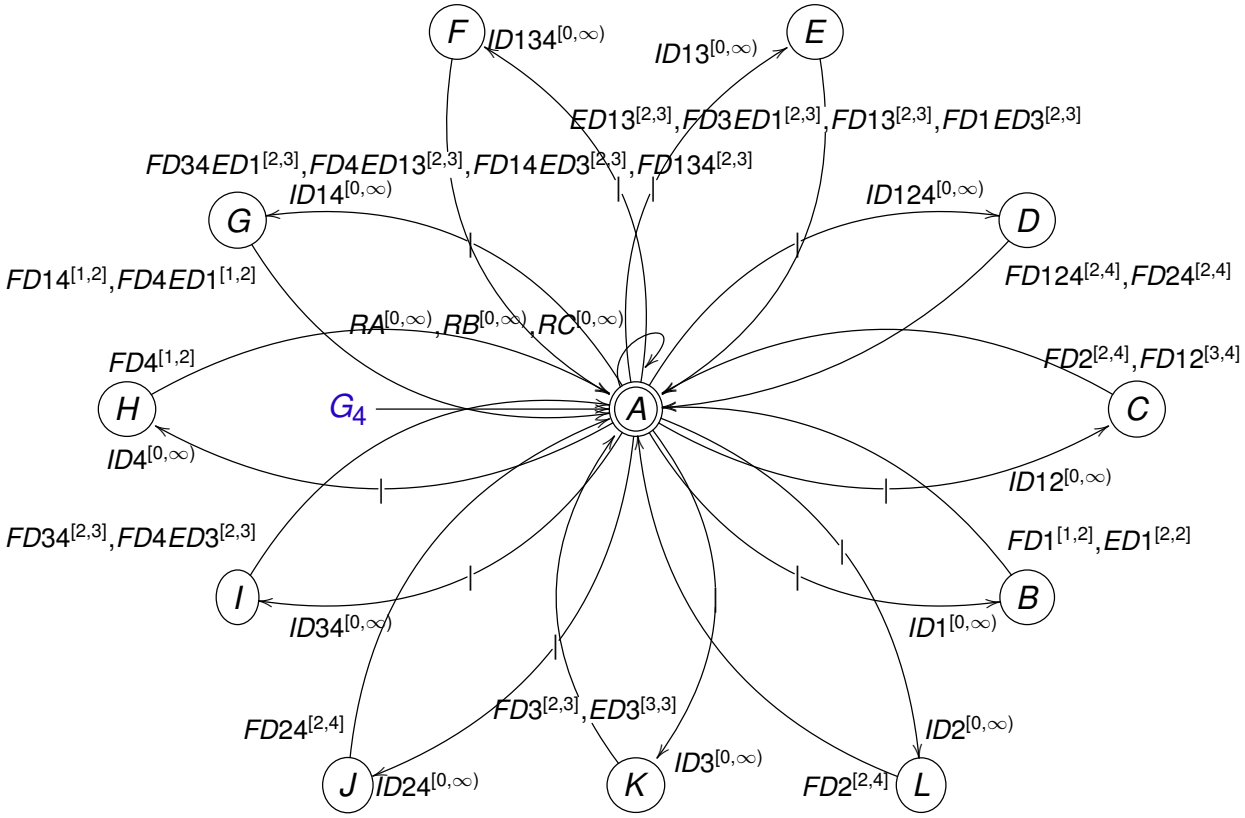
Atividades	
A	buffer descarte vazio
B	buffer descarte cheio

Eventos		Descrição
DESC_LIVRE	não-controlável	buffer esteira descarte está vazio
DESC_CHEIO	não-controlável	buffer esteira descarte está cheio

Fonte: Elaborado pelo autor (2021).

G4 – Deslocamentos: conforme o GTA da Figura 43, os eventos *IDi*, *RA*, *RB*, *RC* estão associados ao intervalo [0,∞), logo a ocorrência desses eventos, a partir da atividade inicial, está habilitada em qualquer instante de tempo do relógio digital global. Os eventos *FD*1 e *FD*4 estão associados ao intervalo de tempo [1, 2], que representa a necessidade de, no mínimo, uma ocorrência do evento *tick* após a ocorrência do seu respectivo início de deslocamento, para que o evento esteja elegível e, no máximo, duas ocorrências do evento *tick* para que o evento seja iminente, isto é, ocorra de forma mandatória antes de qualquer outro evento *tick*. O evento *ED*1 está associado ao intervalo de tempo [2, 2]. Após a ocorrência de dois *ticks*, o mesmo estará iminente (alcançou seu deadline) e sua ocorrência é obrigatória antes de qualquer outro evento do relógio. Na Figura 43, pode-se verificar os limites de tempo inferior l_σ e superior u_σ para cada evento pertencente a *G*4.

Figura 43 – $\mathbf{G_4}$: Deslocamentos a) GTA b)GTT



Atividades	
A	esteiras desligadas
B, E, F, G, H, I, K	esteira 1 ligada
L	esteira 2 ligada
C,D, J	esteiras 1 e 2 ligadas

Eventos		**Descrição**
IDi	controlável e forçável	início do deslocamento i
FDi	não-controlável	fim do deslocamento i
RA	não-controlável	retira peça buffer A
RB	não-controlável	retira peça buffer B
RC	não-controlável	retira peça buffer C
EDi	não-controlável	erro de deslocamento i

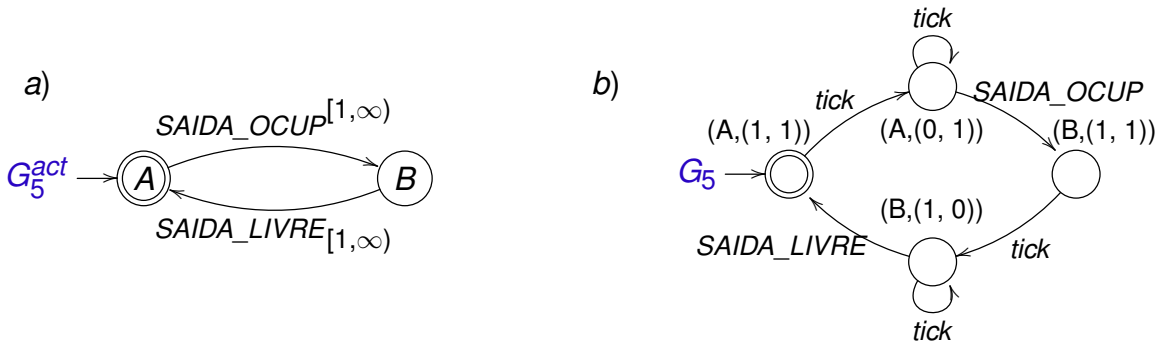
Fonte: Elaborado pelo autor (2021).

G5 – Saída: conforme o GTA da Figura 44, o intervalo de tempo associado aos eventos deste subsistema é [1,∞). O evento *SAIDA_LIVRE* estará elegível em uma unidade de tempo, a partir da atividade inicial ou em qualquer instante depois. E, logo após o mesmo ocorrer, o evento *SAIDA_OCUP* torna-se elegível após uma unidade de tempo ou em qualquer instante depois.

Para o exemplo apresentado, apenas o *tick* é compartilhado entre os subsistemas e a planta global pode ser obtida pelo produto síncrono dos autômatos (GTT) dos subsistemas ($\mathbf{G = G_1||G_2||G_3||G_4||G_5}$). O autômato gerado representa o comportamento da planta global em malha aberta (sem ação de controle) e possui 4416 estados e 22176 transições, assim, utiliza-se para esse cálculo, a ferramenta computacional TTCT [1].

[1] Desenvolvida pelo grupo de pesquisa *Systems Control Group at the University of Toronto*.

Figura 44 – G_5: Saída a) GTA b) GTT



Atividades	
A	saída autorizada
B	saída interrompida

Eventos		Descrição
SAIDA_LIVRE	não-controlável	buffer esteira descarte está vazio
SAIDA_OUCP	não-controlável	buffer esteira descarte está cheio

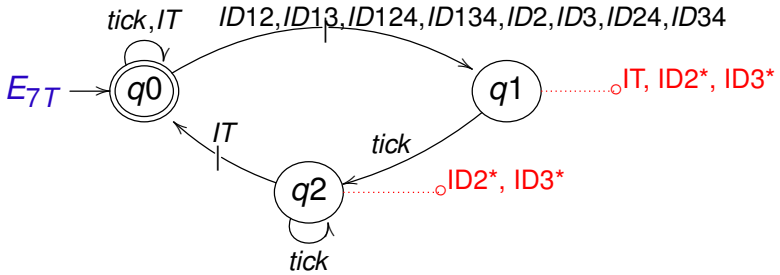
Fonte: Elaborado pelo autor (2021).

4.1.3 Modelagem das especificações temporizadas da Estação de Separação

Os modelos das especificações E1 a E6 serão os mesmos apresentados na Seção 2.2.4, pois o evento do relógio digital (*tick*) não faz parte dessas especificações. Assim, para otimizar o tempo de produção da estação, foram criadas duas novas especificações temporizadas, como segue.

*E*7*T* **– Especificação Temporizada Início de Teste:** conforme apresentado na Figura 45, essa especificação permite que o teste seja iniciado após a ocorrência de um *tick* a partir do início de deslocamento. Isto deve levar a um ganho de produtividade, uma vez que não é necessário aguardar a peça chegar até o buffer de saída para que o início de teste possa ocorrer, como na abordagem não temporizada que especificava a exclusão mútua entre o início de teste e o início de deslocamento.

Figura 45 – GTT: *E*7*T* - Especificação temporizada de início de teste

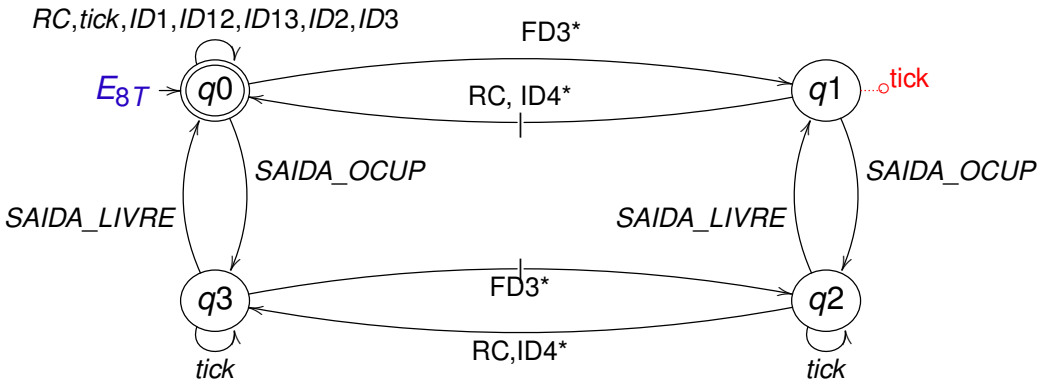


Fonte: Elaborado pelo autor (2021).

*E*8*T* **– Especificação Temporizada de Saída:** essa especificação busca priorizar a saída de peças da estação no menor tempo possível. Conforme a Figura 46, é possível observar que quando a saída estiver livre e o buffer C cheio, o relógio digital global é preemptado e os inícios de deslocamentos *ID*4* são forçados a ocorrer antes

do próximo *tick* [2].

Figura 46 – $E8T$ - Especificação temporizada de saída no menor tempo



Fonte: Elaborado pelo autor (2021).

A especificação global é obtida fazendo-se a composição síncrona das especificações por meio do comando '*sync*' do TTCT, onde $\mathbf{E} = \mathbf{E_1}||\mathbf{E_2}||\mathbf{E_3}||\mathbf{E_4}||\mathbf{E_5}||\mathbf{E_6}||\mathbf{E_{7T}}||\mathbf{E_{8T}}$. O autômato resultante possui 432 estados e 6040 transições. A linguagem alvo é obtida por $K = E||\mathbf{G}$, gerando, assim, o autômato resultante de 21296 estados e 90628 transições, que representa o conjunto de sequências de $\mathbf{G}$, que satisfazem as especificações.

A partir dos modelos temporizados obtidos (planta e especificações), realiza-se a síntese dos supervisores, através da TCS para SEDT, apresentada a seguir por meio de um estudo de caso, a saber, a Estação de Separação.

4.2 SÍNTESE DE CONTROLE SUPERVISÓRIO TEMPORIZADO

Esta seção resume as definições básicas da Teoria de Controle de Supervisório para SEDT. Para maior aprofundamento da teoria, consulte Brandin e Wonham (1994), Wonham e Cai (2019) e Schafaschek, Queiroz e Cury (2017).

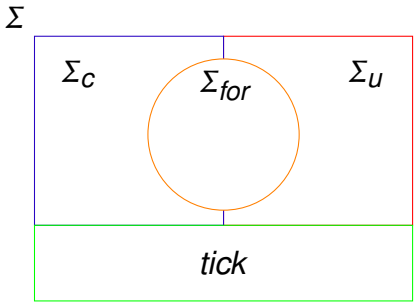
4.2.1 Controle Supervisório Temporizado (CST)

No CST, um supervisor pode efetuar dois tipos de ações de controle sobre SEDTs: desabilitar a ocorrência de eventos *controláveis* no sistema, sendo os eventos passíveis de desabilitação aqueles que podem ser indefinidamente proibidos de ocorrer; e preemptar o *tick* forçando a ocorrência de eventos *forçáveis*, onde o conjunto desses eventos é denotado por Σ_{for}. Assim, Σ_{act} pode ser particionado como $\Sigma_{act} = \Sigma_c \cup \Sigma_u$, onde Σ_c e Σ_u são os subconjuntos disjuntos de eventos controláveis e não-controláveis, respectivamente. O conjunto de eventos forçáveis $\Sigma_{for} \subseteq \Sigma_{act}$ pode ou não ser controlável. O conjunto que contém todos os eventos Σ, nesta abordagem

[2] Os conceitos de preempção e eventos forçáveis serão vistos na Seção 4.2 desta Tese

(ver Figura 47), é dado por $\Sigma = \Sigma_{act} \cup \{tick\}$. Cabe ressaltar que o evento *tick* só pode ser preemptado caso exista algum evento forçável elegível.

Figura 47 – Divisão do alfabeto Σ



Fonte: Elaborado pelo autor (2021).

O supervisor de $\mathbf{G}$ é um mapa $\mathcal{S} : L(\mathbf{G}) \rightarrow 2^{\Sigma}$ que obedece, $\forall s \in L(\mathbf{G})$, as seguintes condições:

(i) $\Sigma_u \subseteq S(s)$.
(ii) $S(s) \cap \Sigma_G(s) \cap \Sigma_{for} = \emptyset$ & $tick \in \Sigma_G(s) \Rightarrow tick \in S(s)$.

Portanto, um evento não-controlável não pode ser desabilitado e o supervisor só pode preemptar o relógio quando $S(s) \cap \Sigma_G(s) \cap \Sigma_{for} \neq \emptyset$.

A planta $\mathbf{G}$ sob supervisão S é denotada por $S/\mathbf{G}$. Além de restringir a ocorrência de eventos, um supervisor pode decidir as cadeias de $L(S/\mathbf{G})$ que permanecem marcadas sob supervisão. Uma linguagem $M \subseteq L_m(\mathbf{G})$ é, portanto, associada a $\mathcal{S}$, e o comportamento marcado de $S/\mathbf{G}$ é dado por $L_m(\mathcal{S}/\mathbf{G}) = L(\mathcal{S}/\mathbf{G}) \cap M$. O supervisor é não bloqueante para $\mathbf{G}$ se $\overline{L_m(S/\mathbf{G})} = L(S/\mathbf{G})$.

A existência de um supervisor não bloqueante S, tal que $L_m(S/\mathbf{G}) = K$ para uma linguagem $K \subseteq L_m(\mathbf{G})$, depende da controlabilidade de K (BRANDIN; WONHAM, 1994). K é dito ser *controlável* com relação a $\mathbf{G}$ se, $\forall s \in \overline{K}$, as seguintes condições são válidas:

i. $\sigma \in \Sigma_u$ & $s\sigma \in L(\mathbf{G}) \;\Rightarrow\; s\sigma \in \overline{K}$;

ii. $s\,tick \in L(\mathbf{G})$ & $\nexists \sigma \in \Sigma_{for} \,|\, s\sigma \in \overline{K} \;\Rightarrow\; s\,tick \in \overline{K}$.

Portanto, K controlável em relação a $\mathbf{G}$ significa que: i) os eventos não-controláveis nunca são desabilitados e; ii) o evento *tick* só pode ser impedido pela preempção com um evento forçável da planta.

Caso a controlabilidade não seja atendida, o conjunto de sublinguagens controláveis de K, denotado $\mathcal{C}(K,\mathbf{G})$, é sempre não vazio e tem um único elemento supremo

$sup\mathcal{C}(K, \mathrm{G})$, que representa o comportamento, minimamente restritivo, que um supervisor pode impor sobre G para respeitar K.

Brandin e Wonham (1994) demonstram que existe um supervisor S, tal que $L_m(\mathcal{S}/\ \mathrm{G}) = K$, se e somente se K for controlável em relação a G. É demonstrado também que, caso a controlabilidade não seja verificada, existe uma máxima sublinguagem controlável de K para SEDTs, dada por $Sup\mathcal{C}(K)$, que representa o comportamento minimamente restritivo, possível de ser imposto por supervisão sobre G de modo a respeitar K.

Nesse exemplo motivador (estação de separação), K é não controlável em relação à G. A máxima linguagem controlável $Sup\mathcal{C}(K,\mathrm{G})$ é modelada por um autômato de 7828 estados e 32590 transições, que corresponde a um supervisor não bloqueante e minimamente restritivo, que garante o cumprimento das especificações de controle. O supervisor reduzido calculado pelo comando *Supreduce* do *TTCT*, possui 204 estados e 1994 transições.

4.2.2 Controle Supervisório Modular Local Temporizado (CMLT)

O controle supervisório para sistemas a eventos discretos temporizados introduzido por Brandin e Wonham (1993) foi estendido para uma abordagem modular local por Schafaschek, Queiroz e Cury (2017), com o objetivo de reduzir a complexidade computacional na síntese de supervisores modulares para sistemas compostos. A proposta explora o fato de que, em muitos casos de interesse prático, a planta a ser controlada é composta por subsistemas assíncronos, com exceção do compartilhamento do relógio. Esta abordagem consiste em obter modelos locais pela composição dos subsistemas apenas afetados em cada uma das especificações. Supervisores, então, são calculados sobre esses modelos, impondo localmente o comportamento especificado. Esta proposta consiste na extensão para o contexto de sistemas a eventos discretos temporizados do controle modular local, proposto por Queiroz e Cury (2000).

Um Sistema Produto Temporizado (SPT) é definido como qualquer SEDT dado por $\mathrm{G} = comp(\mathrm{G_1},...,\mathrm{G_N})$ com $\Sigma_\mathrm{i} \cap \Sigma_\mathrm{k} = \{tick\}$, $\forall i,k \in \{1,....,N\}$, $i \neq k$. Pode-se escrever o produto síncrono do gerador como $\mathbf{G} = ||_{i=1}^{N} \mathbf{G}_i$.

Um supervisor local age sobre os subsistemas $\mathbf{G}_i, i \in I = \{1,...,N\}$, atuando localmente sobre eles. Pode-se dizer que uma especificação afeta o subsistema $\mathbf{G}_i$ se influenciar diretamente a ocorrência de algum dos eventos de $\Sigma_i - \{tick\}$. Assim, as plantas locais para as especificações $E_a \subseteq \Sigma_a^*$ e $E_b \subseteq \Sigma_b^*$ são representadas por: $\mathbf{G}_A = ||_{i \in I_A} \mathbf{G}_i$ e $\mathbf{G}_B = ||_{i \in I_B} \mathbf{G}_i$, onde os conjuntos $I_A = \{i \in I | (\Sigma_i \cap \Sigma_a) - \{tick\} \neq \emptyset\}$ e $I_B = \{i \in I | (\Sigma_i \cap \Sigma_b) - \{tick\} \neq \emptyset\}$. A planta complementar $\mathbf{G}_{AB}^{c}$ representa a parte formada pelos subsistemas que não são afetados por nenhuma especificação, de forma que $\mathbf{G} = \mathbf{G}_A || \mathbf{G}_B || \mathbf{G}_{AB}^{c}$. Assim, o comportamento das plantas locais que atendem

à respectiva especificação é dado por $K_A = E_a||L_m(\mathbf{G}_A)$ e $K_B = E_b||L_m(\mathbf{G}_B)$.

O Problema de Controle Supervisório Modular Local (PCSML) é definido por Schafaschek, Queiroz e Cury (2017) para o caso de duas especificações como segue: dadas duas especificações de controle E_a e E_b, encontrem supervisores locais $S_A : L(\mathbf{G}_A) \rightarrow 2^{\Sigma_A}$ e $S_B : L(\mathbf{G}_B) \rightarrow 2^{\Sigma_B}$ para que a conjunção $S_A \wedge S_B$ seja um supervisor não bloqueante para $\mathbf{G}$ e $L_m(S_A \wedge S_B/\mathbf{G}) \subseteq E_a||E_b||L_m(\mathbf{G})$. A função de conjunção é o mapa $S_A \wedge S_B : L(\mathbf{G}_A)||L(\mathbf{G}_B) \rightarrow 2^{\Sigma_A \cup \Sigma_B}$, de modo que um evento compartilhado de $\mathbf{G}_A$ e $\mathbf{G}_B$ pertença a $(S_A \wedge S_B)(s)$, apenas se for habilitado por ambos os supervisores S_A e S_B, e um evento exclusivo de um dos sistemas pertença a $(S_A \wedge S_B)(s)$ quando sua ocorrência for permitida pelo respectivo supervisor.

Para estabelecer as condições necessárias e suficientes sob as quais a conjunção de dois supervisores locais não bloqueantes $\mathcal{S}_A \wedge \mathcal{S}_B$ é um supervisor global não bloqueante Schafaschek, Queiroz e Cury (2017) apresentam 3 teoremas para o PCSML temporizado:

- Teorema 1 " Sejam $\mathcal{S}_A$ e $\mathcal{S}_B$ supervisores não bloqueantes para $\mathbf{G}_A$ e $\mathbf{G}_B$, respectivamente. $\mathcal{S}_A \wedge \mathcal{S}_B$ define um supervisor para $\mathbf{G}_{AB}$=$\mathbf{G}_A||\mathbf{G}_B$, se e somente se $L(\mathcal{S}_A/\mathbf{G}_A)$ e $L(\mathcal{S}_B/\mathbf{G}_B)$ forem juntamente coercivas com respeito a $\mathbf{G}_{AB}$. Além disso, esse supervisor é não bloqueante para $\mathbf{G}$ se e somente se $L_m(\mathcal{S}_A/\mathbf{G}_A)$, $L_m(\mathcal{S}_B/\mathbf{G}_B)$ e $L_m(\mathbf{G}^c_{AB})$ forem não conflitantes." Portanto nos sistemas temporizados, o compartilhamento de *tick* pode levar a um resultado global bloqueante mesmo se $\mathcal{S}_A \wedge \mathcal{S}_B$ for não bloqueante para $\mathbf{G}_{AB}$. Isso torna necessário que se inclua $L_m(\mathbf{G}^c_{AB})$ no teste de conflito.
- Teorema 2 "A solução ótima local é sempre menos restritiva ou igual a solução global: $sup\mathcal{C}(K_A)||sup\mathcal{C}(K_B)||L_m(\mathbf{G}^c_{AB}) \supseteq sup\mathcal{C}(K)$".
- Teorema 3 "Sejam $\mathcal{S}_A$ e $\mathcal{S}_B$ supervisores não bloqueadores, tais que $L_m(\mathcal{S}_A/\mathbf{G}_A) = sup\mathcal{C}(K_A)$ e $L_m(\mathcal{S}_B/\mathbf{G}_B) = sup\mathcal{C}(K_B)$. Então, $\mathcal{S}_A \wedge \mathcal{S}_B$ é uma solução ótima para PCSML, se e somente se $L_m(\mathcal{S}_A/\mathbf{G}_A)$, $L_m(\mathcal{S}_B/\mathbf{G}_B)$ e $L_m(\mathbf{G}^c_{AB})$ não sejam conflitantes, e $L_m(\mathcal{S}_A/\mathbf{G}_A)$ e $L_m(\mathcal{S}_B/\mathbf{G}_B)$ forem conjuntamente coercivos em relação a $\mathbf{G}_A||\mathbf{G}_B$".

Conforme definido em Schafaschek, Queiroz e Cury (2017), uma coleção de linguagens $L_i, i \in \{1,...,n\}$ é não conflitante quando $||_{i=1}^n \overline{L_i} = \overline{||_{i=1}^n L_i}$. Uma linguagem $L \subseteq L(\mathbf{G})$ é coerciva em relação a $\mathbf{G}$ se, $\forall s \in \overline{L}$, $s.tick \in L(\mathbf{G})$ & $(\nexists \sigma \in \Sigma_{for} \mid s\sigma \in \overline{L})$, então $s.tick \in \overline{L}$. Duas linguagens $L_A \subseteq L(\mathbf{G}_A)$ e $L_B \subseteq L(\mathbf{G}_B)$ são consideradas conjuntamente coercivas em relação a $\mathbf{G}_A||\mathbf{G}_B$, no caso $L_A||L_B$ é coerciva em relação a $\mathbf{G}_A||\mathbf{G}_B$. Assim, a coercividade conjunta significa que o *tick* somente pode ser preemptado por qualquer das duas linguagens, caso exista algum evento forçável σ que

sobreviva a sua sincronização. No Teorema 3 fica estabelecido que: dados dois supervisores locais ótimos, as condições do Teorema 1 são necessárias e suficientes para garantir que a sua conjunção seja um supervisor não bloqueante, que impõe sobre o sistema, o comportamento especificado, de forma minimamente restritiva e, assim, define uma solução ótima para o problema de controle apresentado. Esse resultado sedimenta a metodologia de controle modular local para SED temporizados. Embora os resultados acima tenham sido mostrados para o caso de 2 especificações, eles são facilmente extensíveis ao caso geral de múltiplas especificações.

Ressalta-se que o cálculo do coordenador é um problema global que possui a complexidade da síntese monolítica. Nesse sentido, métodos eficientes para resolver conflitos são propostos na literatura Chen e Lin (2000), Gohari e Wonham (2000), Saadatpoor, Ma e Wonham (2008), Wong e Wonham (1998), Wujie *et al.* (2013), Zhang e Wonham (2002) e mesmo quando o espaço de estados completo tem que ser computado para controle de coordenação, o supervisor resultante leva a uma implementação mais legível e flexível.

Conforme Schafaschek, Queiroz e Cury (2014), para a aplicação da metodologia de CMLT, podem ser seguidos os seguintes passos:

1. obter um representação por sistema produto temporizado para a planta, isto é, modelar cada subsistema com um autômato GTA, de maneira que os eventos regulares (diferentes de *tick*) sejam disjuntos dois a dois $\mathbf{G}_i$ para $i = \{1,...,n\}$;
2. modelar cada especificação como um autômato GTT, e incluir apenas os eventos que se deseja, de alguma forma, restringir sua ocorrência $\mathbf{E}_j$ para $j = \{1,...,m\}$;
3. construir as plantas locais (GTT) relacionadas a cada especificação, compondo os subsistemas que compartilham eventos com ela, diferente de *tick* $G_{l,j}$ para $j = \{1,...,m\}$;
4. obter o comportamento local desejado (GTT), compondo as plantas locais com as especificações correspondentes K_j para $j = \{1,...,m\}$;
5. calcular a máxima sublinguagem controlável (GTT) de cada comportamento local desejado $Sup\mathcal{C}(K_j, G_{l,j})$ para $j = \{1,...,m\}$; e
6. verificar o não conflito e a coercividade conjunta das linguagens resultantes:

   (a) caso ambas as condições sejam satisfeitas, projetar o supervisor não bloqueante que imponha as máximas sublinguagens controláveis sobre as respectivas plantas locais, em que sua ação conjunta resultará em um comportamento global ótimo; e

   (b) caso qualquer umas das condições seja violada, resolver o conflito.

4.2.2.1 Síntese do CMLT para a Estação de Separação

Para a realização da síntese de CMLT para a Estação de Separação, utilizou-se a metodologia apresentada por Schafaschek, Queiroz e Cury (2014). Assim, para a implementação dessas etapas, que serão minuciosamente descritas a seguir, utiliza-se a ferramenta de computação TTCT Wonham (2011), que foi desenvolvida na Universidade de Toronto, Canadá.

Plantas locais: para a síntese dos supervisores modulares, primeiramente, obtêm-se as plantas locais $G_{l,j}$ por meio da composição dos autômatos dos subsistemas G_i, os quais são afetados pelas especificações E_j. A composição foi realizada por meio do comando "*Comp*".

$G_{l1} = Comp(G_1, G_4)$ (24,98);
$G_{l2} = Comp(G_2, G_4)$ (24,110);
$G_{l3} = Comp(G_3, G_4)$ (24,98);
$G_{l4} = G_4$ (12,37);
$G_{l5} = Comp(G_2, G_4)$ (24,110);
$G_{l6} = Comp(G_4, G_5)$ (24,98);
$G_{l7} = Comp(G_2, G_4)$ (24,110);
$G_{l8} = Comp(G_4, G_5)$ (24,98);

Pode-se observar, que as plantas locais G_{l2}, G_{l5} e G_{l7} representam a composição dos subsistemas de entrada (G_2) e deslocamento (G_4). Consequentemente, as mesmas podem ser substituídas por uma única planta local G_{l257}, de 24 estados e 110 transições. Da mesma forma, as plantas locais G_{l6} e G_{l8} podem ser representadas por uma única planta local G_{l68}.

Restrições locais: o segundo passo consiste em obter o comportamento local desejado K_j, para cada planta local $G_{l,j}$. Esse passo é realizado fazendo-se a composição síncrona de cada planta local com a especificação em comum: ($K_j = G_{l,j} || E_j$). Para essa etapa, primeiramente, por meio do comando "*TimedGraph*" é gerado o grafo de transições temporizado de cada planta local, isto é, esse comando transforma um GTA em um GTT. Como por exemplo, o grafo GTT gerado para a planta local $G_{l1} = TimedGraph(G_{l1})$ (92,278), possui 92 estados e 278 transições. O mesmo procedimento foi aplicado a todas as plantas locais, como pode ser observado a seguir.

G_{l257} = TimedGraph(G_{l257})(138,417)
G_{l3} = TimedGraph(G_{l3}) (184,464)
G_{l4} = TimedGraph(G_{l4}) (46,93)
G_{l68} = TimedGraph(G_{l6}) (184,464)

Dessa forma, com o GTT das plantas locais pode-se calcular a restrição local K_j para cada planta local $G_{l,j}$, através do comando "*Sync*". Essa função realiza o produto

síncrono entre dois autômatos, como pode ser visto a seguir, para todas as restrições locais.

K_1 = Sync(G_{l1},E_1) (184,488) Blocked events = None
K_2 = Sync(G_{l257},E_2) (272,802) Blocked events = None
K_3 = Sync(G_{l3},E_3) (180,448) Blocked events = None
K_4 = Sync(G_{l4},E_4) (47,97) Blocked events = None
K_5 = Sync(G_{l257},E_5) (206,525) Blocked events = None
K_6 = Sync(G_{l68},E_6) (178,440) Blocked events = None
K_7 = Sync(G_{l257},E_7T) (198,637) Blocked events = None
K_8 = Sync(G_{l68},E_8T) (188,480) Blocked events = None

Máxima linguagem controlável: no terceiro passo, calcula-se a máxima linguagem controlável para cada restrição local K_j. Essa etapa gera um supervisor local $S_{l,j}$ para o GTT de cada planta local $G_{l,j}$. Foram gerados 8 supervisores locais para o módulo de separação, sendo todos controláveis e não bloqueantes. A máxima linguagem controlável é calculada com o comando "*Supcon*".

S_{l1}= Supcon(G_{l1},K_1) (138,354)
S_{l2} = Supcon(G_{l257},K_2) (254,760)
S_{l3} = Supcon(G_{l3},K_3) (180,448)
S_{l4} = Supcon(G_{l4},K_4) (47,97)
S_{l5} = Supcon(G_{l257},K_5) (206,525)
S_{l6} = Supcon(G_{l68},K_6) (178,440)
S_{l7} = Supcon(G_{l257},K_7) (198,637)
S_{l8} = Supcon(G_{l68},K_8) (188,480)

É possível observar, por exemplo, que a máxima linguagem controlável gerada para o supervisor $\mathcal{S}_{l1}$, pode ser representado por um autômato de 138 estados e 354 transições.

Comportamento em conjunto dos supervisores: no quarto passo, verifica-se o não conflito e a coercividade conjunta das linguagens resultantes. Para garantir que não haja conflito entre as ações dos supervisores locais em conjunto, é preciso verificar se existe um conjunto de supervisores conflitantes que impedem que o controle modular local seja diretamente aplicado. Caso não exista, o resultado da ação modular é o mesmo de uma ação centralizada.

Por meio da composição síncrona dos 8 supervisores locais $\mathcal{S} = \mathcal{S}_{l1}|| \mathcal{S}_{l2}||\mathcal{S}_{l3}$ $||\mathcal{S}_{l4}$ $||\mathcal{S}_{l5}||$ $\mathcal{S}_{l6}$ $||\mathcal{S}_{l7}$ $||\mathcal{S}_{l8}$, obteve-se um supervisor de 14448 estados e pela composição síncrona dos 5 subsistemas $\mathbf{G} = \mathbf{G}_1||\mathbf{G}_2$ $||\mathbf{G}_3$ $||\mathbf{G}_4$ $||\mathbf{G}_5$, uma planta global de 4416 estados. Verifica-se, assim, por meio do comando "*Trim*", que o supervisor gerado é não bloqueante e pelo comando "*Condat*" que é não controlável. Logo, os supervisores locais são não conflitantes e não juntamente coercivos em relação a $\mathbf{G}$. Isto significa

que, quando o *tick* for preemptado, nenhum evento forçável sobrevive à sincronização dos mesmos.

$S_{l12345678}$ = Sync($S_{l1234567}$,S_{l8}) (14448,61858) Blocked_events = None
true = Nonconflict(G,$S_{l12345678}$)
$S_{l12345678}$ = Trim($S_{l12345678}$) (14448,61858)
$CDS_{l12345678}$ = Condat(G,$S_{l12345678}$) Uncontrollable

Uma forma de resolver o conflito anteriormente apresentado, seria aplicar o controle monolítico e, dessa forma, descartar os supervisores locais. Porém, como o problema de conflito está restrito a uma parte específica, pode-se calcular um coordenador para o conjunto de especificações conflitantes. Compondo todas as plantas locais em malha fechada, é obtida a planta global conflitante $\mathbf{G}_{\text{conf}}$, dada por:

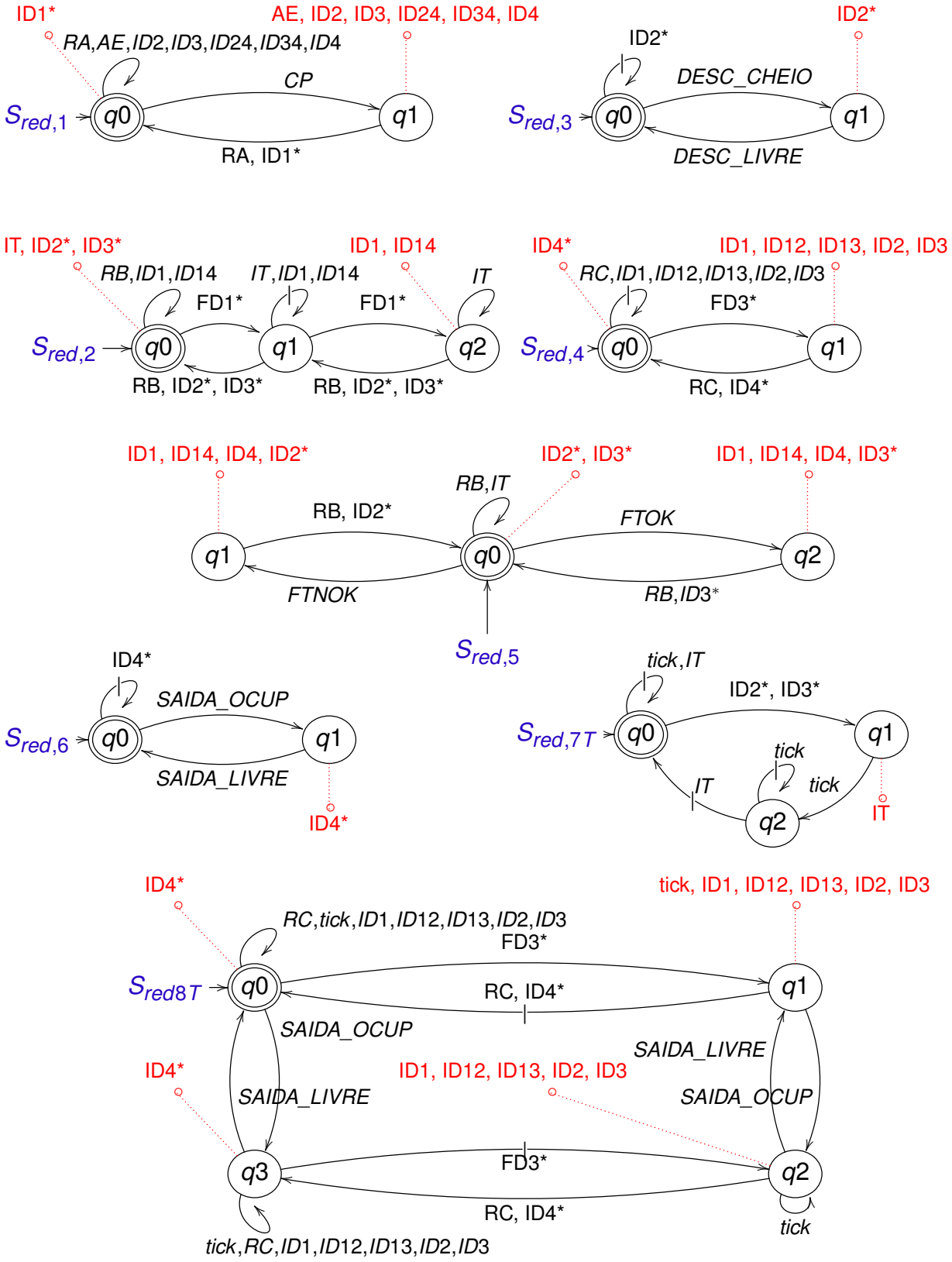
$$\mathbf{G}_{\text{conf}} = \mathcal{S}_{l1} \wedge \mathcal{S}_{l2} \wedge \mathcal{S}_{l3} \wedge \mathcal{S}_{l4} \wedge \mathcal{S}_{l5} \wedge \mathcal{S}_{l6} \wedge \mathcal{S}_{l7} \wedge \mathcal{S}_{l8} / \mathbf{G}_1 \,||\, \mathbf{G}_2 \,||\, \mathbf{G}_3 \,||\, \mathbf{G}_4 \,||\, \mathbf{G}_5 \,,$$

Conforme o Teorema 2 de Schafaschek, Queiroz e Cury (2017), a solução ótima é sempre menos restritiva ou igual a global. Então, para resolver este conflito sobre $\mathbf{G}_{conf}$, obtêm-se o coordenador pelo cálculo da máxima linguagem controlável $Sup\mathcal{C}(L_m(\mathbf{G}_{conf}))$. A ação do coordenador juntamente com os supervisores locais $\mathcal{S}_{l1}$ a $\mathcal{S}_{l8}$, leva o sistema ao comportamento desejado (*Coord* = Supcon(G_{conf},G) (7828, 32590). Assim, o coordenador *Coord* é um supervisor que resolve o problema do conflito sobre $\mathbf{G}_{\text{conf}}$.

Supervisores reduzidos: como os supervisores locais encontrados possuem um número grande de estados (254) e transições (760), realiza-se a redução dos supervisores locais por meio do comando "*Supreduce*". Obteve-se como resultado da abordagem modular local, 8 supervisores reduzidos locais de 2 e 4 estados, conforme a Figura 48, que mostra os supervisores locais reduzidos e as suas desabilitações, e um coordenador reduzido de 204 estados e 1994 transições.

SR_1 = Supreduce(G_{l1},S_l1) (2,63;slb=2)
SR_2 = Supreduce(G_{l257},S_l2) (3,86;slb=3)
SR_3 = Supreduce(G_{l3},S_l3) (2,70;slb=2)
SR_4 = Supreduce(G_{l4},S_l4) (2,40;slb=2)
SR_5 = Supreduce(G_{l257},S_l5) (3,89;slb=3)
SR_6 = Supreduce(G_{l68},S_l6) (2,68;slb=2)
SR_7 = Supreduce(G_{l257},S_l7) (3,73;slb=3)
SR_8 = Supreduce(G_{l68},S_l8) (4,83;slb=3)
SR_{Coord} = Supreduce($Sl12345678$,*Coord*) (204,1994;slb=56)

Figura 48 – Supervisores Locais Reduzidos



Fonte: Elaborado pelo autor (2021).

Já a Tabela 4 mostra de uma forma resumida, o número de estados dos autômatos gerados para a solução do problema. Pode-se observar que para j=3,4,5,6 $\mathcal{S}_{l,j}$ = K_j, logo, pode-se usar diretamente a especificação E_j como um supervisor reduzido, sem a necessidade de usar o algoritmo de redução.

Tabela 4 – Número de estados dos autômatos da síntese

j	E_j	$G_{l,j}$	K_j	$\mathcal{S}_{l,j}$	$\mathcal{S}_{r,j}$
1	2	92	184	113	2
2	3	138	272	254	2
3	2	184	180	180	2
4	2	46	47	47	2
5	3	138	206	206	3
6	2	184	178	178	2
7	3	138	198	198	3
8	4	184	188	188	4

Fonte: Elaborado pelo autor (2021).

4.2.2.2 Simulação temporizada da Estação de Separação

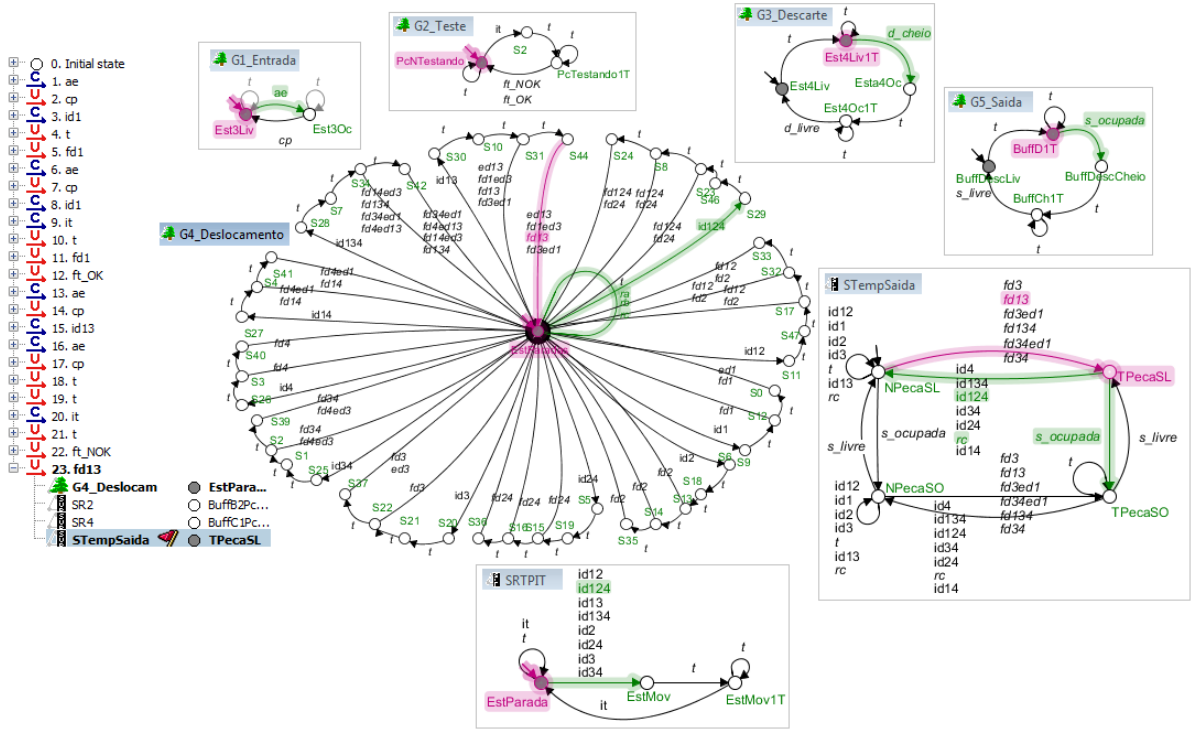
A emulação permite visualizar, por meio de animações gráficas, o comportamento resultante entre os autômatos: das plantas; das especificações; e dos supervisores temporizados. É possível observar o progresso do sistema em diferentes cenários e a sequência de eventos gerados, facilitando a identificação de possíveis erros de modelagem. O modelo encontrado foi testado no simulador do Supremica que não trata de modelos temporizados. O evento *tick* foi definido como um evento não-controlável, pois sempre que o mesmo for preemptado, o simulador indicará um alerta, e a simulação só deve continuar caso exista um evento forçável habilitado. Cabe ressaltar que este processo de simulação para sistemas temporizados é totalmente manual.

Após simular e analisar o modelo para diferentes cenários e buscando otimizar a produção, fez-se necessário alterar novamente algumas especificações, como por exemplo, o início do teste pode ficar aguardando por até três *ticks*, até que o final de deslocamento 3 (*FD*3) ocorra. Logo, nesta situação, o sistema fica ocioso durante esse intervalo de tempo. Para otimizar a produção, substitui-se a especificação de exclusão mútua (*E*7) entre o início do teste *IT* e o início de movimento das esteiras *IDi*, por uma especificação temporizada (*E*7*T*). Essa nova especificação impede apenas que o (*IT*) ocorra imediatamente após o início de algum deslocamento (*ID*2* e *ID*3*), restringindo o (*IT*) em um *tick*.

Para ilustrar o comportamento do sistema com a ação dessa especificação temporizada, a Figura 49 traz o controle supervisório sendo executado em um cenário,

onde o sistema opera com três peças em paralelo, isto é, com uma peça na entrada, uma peça testada e outra na saída.

Figura 49 – Simulação Cenário 2 - Preempção *tick*



Fonte: Elaborado pelo autor (2021).

Como pode ser observado pela sequência de eventos gerados, os passos “2”, “7”, “14” indicam a chegada de peças no sistema e os passos “12” e “22” o fim de teste. No passo ”23”, o controlador atua e desabilita o evento controlável *AE*, preempta o *tick* e força o evento *ID*124, resolvendo o problema de ociosidade do sistema, descrito anteriormente.

4.3 CONCLUSÃO DO CAPÍTULO

A metodologia de controle supervisório para sistemas a eventos discretos temporizados mostrou-se satisfatória para a síntese dos supervisores modulares locais temporizados. Os resultados mostraram que os supervisores locais temporizados estabelecem condições necessárias e suficientes para garantir que a ação conjunta desses confira ao sistema um comportamento global não bloqueante e minimamente restritivo. A modelagem de especificações temporizadas mostrou-se capaz de resolver o problema de conflito entre os supervisores locais, além de calcular o tempo mínimo para a execução das tarefas. No próximo capítulo, esses modelos temporizados serão implementados em um CLP para realizar o controle temporizado do sistema real.

5 PROPOSTA DE IMPLEMENTAÇÃO DO CONTROLE SUPERVISÓRIO MODULAR LOCAL TEMPORIZADO

Nesta seção, apresenta-se uma estrutura de implementação do sistema de controle para sistemas a eventos discretos temporizados, com base na estrutura preliminar de implementação de três níveis, proposta por Queiroz e Cury (2002). Após a apresentação geral da estrutura proposta, analisa-se como ela pode tratar os principais problemas de implementação em CLP, destacados na literatura e, na sequência, ilustra-se a proposta com a sua aplicação à Estação MPS de Separação, com os supervisores temporizados calculados no Capítulo 4.

5.1 ARQUITETURA DE IMPLEMENTAÇÃO

Como a implementação do controle supervisório temporizado não está bem estabelecida na literatura, esta Tese propõe uma estrutura que estende a arquitetura de Queiroz e Cury (2002), como apresentado na Figura 50, em que foi introduzido o evento *tick* para o tratamento do relógio digital global e o conjunto de eventos forçáveis. Nesta nova arquitetura, o sistema de controle é todo implementado no CLP e possui os seguintes módulos:

(i) um conjunto de Supervisores Modulares (SM) obtido com a aplicação da abordagem modular local temporizada, da teoria de controle supervisório;

(ii) um Relógio Digital Global (RDG) que gera o evento *tick*, para representar a passagem de uma unidade de tempo do relógio, o qual assume valores temporais discretos, determinados pelo número de ocorrências de *tick*;

(iii) uma representação por Sistema Produto Temporizado (SPT), composto de subsistemas concorrentes e assíncronos com exceção do evento *tick*, que descreve o comportamento livre do sistema de manufatura a ser controlado;

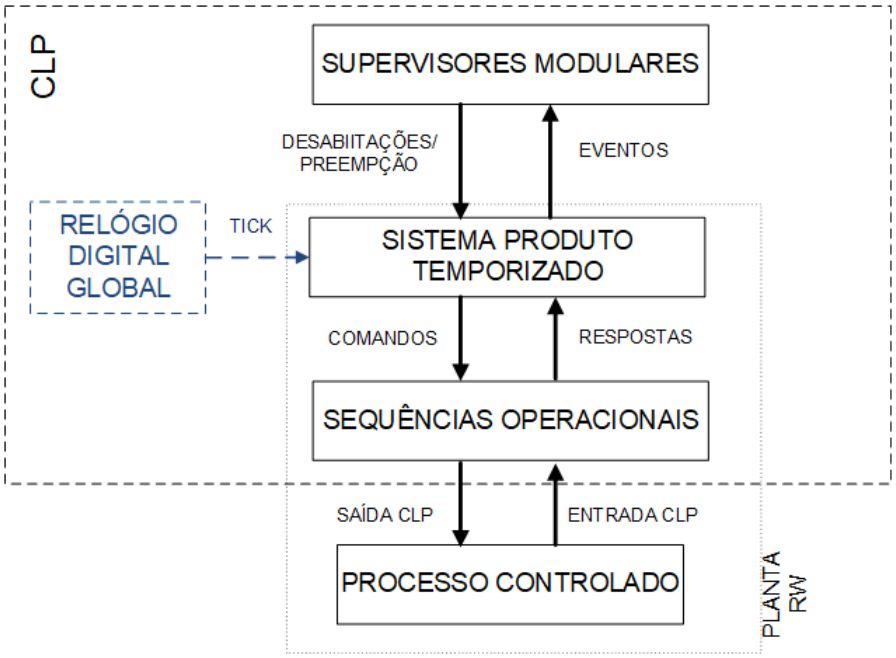
(iv) um conjunto de Sequências Operacionais (SO) que detalha as atividades a serem realizadas pelo sistema a ser controlado; e

(v) um processo a ser controlado.

Assim, a implementação dos autômatos gerados pela síntese do CMLT é tratada em dois níveis: Supervisores Modulares Reduzidos como GTTs; e Sistema Produto Temporizado como GTAs.

Para uma melhor análise, algumas questões serão levantadas, de forma a garantir o correto tratamento dos eventos. A seguir detalham-se os módulos da arquitetura

Figura 50 – Arquitetura de Controle Supervisório Temporizado.



Fonte: Adaptado de Queiroz e Cury (2002).

proposta, bem como, um estudo sobre os possíveis problemas que podem ocorrer em sua implementação, com o objetivo de consolidar a arquitetura proposta.

5.1.1 Relógio Digital Global

Este módulo é responsável pela geração espontânea dos eventos *tick*, que alterará as máquinas de estados, tanto para o SPT como para SM. Dois pontos importantes devem ser considerados para a correta operação da estrutura de controle:

- i) para o ajuste entre o relógio digital e o ciclo do CLP, o evento *tick* é gerado de forma espontânea, por meio de um relógio digital e sempre que um *tick* for gerado pelo RDG, o SPT receberá um sinal (variável *Ae_* < *tick* >). É importante que a ocorrência do evento *tick* seja sincronizada com o início do ciclo de varredura do CLP, para que o início e término dos mesmos sejam coincidentes, pois o CLP realiza, antes do início do ciclo de varredura, a leitura de todas as entradas, e somente a partir de então, o *tick* é gerado, mantendo uma coerência com os sinais de entrada. Assume-se aqui, que o período do relógio sempre será um múltiplo do período de varredura do CLP; e

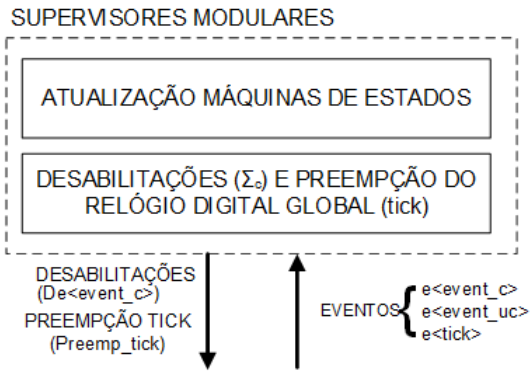
- ii) para evitar a ocorrência de múltiplos eventos do relógio global dentro do mesmo ciclo do CLP, o período do CLP deve ser menor do que o período do relógio digital global. Em sistemas de manufatura, isso é admissível, visto que os tempos de ciclo de varredura de CLP, da ordem de 50ms, são muito pequenos em relação à dinâmica do processo.

5.1.2 Supervisores Modulares

Da mesma forma que em Queiroz e Cury (2002), os supervisores modulares reduzidos podem ser aplicados à arquitetura temporizada, exatamente como gerados pela TCS. Nesse nível, os supervisores atualizam os estados ativos, e um mapa associa esses estados a um conjunto de sinais de desabilitações de eventos controláveis e preempção do evento do relógio digital global. Cabe destacar que a máquina de estados evolui com o evento *tick* e a ação dos supervisores, também, inclui preempção do *tick*.

Como ilustrado na Figura 51, o nível SM recebe como entrada os sinais dos eventos controláveis (Σ_c), dos eventos não-controláveis (Σ_u) e do evento *tick*, gerados pelo SPT.

Figura 51 – Arquitetura Temporizada - Nível SM

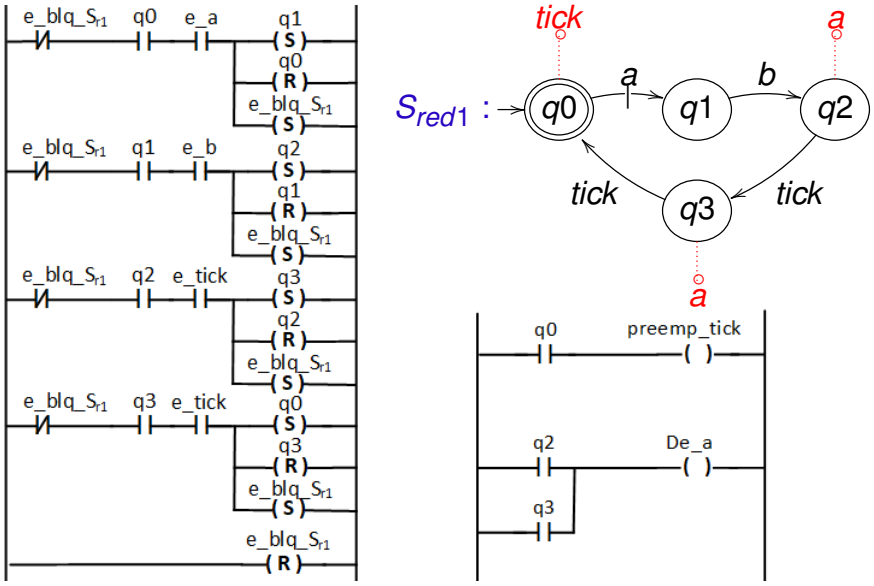


Fonte: Elaborado pelo autor (2021).

A implementação do conjunto de supervisores reduzidos, nessa arquitetura, é dividida em duas etapas: atualização das máquinas de estados; e desabilitações e preempção. A primeira parte é responsável pela atualização dos estados ativos dos supervisores, de acordo com a estrutura de seus geradores e com a ocorrência dos eventos Σ_c, Σ_u e $\{tick\}$, sinalizados pelo SPT (gerados pela planta). A segunda parte trata das desabilitações de eventos controláveis (variável *De_ < event_c >*), que não são permitidos de ocorrer na planta, e a preempção do evento do relógio digital (variável *Preemp_ < tick >*). Nos estados dos supervisores em que o *tick* é desabilitado, a controlabilidade garante que sempre haja, pelo menos, um evento forçável elegível na planta. Assim, o nível SM informa o sinal *Preemp_ < tick >* para que, no SPT, seja forçado um evento antes do próximo *tick*.

A implementação das máquinas de estado é similar ao que foi apresentado na Seção 3.2, visto que os supervisores modulares são representados por GTTs, que apenas se diferem dos autômatos não temporizados por possuírem transições disparadas pelo evento *tick*. A Figura 52 ilustra, como exemplo, a implementação do supervisor temporizado (S_{red1}) na linguagem *Ladder*.

Figura 52 – Exemplo de implementação de supervisor temporizado na linguagem LD



(a) Máquina de estados (b) Ação de controle

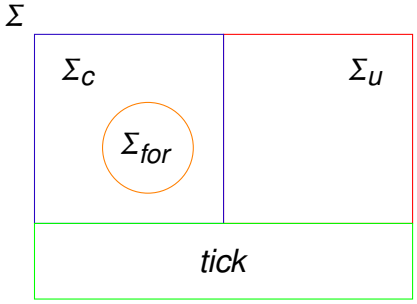
Fonte: Elaborado pelo autor (2021).

5.1.3 Sistema Produto Temporizado

Como na abordagem não temporizada de Queiroz e Cury (2002), propõe-se, neste nível, que o SPT represente o Sistema Real em conjunto com uma interface para o condicionamento de sinais de desabilitação e de ocorrência de eventos. As principais mudanças referem-se a implementação dos subsistemas como GTAs, a adição da ação de controle sobre o evento *tick* e o forçamento de eventos. Assume-se, nesta arquitetura temporizada, que todos os eventos forçáveis são também controláveis (mas nem todos os controláveis são forçáveis), uma vez que nos modelos de sistemas de manufatura, normalmente todos os eventos que podem ser forçados são comandados pelo CLP e, portanto, também podem ser impedidos de ocorrer.

A Figura 53 ilustra a divisão dos conjuntos de eventos adotada nesta arquitetura, que se dividem em quatro tipos, quais sejam: 1) não-controláveis não-forçáveis; 2) controláveis não-forçáveis; 3) controláveis forçáveis; e o 4) *tick*. Essa divisão será utilizada para se definir a ordem de implementação no nível SPT.

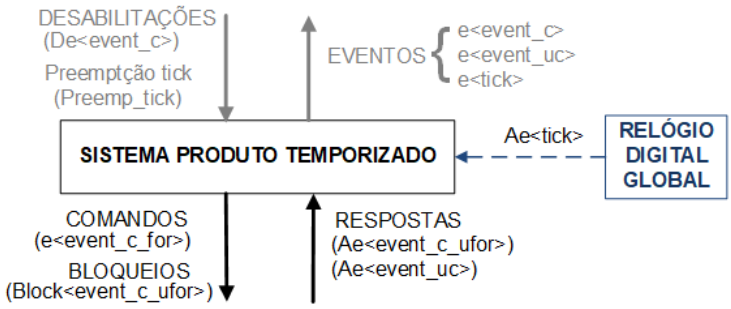
Figura 53 – Divisão do alfabeto Σ para arquitetura temporizada



Fonte: Elaborado pelo autor (2021).

Pelas características de sistemas de manufatura e no intuito de simplificar a apresentação, nesta arquitetura, assume-se, ainda, que os eventos controláveis forçáveis sejam associados a comandos, e os eventos não-controláveis e os controláveis não-forçáveis sejam associados a respostas. Assim, como observado na Figura 54, o SPT recebe sinais (respostas) de eventos não-controláveis não-forçáveis e controláveis não-forçáveis vindos da SO (variáveis *Ae_* < *event_uc* > e *Ae_* < *event_c_ufor* >), recebe, também, o sinal do evento *tick* gerado pelo RDG (variável *Ae_* < *tick* >) e, conforme o comportamento gerado para a planta, sob a ação de controle dos supervisores, envia comandos para a ocorrência dos eventos controláveis forçáveis (variável *e* < *_event_c_for* >) e sinais de bloqueios (variáveis *Block_* < *event_c_ufor* >.

Figura 54 – Arquitetura Temporizada - Nível SPT



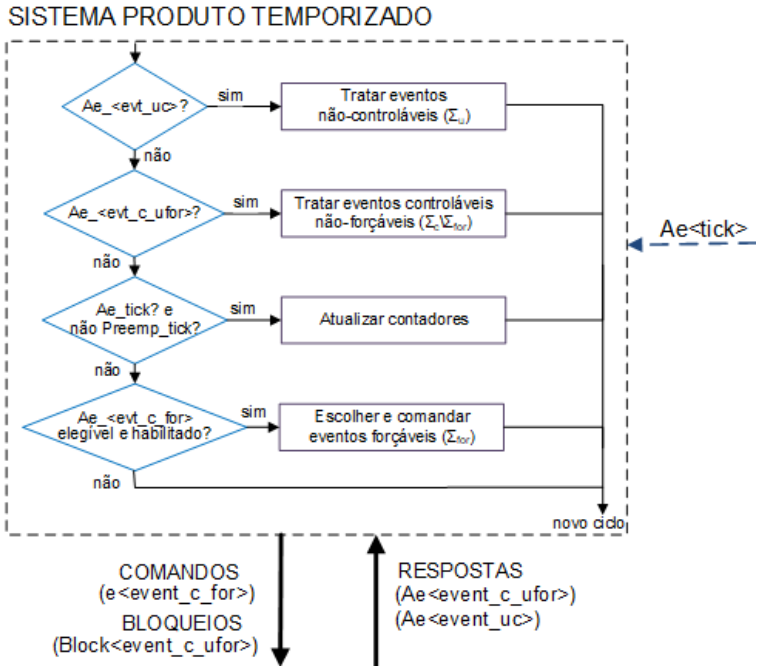
Fonte: Elaborado pelo autor (2021).

Nessa arquitetura, o SPT é implementado como autômatos GTAs (por meio de cronômetros), gerando modelos mais compactos e, consequentemente, ocupando menos memória do CLP do que se implementado como GTTs (transições do *tick*), uma vez que, quanto maior for o período de relógio, maior será o código. Para comandar a execução dos subsistemas modelados como GTAs, esses devem ser implementados na forma de máquinas de estados concorrentes. Assim, será realizado o tratamento de apenas um evento (uma transição) por ciclo de varredura do CLP, com exceção do *tick*, que é o único evento compartilhado entre os GTAs e pode gerar várias transições no mesmo ciclo do CLP. O tratamento de apenas um ciclo de varredura poderia ser flexibilizado, segundo a estrutura proposta no artigo de Leal, Cruz e Hounsell (2009).

Para evitar a execução de transições indesejáveis, deve-se garantir, então, que a ação dos supervisores modulares esteja sempre atualizada, antes que uma nova transição controlável ou forçável ocorra no SPT, e que a sinalização de ocorrência de eventos para o SM seja sequencial (uma por ciclo de varredura). Assim, para garantir uma execução consistente dos subsistemas (G^{act}) implementados, deve-se respeitar a ordem de prioridade, como pode ser observado na Figura 55, onde: 1) as transições elegíveis dos eventos não-controláveis não-forçáveis são processadas antes dos outros eventos no código do CLP, uma vez que são orientadas a partir de respostas anteriores da SO; 2) as transições elegíveis dos eventos controláveis não-forçáveis são

processadas, já que também são orientadas a partir de respostas da SO e podem ser adiadas; 3) na sequência, caso nenhuma transição tenha sido tratada, se o *tick* ocorrer e não estiver preemptado, então nenhuma transição de eventos forçáveis estará ativa, logo, o evento do relógio digital global pode ser tratado e os cronômetros associados aos eventos controláveis, atualizados; e, por fim, 4) se o *tick* estiver sendo preemptado, então, as transições elegíveis dos eventos forçáveis devem ser processadas, já que devem ocorrer antes do próximo *tick*.

Figura 55 – Nível SPT - Ordem de prioridade dos eventos



Fonte: Elaborado pelo autor (2021).

A transição de um evento não-controlável é aceita somente se o seu estado predecessor estiver ativo e seu evento correspondente estiver sendo sinalizado pela SO ou pelo RGD. Já a transição de um evento controlável é aceita somente se o seu estado predecessor estiver ativo e se não estiver sendo desabilitada por nenhum dos supervisores (pela ação conjunta dos supervisores). E uma transição forçável é aceita somente se o seu estado predecessor estiver ativo e se o relógio estiver sendo preemptado por algum supervisor.

Nessa proposta, os subsistemas são implementados como GTAs e os tempos associados aos eventos devem ser tratados. Assim, os tempos associados aos eventos controláveis (Σ_c) necessitam ser cronometrados, uma vez que a informação do valor do tempo é uma condição necessária para a habilitação das transições associadas a esses eventos. Já os tempos associados aos eventos não-controláveis (Σ_u) não necessitam ser implementados, bastando apenas observar a sua ocorrência vinda da SO. Nesse caso, serão implementados apenas os cronômetros para os eventos

controláveis.

Para implementar os tempos associados aos eventos (Σ_c), utilizam-se contadores de pulsos, que contabilizam "eventos", independentemente do instante de tempo em que eles ocorram. Todos os contadores devem ser atualizados no mesmo ciclo de varredura.

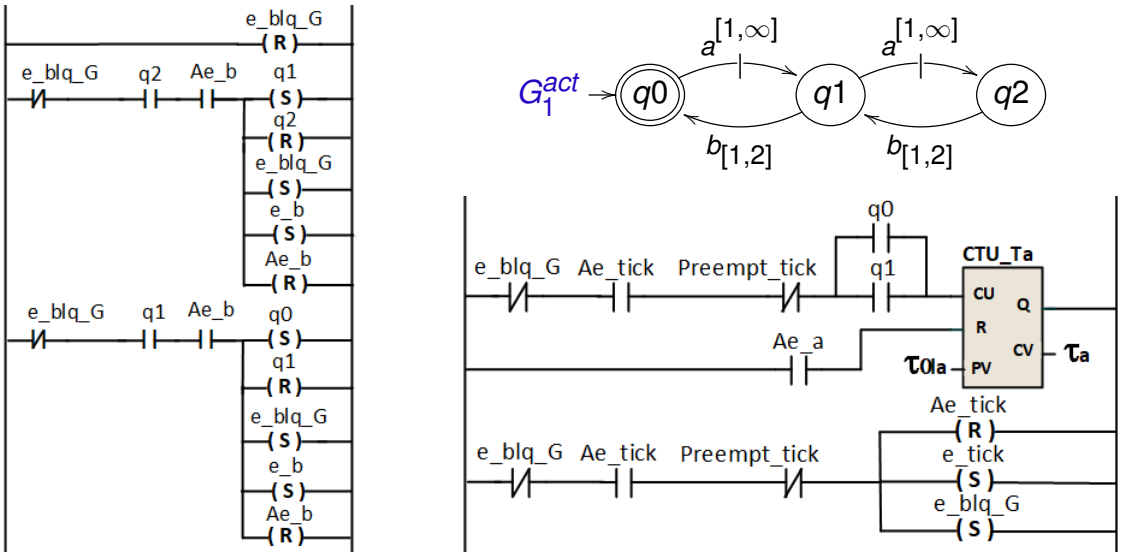
Um problema que poderia ocorrer na implementação dos tempos, é o fato do evento *tick* incrementar vários pulsos consecutivos em um contador, em um mesmo ciclo de varredura do CLP (efeito avalanche). Assim, cada vez que a variável *Ae_tick* for gerada pelo relógio digital, deve-se restringir a contagem de apenas um pulso em cada contador ativo. Além disso, o período do relógio digital tem de ser, pelo menos, o dobro da frequência do ciclo de varredura, senão a variável *Ae_tick* ficará ativa em todos os ciclos e o contador não detectará a borda de subida (não realizará a contagem).

A Figura 56 mostra um exemplo de como é possível implementar um subsistema em GTA no nível SPT, de forma a evitar o problema de efeito avalanche. Nesse exemplo, a Figura 56a ilustra a implementação da máquina de estados para os eventos não-controláveis. Se o subsistema estiver no estado $q2$ e ocorrer o evento b, isso poderia resultar na ativação do estado $q0$, uma vez que o $q1$ será setado e resetado no mesmo ciclo de varredura, gerando o problema do efeito avalanche. Já a Figura 56b ilustra a implementação do contador associado ao evento controlável a. O contador será incrementado se, e somente se: nenhuma transição da máquina de estados tenha ocorrido (*e_blq_G*); a variável *Ae_tick* ocorrer; o *tick* não for preemptado (NOT *Preempt_tick*); e o subsistema G_1^{act} estiver no estado $q0$ ou $q1$. Cabe ressaltar que, como o contador conta um pulso a partir da detecção da borda de subida, o mesmo é incrementado em apenas uma unidade por ciclo de varredura e, assim, o problema do efeito avalanche não ocorrerá.

5.1.4 Sequências Operacionais

Neste nível e da mesma forma que na abordagem não temporizada, as sequências operacionais funcionam como uma interface entre o sistema produto temporizado e o sistema real. As sequências operacionais geram os sinais de saída do sistema de controle (entradas do sistema real - entrada do CLP) e leem os sinais de entradas (saídas do sistema real - saídas do CLP), fornecendo ao SPT, respostas lógicas que refletem a ocorrência de eventos não-controláveis ou de eventos controláveis não-forçáveis (variáveis *Ae_* < *event_uc* > ou *Ae_* < *event_c_ufor* >), fazendo a comunicação com o sistema físico a ser controlado. Conforme Queiroz e Cury (2002), as sequências operacionais permitem realizar um detalhamento do processo de modelagem da planta e das especificações, o que simplifica e reduz a complexidade de síntese dos supervisores.

Figura 56 – Problema do efeito avalanche - Nível SPT



(a) implementação máquina de estados - evento não-controlável *b* com *evt_blq_G*

(b) implementação contador - evento controlável *a* com *evt_blq_G*

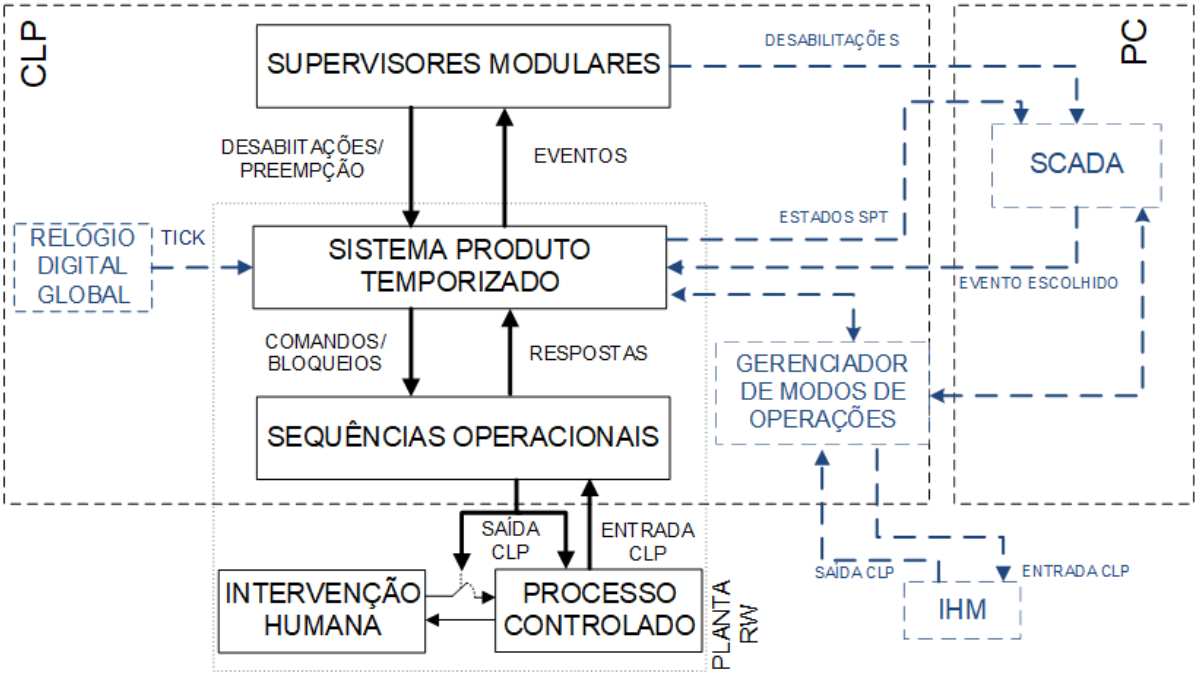
Fonte: Elaborado pelo autor (2021).

Segundo Vieira (2007), em geral, cada evento controlável em Σ_c é associado a uma sequência operacional. Na arquitetura temporizada proposta, cada comando (como o comando forçar eventos) gerado pelo SPT será associado a um procedimento operacional distinto. Na implementação das SOs, cada comando gerado pelo SPT, é associado a uma variável (*e_* < *event_c* >). A desativação do comando (*e_* < *event_c* >=0) confirma que o mesmo já foi processado pela sequência operacional correspondente. Assim, o programa fica aguardando a ocorrência de um determinado evento (variáveis *Ae_* < *event_uc* > e *Ae_* < *event_c_ufor* >), que será sinalizado para o SPT.

A Figura 57 apresenta a arquitetura temporizada completa, em que é adicionado: i) o Gerenciador de Modos de Operação (GMO), que permite definir o modo de operação a ser executado pelo SPT; e ii) o sistema SCADA, que recebe informações de desabilitações e preempção do SM, troca informações sobre os modos de operações com o GMO, observa estados do SPT, e retorna eventos escolhidos para o SPT atualizar a planta e gerar comandos e sinais de bloqueio para a SO.

A partir do exposto, passa-se a uma análise relativa aos problemas de implementação do controle supervisório temporizado em CLPs.

Figura 57 – Arquitetura de Controle Supervisório Temporizado.



Fonte: Elaborado pelo autor (2021).

5.2 ANÁLISE DA ARQUITETURA DE IMPLEMENTAÇÃO DE CONTROLE SUPERVISÓRIO TEMPORIZADO EM CLPS

Nesta seção serão tratados os principais problemas relativos a implementação do CMLT em CLPs, como base no estudo apresentado na Seção 3.1, analisando assim, como esses problemas se comportarão na nova arquitetura temporizada proposta.

5.2.1 Causalidade

Na arquitetura temporizada aqui proposta, aborda-se a questão da causalidade da mesma forma que em Queiroz e Cury (2002), em que a função dos supervisores modulares é seguir a sequência de eventos gerados e habilitar ou desabilitar eventos controláveis, de acordo com a sua definição original, apresentada por Wonham e Ramadge (1988). Os subsistemas são implementados no CLP e responsáveis pela geração dos eventos controláveis, que correspondem a acionamentos na saída do CLP. Os supervisores seguem mantendo a sua função original de apenas estabelecer as desabilitações de eventos controláveis e preempção do *tick*. Já o Relógio Digital tem a função de gerar o evento *tick* que alterará as máquinas de estados, tanto para o SPT como para o SM.

5.2.2 Efeito Avalanche

Segundo Fabian e Hellgren (1998), o efeito avalanche ocorre quando a mudança de valor em um sinal do CLP é registrada por intermédio de um evento e esse provoca a transição indevida de mais de um estado do sistema, em um mesmo ciclo de varredura do CLP.

Na arquitetura temporizada aqui proposta, o problema do efeito avalanche poderia ocorrer, tanto no nível SM quanto no nível SPT. No nível SM poderia ocorrer na atualização das máquinas de estados dos supervisores ($\mathbf{S}r_i$), e no SPT, de duas formas, na atualização das máquinas de estados dos subsistemas ($\mathbf{G}_i^{act}$) e dos tempos associados aos eventos $\sigma \in \Sigma_c$ (atualização dos cronômetros).

No nível dos supervisores modulares, a implementação das máquinas de estados correspondentes aos respectivos GTTs segue a mesma estrutura da Figura 22b, pois o *tick* será tratado como um evento qualquer. A proposta para evitar que o efeito avalanche ocorra é similar a apresentada na Seção 3.1.1, em que uma variável do CLP (*evt_blq_Sri*) é associada a cada supervisor, impedindo que mais de uma transição ocorra dentro de um mesmo ciclo de varredura. Essa variável deve ser ativada sempre que uma transição de estado ocorrer e desativada no final do código, em que a estrutura do supervisor está implementada. A Figura 52a, vista na Seção 5.1.2, ilustra o problema do efeito avalanche na implementação do supervisor temporizado (S_{red1}).

O problema do efeito avalanche no nível SPT é ilustrado na Figura 56, visto na Seção 5.1.3. Na abordagem temporizada, os eventos Σ_c e Σ_u não são compartilhados, logo, existe só um autômato que pode ser atualizado por ciclo de varredura. Dessa forma, como ilustrado na Figura 56a, apenas uma variável de bloqueio (*evt_blq_G*) é suficiente para garantir que cada autômato execute apenas uma transição por ciclo de varredura do CLP. Já para a atualização dos cronômetros, conforme ilustrado na Figura 56b, o uso do contador resolve o problema do efeito avalanche, uma vez que incrementa apenas uma unidade por ciclo de varredura.

5.2.3 Ordem de eventos ocorridos no mesmo ciclo de varredura

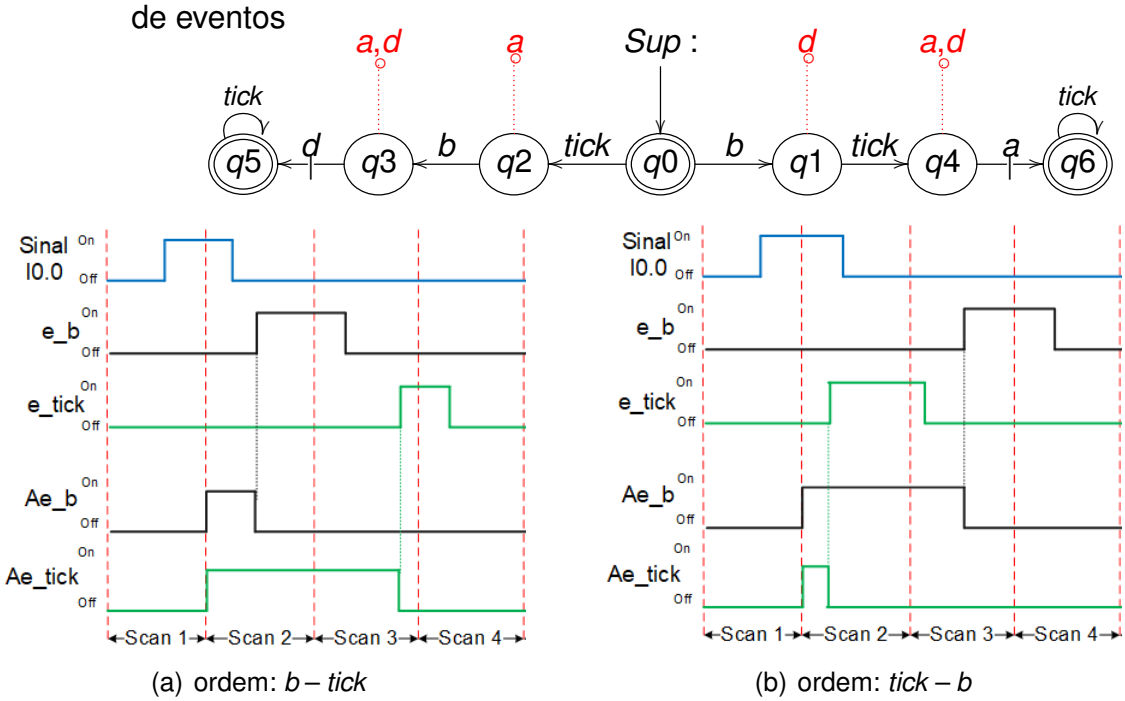
A insensibilidade ao entrelaçamento, proposta em Fabian e Hellgren (1998), é uma propriedade que evita que um supervisor adote uma ação de controle incorreta em função de não reconhecer a ordem de ocorrência dos eventos não-controláveis ativos no mesmo ciclo de varredura do CLP. Essa propriedade verifica se todas as sequências de eventos não-controláveis consecutivos, levam à mesma ação de controle, independentemente da ordem de observação.

Na arquitetura temporizada proposta, os eventos controláveis são gerados pelo CLP e, apesar de que o evento *tick* também seja gerado pelo CLP, não é possível

alterar o instante em que ele ocorre e nem desabilitá-lo. O evento *tick* é gerado pelo RDG e do mesmo modo que ocorre com os eventos não-controláveis, ele é sinalizado no SPT. Assim, assume-se que o evento *tick* deve ter o mesmo comportamento de um evento não-controlável.

Dessa forma, uma vez que é possível ter vários eventos não-controláveis ativos em um mesmo ciclo e o controlador pode não reconhecer qual ocorreu primeiro, a Figura 58 mostra que diferentes prioridades no SPT levam a ações diferentes no SM, isto é, o supervisor toma decisões diferentes de controle, de acordo com a ordem de prioridade dos eventos não-controláveis e o *tick*. Assim, se o evento *b* tivesse prioridade, o evento *d* seria desabilitado no estado *q*3 (Figura 58a) e, caso o *tick* tivesse prioridade em relação ao evento *b* (Figura 58b), o evento *a* seria desabilitado no estado *q*4. Salienta-se ainda que, se os eventos *b* e *tick* estiverem ativos no mesmo ciclo, significa que, a rigor, o evento *b* já ocorreu no ciclo anterior (Sinal I0.0), como pode ser visto no *Scan* 1. Além disso, como o *tick* é sincronizado com o início do ciclo de varredura do CLP, o mais natural é assumir a prioridade dos não-controláveis em relação ao *tick*, conforme foi definido na Seção 5.1.3 (Figura 55).

Figura 58 – Supervisor sujeito ao problema de incapacidade de reconhecer a ordem de eventos



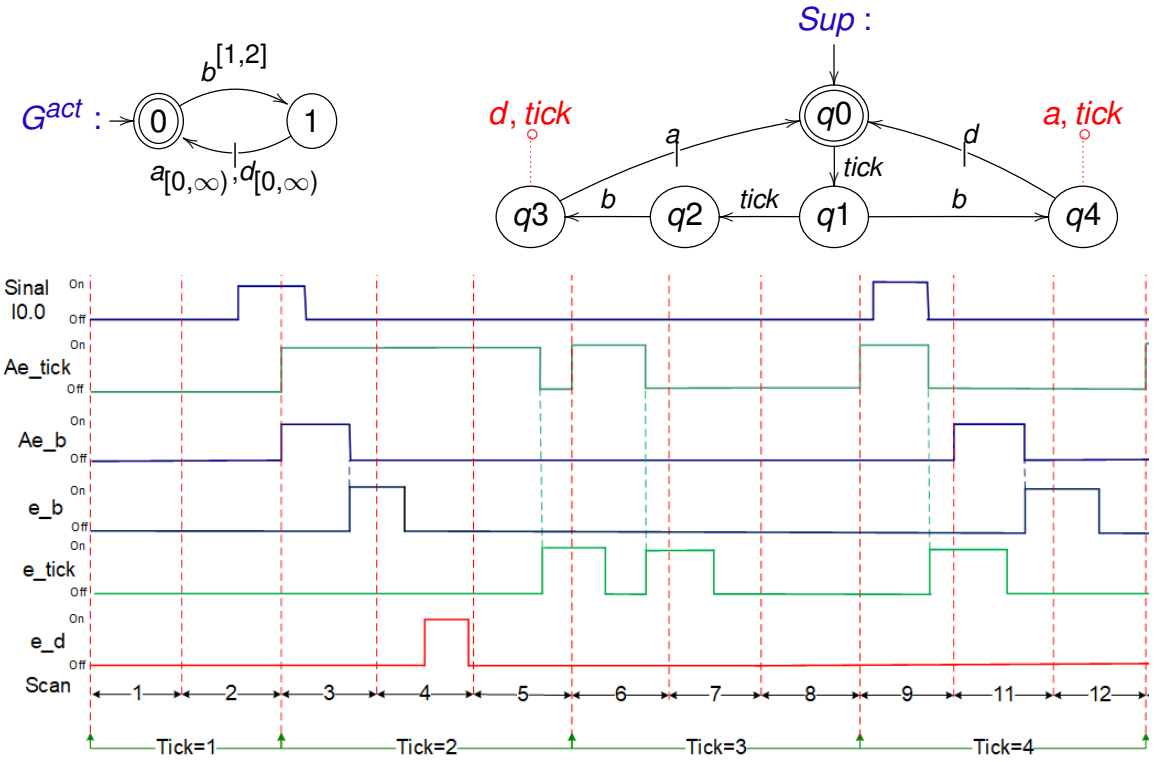
(a) ordem: *b* – *tick* (b) ordem: *tick* – *b*

Fonte: Elaborado pelo autor (2021).

A Figura 59 refere-se ao problema de controlabilidade que poderia ocorrer, ao se assumir que um evento não-controlável tenha prioridade sobre o evento *tick*. Para analisar esse problema, considera-se que o supervisor se encontra no estado *q*1, os eventos *b* e *tick* estejam elegíveis no mesmo ciclo de varredura (*Scan* 3) e que, no estado de destino (*q*4), o supervisor tenha de ser preemptado para forçar um evento

(caminho q1-q4). Nesse caso, um problema de controlabilidade poderia ocorrer, pois no estado *q*4, o evento *d* deve ser forçado e ocorrer antes do *tick* (*Scan* 4), quando a rigor, o *tick* já ocorreu. Esse exemplo ilustra o caso em que o *tick* é atrasado em mais de um ciclo de varredura, uma vez que o evento *b* tenha prioridade sobre o *tick* e que no estado *q*4, o tick seja preemptado. Dessa forma, o evento *b* ocorrerá no *Scan* 3 e o evento *d* no *Scan* 4. Portanto, conforme a ordem definida na Seção 5.1.3, uma maior prioridade também é dada para os eventos forçáveis e, assim, o *tick* é tratado apenas no (*Scan* 5).

Figura 59 – Problema - Prioridade do *tick* em relação ao evento *b* pode causar inconsistência na ação de preempção



Fonte: Elaborado pelo autor (2021).

Deve-se considerar, porém, que a modelagem dos intervalos de tempo do sistema deve ser robusta a esses possíveis atrasos no período do relógio global digital. No máximo, esses atrasos serão de N^*ST, onde ST é o período do ciclo de varredura do CLP e N é o comprimento da maior sequência de transições sem *tick*, de eventos não-controláveis e de eventos forçáveis em S/G.

Assim, conforme a arquitetura temporizada proposta, a Figura 59 ilustra a ordem de ocorrência dos eventos. Se o supervisor estiver no estado *q*1 e acontecerem os eventos *b* e *tick* ao mesmo tempo, tem-se que o evento não-controlável será tratado primeiro, na sequência o evento *d* será forçado e, por fim, o *tick* será processado.

Outro problema que pode ocorrer, está relacionado ao tamanho do período

do relógio (*tick*), como pode ser visto no início do *Scan* 7 da Figura 59, em que uma nova variável *Ae_tick* foi gerada pelo RDG e a variável *e_tick* ainda não foi atualizada nos supervisores. Assim, é necessário que o período do relógio digital, definido na modelagem do SEDT, seja superior ao maior atraso que possa ser gerado pelo processamento de eventos simultâneos no SPT. Para sistemas de manufatura, essa hipótese é bastante razoável, visto que o tempo de varredura de CLPs, da ordem de 50ms, é muito menor que os tempos de operação do processo.

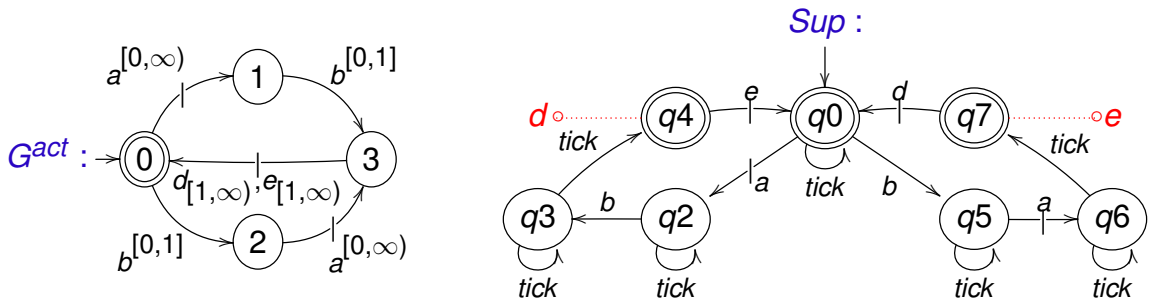
Para que isso não ocorra, o problema de *Interleave Insensitivity* é formalmente definido conforme a propriedade 1 proposta em Fabian e Hellgren (1998). Adaptando essa propriedade para a abordagem temporizada, considera-se que, para que um supervisor possa ser implementado sem que haja problemas, a ação de controle não pode depender da ordem de ocorrência de eventos não-controláveis (Σ_u) e dos eventos controláveis e não-forçáveis ($\Sigma_c \backslash \Sigma_{for}$). Portanto, essa propriedade pode ser reescrita de forma simplificada, como segue: um supervisor $\mathcal{S}$ é insensível ao entrelaçamento com relação a uma planta **G**, se para $\forall s \in \Sigma^*$, $t \in (\Sigma_u \cup (\Sigma_c \backslash \Sigma_{for}))^*$ e $\sigma \in \Sigma_{for}$ tem-se que:

$$\sigma \in \mathcal{S}(s.t) \Rightarrow \forall v \in I(t),\ \sigma \in \mathcal{S}(s.v)$$

, onde $I(t) = \{v \in \Sigma^* : \forall \sigma \in \Sigma, |v|_\sigma = |t|_\sigma\}$ e $|t|_\sigma$ representa o número de ocorrências de σ em t.

A seguir apresenta-se um exemplo para ilustrar essa propriedade. Considerando o autômato da Figura 60, em que $s = \varepsilon \in \Sigma^*$, $t = b,a \in (\Sigma_u \cup (\Sigma_c \backslash \Sigma_{for}))^*$, $\sigma = d \in \Sigma^*_{for}$, tem-se que:

Figura 60 – Problema - Prioridade dos eventos controláveis não-forçáveis em relação aos eventos não-controláveis podem gerar diferentes ações de controle



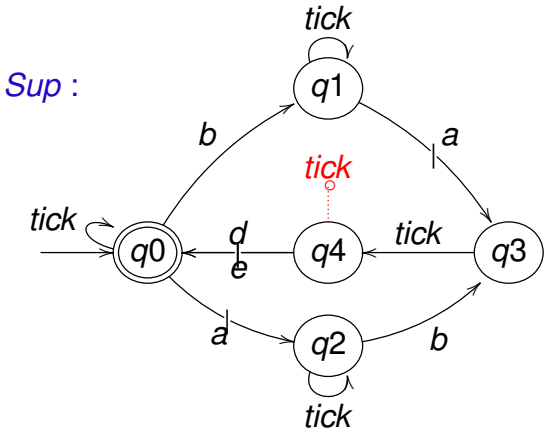
Fonte: Elaborado pelo autor (2021).

$$\mathcal{S}\{a.b\} = \{e \cup \Sigma_u\}, \quad \text{mas } \mathcal{S}\{b.a\} = \{d \cup \Sigma_u\}$$

Conforme a Figura 60, fica evidente que a ação de controle do supervisor, após a sequência *b.a*, é diferente da ação após *a.b*, o que viola a condição anteriormente definida. Dessa forma, o supervisor não é insensível ao entrelaçamento e sua ação de controle não é confiável.

Então, assume-se que os supervisores devem ser insensíveis ao entrelaçamento entre quaisquer eventos não-controláveis e controláveis não-forçáveis. Dessa forma, não existiria o problema de forçar um evento controlável, como é possível observar na Figura 61. Conclui-se, portanto, que pela arquitetura proposta, apenas é necessário verificar que o comportamento em malha fechada (S/G) tenha *Interleaving Insensitivity* entre eventos não-controláveis e controláveis não-forçáveis.

Figura 61 – Supervisor insensível ao entrelaçamento



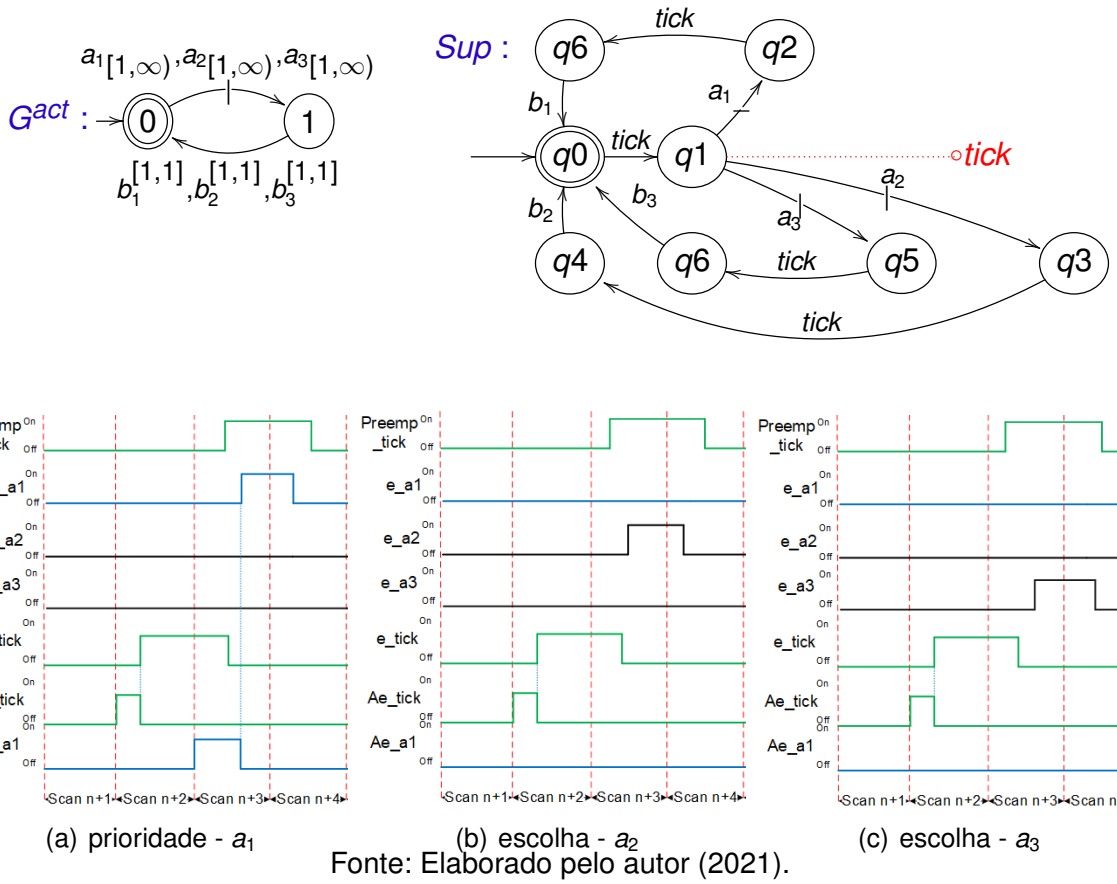
Fonte: Elaborado pelo autor (2021).

5.2.4 Escolha

O problema da escolha ocorre quando o supervisor possuir algum estado em que mais de um evento controlável esteja habilitado, assim, faz-se necessária a escolha de qual evento deve ser o único habilitado. Na arquitetura temporizada proposta, o problema da escolha ocorrerá apenas entre os eventos forçáveis.

A Figura 62 ilustra o caso em que três eventos controláveis, sendo a_1 não-forçável e a_2 e a_3 forçáveis, estão elegíveis no estado $q1$. No início do *Scan* $n+1$, o *tick* é gerado pelo RDG e o sinal da variável *Ae_tick* é enviado para o SPT, que então atualizará os contadores das variáveis ($a1$, $a2$ e $a3$). Assim, o evento *Ae_tick* será resetado e o evento *e_tick*, que será enviado para o SM, será setado. Na sequência, o supervisor é atualizado (estado $q1$) e a preempção do evento *tick* (variável *preempt_tick*) é gerada e enviada para o SPT. Nesse caso, conforme a ordem de implementação constante da Seção 5.1.3 (Figura 55), as seguintes possibilidades podem ocorrer: se o evento a_1 ocorrer primeiro, como ilustrado no *Scan* $n+3$ da Figura 62a, então o evento a_1 será tratado primeiro, pois fisicamente ele já ocorreu, assim, no próximo ciclo, o SM recebe o sinal *e_a1* e atualiza os supervisores (seta estado $q2$); se o evento a_1 não ocorrer, então uma escolha entre os eventos forçáveis a_2 e a_3 deverá ser realizada no SPT. Dessa forma, a Figura 62b ilustra o caso em que o evento a_2 é forçado a ocorrer e a Figura 62c em que o evento a_3 é forçado a ocorrer.

Figura 62 – Autômato sujeito ao problema de escolha



(a) prioridade - a_1 (b) escolha - a_2 (c) escolha - a_3

Fonte: Elaborado pelo autor (2021).

Cabe ressaltar que, um ponto importante a ser considerado é o de como tratar essa escolha entre eventos forçáveis. No caso em que uma prioridade fixa pode causar bloqueio, como visto na Seção 3.1.3, sugere-se deixar a decisão para o sistema no nível mais alto (*SCADA*), priorizando o evento que deve acontecer na fábrica. Outra possibilidade seria, utilizar um valor aleatório (0 e 1), onde todos os eventos, em algum momento irão ocorrer.

5.2.5 Sincronização Inexata

O problema de sincronização inexata pode acontecer quando um supervisor possuir algum estado, no qual a mudança em um sinal de entrada do CLP invalida a ação de controle, devido ao atraso entre a mudança do sinal na entrada e o processamento do evento pelos supervisores, no CLP.

Assim, para que esse problema não ocorra, a linguagem gerada pelo autômato, que modela o supervisor, deve ter a propriedade de ser insensível ao atraso, conforme definido por Balemi Balemi (1992) para uma estrutura não temporizada, conforme consta na Seção 3.1.4, em que o autor trata da relação entre eventos controláveis (comandos) e não-controláveis (respostas).

A estrutura temporizada aqui proposta difere da não temporizada, de duas

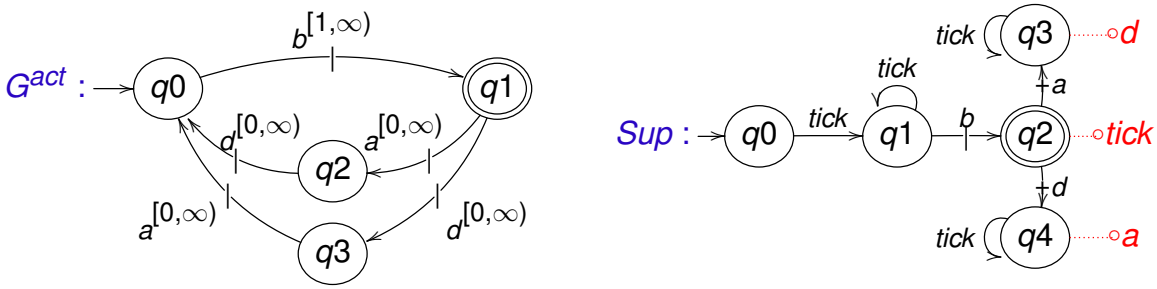
formas: o *tick* é tratado como uma resposta vinda do RDG; e o evento controlável não-forçável é tratado como uma resposta vinda da SO. Assim, o evento *tick* não irá gerar o problema de sincronização, uma vez que, apesar de ser uma resposta, é definido que ele ocorra, exatamente, no início do ciclo de varredura do CLP e, nesse caso, a sincronização é exata, logo, se um evento forçável (comando) e o *tick* (resposta) estiverem ativos no mesmo ciclo, essa perda de sincronia não ocorrerá. Já para os eventos controláveis não-forçáveis esse problema poderá ocorrer, pois esses são tratados como respostas e, dessa forma, devem ser considerados, também, como eventos não-controláveis.

Portanto, um supervisor temporizado será insensível ao atraso como segue: uma linguagem temporizada $K \subset (\Sigma_{act} \cup \{tick\})^*$ é dita ser insensível ao atraso, se para $s \in \overline{K}$, $\sigma_{for} \in \Sigma_{for}$, e $\sigma_r \in \Sigma_u \cup (\Sigma_c \backslash \Sigma_{for})$.

$$quando\ s.\sigma_{for},\ s.\sigma_r \in \overline{K} \Rightarrow s.\sigma_{for}.\sigma_r, s.\sigma_r.\sigma_{for} \in \overline{K} \tag{3}$$

A Figura 63, mostra o autômato de um supervisor sujeito ao fenômeno da sincronização inexata, em que $\Sigma_{for} = \{d\}$, $\Sigma_c \backslash \Sigma_{for} = \{a,b\}$. Supondo que o supervisor esteja em $q2$, os eventos d, a estejam ativos no mesmo ciclo e a ação de controle consista em preemptar o *tick* e forçar o evento d (gerar comando), e que, durante o tempo necessário para realizar essa ação de controle, o evento a tenha ocorrido na planta (durante o atual ciclo de varredura). Assim, como essa informação não será comunicada imediatamente ao supervisor, o evento a terá sido gerado de forma indevida, resultando na quebra de sincronia entre a planta e o controlador implementado no CLP e, no estado $q4$ do supervisor o evento a não poderá ocorrer pois está desabilitado.

Figura 63 – Supervisor temporizado sujeito ao problema de insensibilidade ao atraso

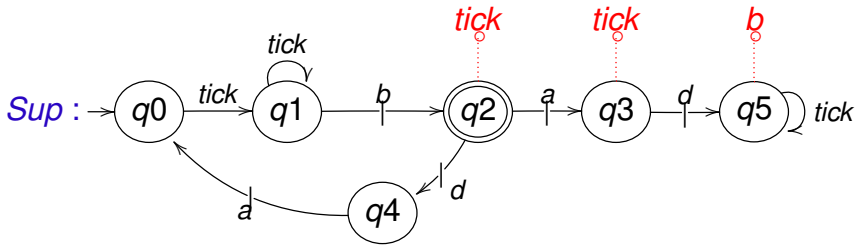


Fonte: Elaborado pelo autor (2021).

Logo, de acordo com essa propriedade, se nenhuma escolha feita pelo supervisor for invalidada pela ocorrência de um evento não-controlável ou controlável não-forçável, pode-se afirmar que o supervisor é insensível ao atraso, uma vez que a

condição diz que, se os eventos σ_{for} e σ_r são elegíveis, então a sequência dos eventos, em qualquer ordem, tem de ser elegível também. O supervisor é insensível ao atraso, visto que as cadeias formadas por $s.\sigma_{for}.\sigma_r$ e $s.\sigma_r.\sigma_{for}$ estão contidas em $\overline{K}$. A Figura 64 ilustra o supervisor insensível ao atraso, para o problema apresentado.

Figura 64 – Supervisor temporizado insensível ao atraso



Fonte: Elaborado pelo autor (2021).

Considerando os autômatos das Figuras 63 e 64 que representam uma linguagem sob supervisão para Σ_{for} = *d*, $\Sigma_c \backslash \Sigma_{for}$ = *a*,*b* e tick, é possível verificar conforme apresentado a seguir.

O autômato da Figura 63 não possui a propriedade de sincronização inexata: sendo s = *tick*.*b*, σ_{for}=d σ_r=a, tem-se que: $s\sigma_{for}$ = *tick*.*b*.*d* $\in \overline{K}$; $s\sigma_r$ = *tick*.*b*.*a* $\in \overline{K}$. Mas, $s.\sigma_r.\sigma_{for}$ = *tick*.*b*.*a*.*d* $\notin \overline{K}$; $s\sigma_{for}\sigma_r$ = *tick*.*b*.*d*.*a* $\notin \overline{K}$. Logo, a linguagem K = {*tick*.*b*.*d*.*a*, *tick*.*b*.*a*.*d*} não é insensível ao atraso.

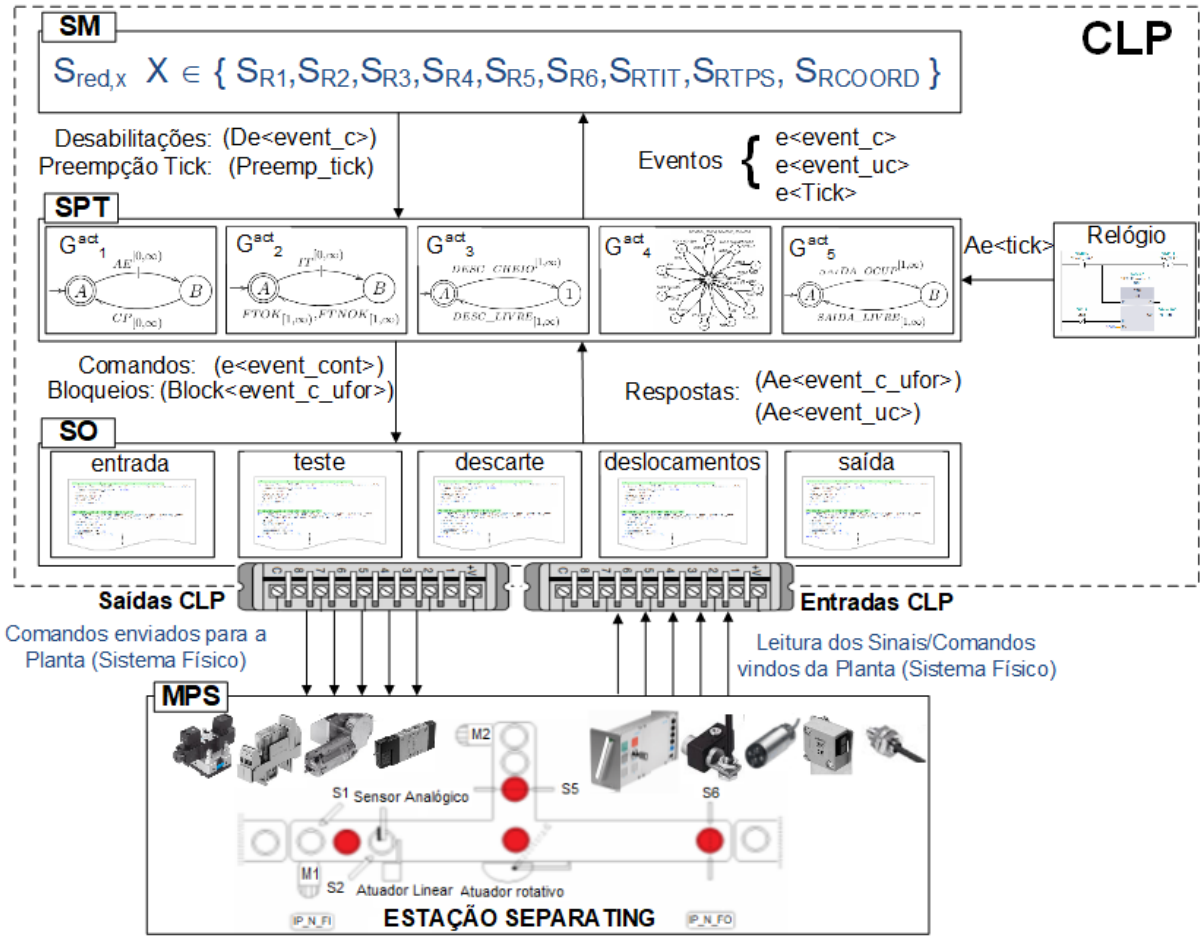
Já o autômato da Figura 64, é possível verificar que o mesmo possui a propriedade de sincronização inexata: sendo s = *tick*.*b*, σ_{for} = *d* e σ_r = *a*, tem-se que: $s\sigma_{for}$ = *tick*.*b*.*d* $\in \overline{K}$; $s\sigma_r$ = *tick*.*b*.*a* $\in \overline{K}$; $s.\sigma_r.\sigma_{for}$ = *tick*.*b*.*a*.*d* $\in \overline{K}$; e $s\sigma_{for}\sigma_r$ = *tick*.*b*.*a*.*d* $\in \overline{K}$. Logo, a linguagem K = {*tick*.*b*.*d*.*a*, *tick*.*b*.*a*.*d*} é insensível ao atraso.

A seguir serão apresentados os resultados da aplicação da arquitetura proposta para implementação da TCST, aos resultados da síntese de controle do sistema real apresentado na Seção 4.2.2.1.

5.3 APLICAÇÃO A ESTAÇÃO DE SEPARAÇÃO

De acordo com a arquitetura proposta na seção anterior, o sistema de controle da estação será implementado em quatro partes: Relógio Digital Global; Supervisores Modulares; Sistemas Produto Temporizado; e Sequências Operacionais, conforme o esquema apresentado na Figura 65.

Figura 65 – Sistema de Controle para a Estação de Separação



Fonte: Elaborado pelo autor (2021).

5.3.1 Implementação do Relógio Digital Global

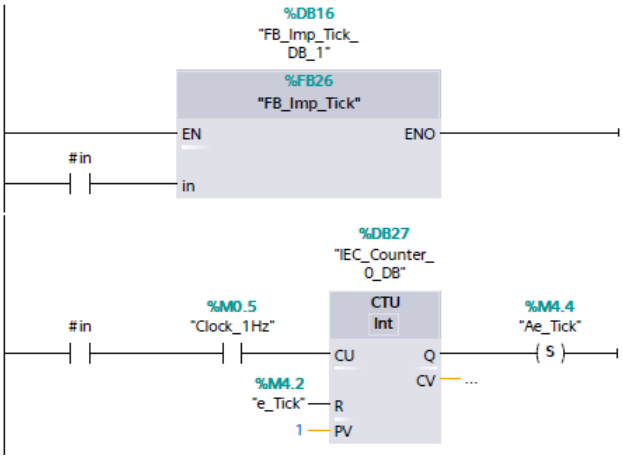
Para a implementação do RDG, propõem-se utilizar a *Organization Block* (“OB1”) que realiza a execução cíclica do programa e, também, permite controlar o tempo de monitoramento do ciclo, isto é, o CLP executará uma sequência de blocos e, assim que finalizá-la, recomeçará um novo ciclo. Além disso, a "OB1"permite controlar o tempo de monitoramento do ciclo, definindo-se um tempo mínimo. Se o ciclo for menor do que o tempo mínimo definido, a CPU aguardará até que esse tempo seja atingido, o que permite trabalhar com um tempo de ciclo de varredura fixo.

Conforme apresentado na Seção 5.1.1, o evento do relógio digital (*tick*) deve ser síncrono com o início do ciclo de varredura, assim, deve-se estimar um valor de tempo mínimo que seja múltiplo do relógio digital global, de forma a permitir a sua sincronização. Para estimar esse tempo, utiliza-se a função *Run Time*, que calcula o tempo de varredura do CLP, em função das linhas de código geradas.

Nessa proposta, a FB que implementa o RDG, é a primeira a ser executada

pela “OB1”. O relógio digital global é implementado através da função *’cycle clock memory’* do CLP da Siemens S7-1200, que nada mais é do que um padrão de bits que muda o seu valor binário periodicamente (Figura 66). A cada *bit* da memória *M*11 é atribuída uma frequência. Para esta implementação utilizou-se o bit *M*0.5 que tem um temporizador com período de duração de 1 segundo (1 *Hz*). Logo, assumindo o *scan cycle time* de 40ms, isso significa que a cada 25 ciclos a variável *Ae_Tick* é gerada.

Figura 66 – Implementação Relógio Digital Global



Fonte: Elaborado pelo autor (2021).

5.3.2 Implementação dos Supervisores Modulares

Os supervisores modulares (e coordenadores) obtidos na Seção 4.2.2.1 são implementados como máquinas de estados concorrentes, de forma que as desabilitações e preempções das respectivas variáveis internas (*De_* < *event_c* > e *Preempt_tick*) sejam sinalizadas ao SPT, de acordo com os estados ativos.

A Figura 67 ilustra a implementação em Texto Estruturado (ST) do supervisor $S_{red,8T}$. Na primeira parte do código, cada estado é representado por uma variável inteira (*Sri _St*) e cada transição seta o estado posterior e reseta o anterior. A variável *e_blk_Sri* garante que apenas uma transição ocorra em cada supervisor, no mesmo ciclo de varredura do CLP (trata o problema do efeito avalanche). Para que se tenha consistência na implementação, essa variável deve sempre ser setada (*e_blk_Sri* := 1) quando uma nova transição ocorrer e resetada (*e_blk_Sri* := 0) após a atualização de todos os supervisores, no final da FB que implementa os supervisores. Observa-se que os auto-laços não são implementados, uma vez que não alteram os estados dos supervisores. Já na segunda parte, estando o sistema produto temporizado e os supervisores atualizados, deve-se definir as ações de controle (desabilitações e a preempção do *tick*) pelos supervisores, a partir do estado em que cada supervisor se encontra.

Figura 67 – Código de Implementação Supervisor $S_{red,8T}$

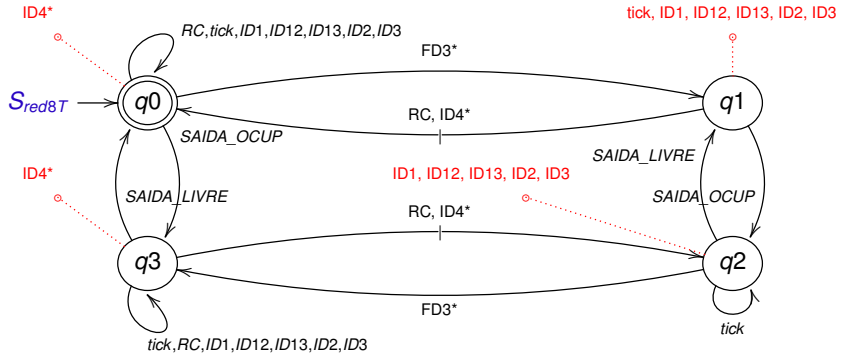
Máquina de Estados

```
(*Uncontrollable events of s_Sr8T*)
"evt_blkSR8T" := 0;
IF ((NOT "evt_blkSR8T") AND "s_Sred8T_St"=0 AND "e_SAIDA_OCUP") THEN
    "evt_blkSR8T" := 1;  "s_Sred8T_St" := 3;
END_IF
IF ((NOT "evt_blkSR8T") AND "s_Sred8T_St"=0 AND ("e_FD13" OR "e_FD134" OR "
    e_FD13" OR "e_FD34" OR "e_FD3ED1" OR "e_FD34ED1")) THEN
    "evt_blkSR8T" := 1;  "s_Sred8T_St" := 1;
END_IF
  ⋮
IF ((NOT "evt_blkSR8T") AND "s_Sred8T_St"=3 AND "e_SAIDA_LIVRE") THEN
    "evt_blkSR8T" := 1;  "s_Sred8T_St" := 0;
END_IF
(*Controllable events of s_Sred8T*)
IF ((NOT "evt_blkSR8T") AND "s_Sred8T_St"=1 AND ("e_ID124" OR "e_ID134" OR "
    e_ID14" OR "e_ID24" OR "e_ID34" OR "e_ID4"))
    "evt_blkSR8T" := 1;  "s_s_Sred5_St" := 0;
END_IF
IF ((NOT "evt_blkSR8T") AND "s_Sred8T_St"=3 AND ("e_ID124" OR "e_ID134" OR "
    e_ID14" OR "e_ID24" OR "e_ID34" OR "e_ID4"))
    "evt_blkSR8T" := 1;  "s_s_Sred5_St" := 2;
END_IF
```

Desabilitações e Preempção

```
(*e_ID14*) De_ID14:=0;
IF (s_Sred8T_St=0 OR s_Sred8T_St=3) THEN
    De_ID14:= 1;
END_IF
(*e_ID124*) De_ID124:=0;
IF (s_Sred8T_St=0 OR s_Sred8T_St=3) THEN
    De_ID124:= 1;
END_IF
  ⋮
(*e_ID3*) De_ID3:=0;
IF (s_Sred8T_St=1 OR s_Sred8T_St=2) THEN
    De_ID3:= 1;
END_IF

(*e_tick*)Preempt_tick := 0;
IF (s_Sred8T_St=1) THEN
    Preempt_tick := 1;
```

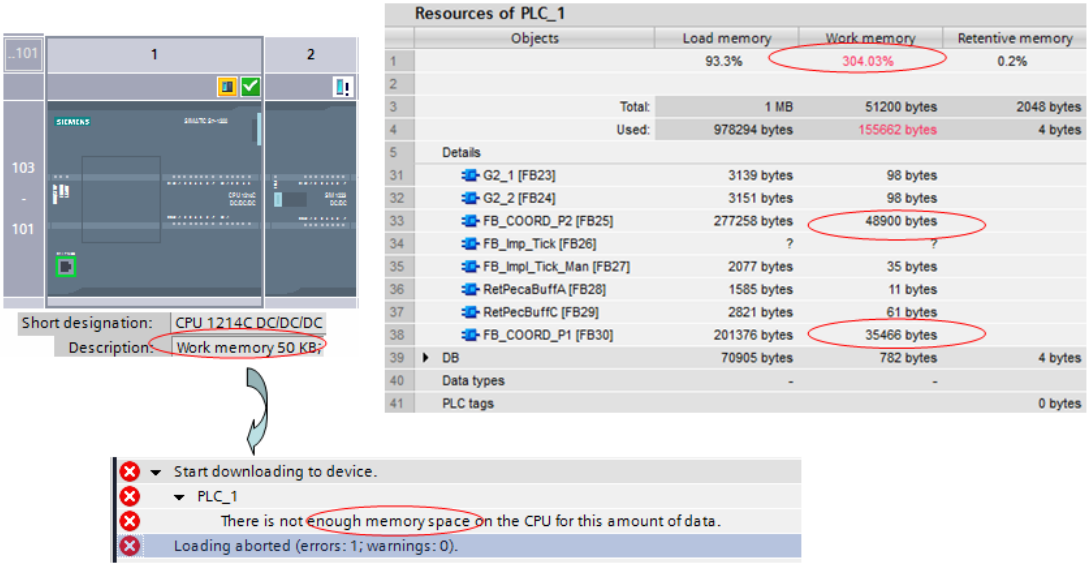


Fonte: Elaborado pelo autor (2021).

Na Seção 4.2.2.1, foi apresentada a solução de um coordenador que resolve o conflito para o modelo temporizado. Porém, a implementação não foi realizada nesta aplicação, pois uma restrição na implementação ocorreu, devido à capacidade de memória de trabalho do CLP ter sido ultrapassada, como pode ser observado na Figura 68. O coordenador obtido possui 1994 transições, gerando um código de aproximadamente 7000 linhas para implementá-lo como uma máquina de estado. Cabe ressaltar, ainda, que atualmente os CLPs industriais, em sua grande maioria, possuem memória suficiente para implementar códigos bem maiores.

Assim, optou-se por implementar apenas a especificação temporizada que prioriza a saída de peças, tornando a implementação da lógica de CMLT possível, pois os supervisores não são conflitantes. Dessa forma, foram implementados 7 supervisores modulares de 2 a 4 estados, resultando em uma lógica implementável na memória do CLP.

Figura 68 – Overload Memória de trabalho CLP



Fonte: Elaborado pelo autor (2021).

5.3.3 Implementação do Sistema Produto Temporizado

Os subsistemas G_1,G_2,G_3, G_4 e G_5 são implementados como máquinas de estados assíncronas, com exceção do evento do relógio digital (*tick*) que é compartilhado em todos os subsistemas. De forma a garantir que duas transições não ocorram de forma seguida no SPT, sem que os supervisores tenham sido atualizados, utiliza-se a variável *evt_blkG* para permitir que apenas uma transição ocorra em cada ciclo de varredura, com exceção do evento *tick*. Para garantir o correto funcionamento do sistema propõem-se uma ordem de prioridades para as transições, conforme descrito na Seção 5.1.3. A Figura 69 exemplifica a implementação do subsistema G_2^{act}.

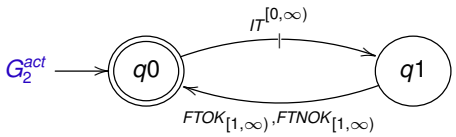
Figura 69 – Código de Implementação SPT - Subsistema G_2

Máquina de Estados - (Σu)

```
(*Uncontrollable events of p_SPT2*)
IF ((NOT "evt_blkG") AND "p_SPT2_St"=1 AND "Ae_FTOK") THEN
    "Ae_FTOK" := 0;
    "e_FTOK" := 1;
    "evt_blkG" := 1;
    "s_SPT2_St" := 0;
END_IF
IF ((NOT "evt_blkG") AND "p_SPT2_St"=1 AND "Ae_FTNOK") THEN
    "Ae_FTNOK" := 0;
    "e_FTNOK" := 1;
    "evt_blkG" := 1;
    "s_SPT2_St" := 0;
END_IF
```

Máquina de Estados - (Σfor)

```
(*Controllable and forcible events of p_SPT2*)
IF ((NOT "evt_blkG") AND "p_SPT2_St"=0 AND (NOT "De_IT")  OR ("Manual_Mode"
    AND "HMI_force_IT" )) THEN
    "e_IT" := 1;
    "evt_blkG" := 1;
    "s_SPT2_St" := 1;
    "HMI_force_IT" := 0;
END_IF
```



Fonte: Elaborado pelo autor (2021).

A partir da lista de eventos habilitados, em nível de sistema produto são identificados quais eventos controláveis são possíveis de serem gerados na planta física. Assim, a cada transição realizada no sistema produto, com um evento controlável que não está desabilitado, é gerado o respectivo evento, promovendo a atualização dos estados do sistema produto. Finalmente, a partir dos eventos que foram gerados no sistema produto, serão atualizados os estados dos supervisores. E a transferência dos eventos gerados no sistema produto para a planta física será ao final do ciclo de varredura, com a escrita das saídas.

Assim, uma vez atualizado o sistema produto e os supervisores com todos os eventos gerados pela planta, a rotina que promove as desabilitações dos eventos controláveis estará em sincronia com o estado da planta física, passando-se, então, à etapa de geração de eventos.

Para a implementação das transições do evento *tick*, propõe-se que as transições associadas a esse evento possam ocorrer em todos os subsistemas da planta, durante o mesmo ciclo de varredura do CLP, desde que não estejam desabilitadas por nenhum supervisor. Para a implementação das transições dos eventos forçados, deve ser verificado se este não está sendo desabilitado por nenhum supervisor e, também,

se o evento *tick* está desabilitado.

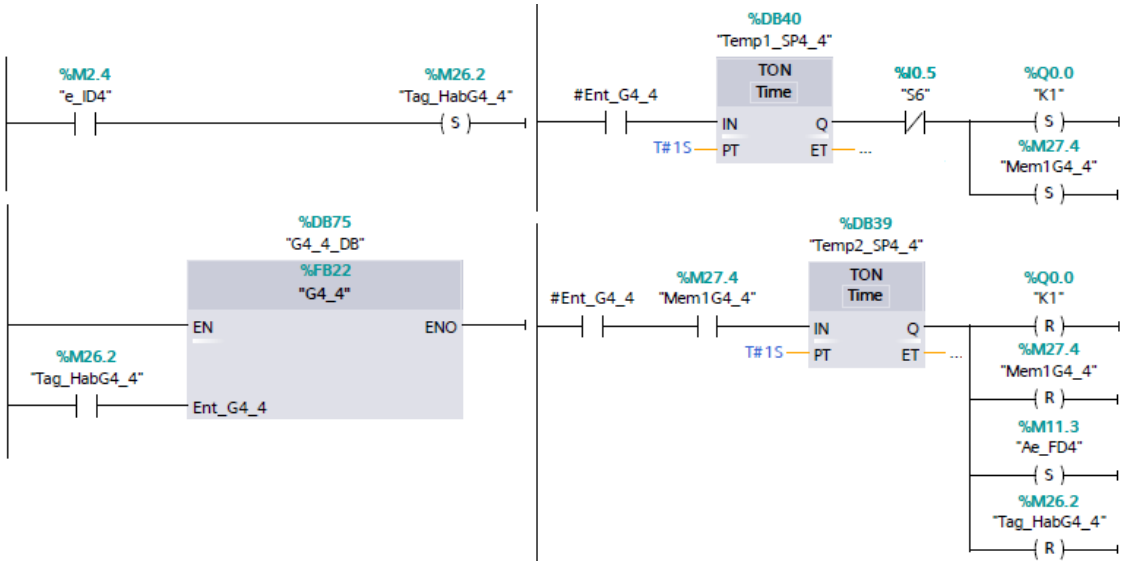
Na primeira implementação realizada nesta estação, o STP foi implementado como GTTs gerando um código de, aproximadamente, 1020 linhas. Na nova proposta, onde o STP é implementado como GTA, obteve-se uma redução do código gerado, ficando com aproximadamente 800 linhas, uma vez que nessa abordagem, não se faz necessário implementar a transição do evento *tick* que, além disso, permite trabalhar com um maior refinamento do período do relógio digital.

Cabe ressaltar que, como apresentado na Seção 5.1.3, o tempo associado aos eventos controláveis são implementados através de contadores. Porém, para a aplicação aqui proposta, a implementação dos cronômetros não foi necessária, uma vez que todos os eventos controláveis podem ocorrer a qualquer instante, isto é, estão associados ao intervalo $[0,\infty)$.

5.3.4 Implementação das Sequências Operacionais

As SOs recebem os comandos gerados pelo SPT (eventos forçáveis) para iniciar a execução da lógica programada de cada ciclo de operação. No final do ciclo ativa-se um sinal (*Ae_event_uc*) que é enviado ao SPT, onde é setada a sua respectiva variável (*e_event_uc*). A Figura 70 mostra a sequência operacional para o deslocamento 4. Quando ocorrer o evento *ID*4 e iniciar a operação da esteira *E*1, a saída *Q*0.0 será ativada e ligará o motor da mesma. Após um tempo mínimo de 1 segundo, quando o sensor *S*6 indicar que a peça não se encontra mais na saída da estação (entrada *I*0.5), o motor é desligado e o evento *Ae_FD*4 é habilitado, indicando o término do ciclo operacional.

Figura 70 – Implementação da Sequência Operacional para o Deslocamento 4



Fonte: Elaborado pelo autor (2021).

5.3.5 Implementação do Gerenciador de Modos de Operação

A metodologia utilizada para a inicialização do sistema é a mesma apresentada por Vieira (2007) e exposta na Seção 3.2. Os modos de operação implementados são: Software Inicialization (SI); *Physical System Initialization* (PSI); *Supervised* (Sup); *remote* (SCADA); *manual* (man); e *emergency* (Emergency). Para a geração automática do código de inicialização, foi utilizada a ferramenta de computação IDES2St (KLINGE, 2007). Na sequência, apresentam-se, de forma resumida, os códigos do modo de inicialização implementado no CLP, como pode ser visto na Figura 71. No modo SI, a variável *Init* será setada sempre na primeira execução do programa, o que faz ativar, automaticamente, a variável *SI_Mode*. O SI é responsável por zerar o valor de todas as variáveis internas do CLP. Após a execução do modo SI, a variável *PSinit* será setada e ficará aguardando o acionamento do botão *Reset* para iniciar o modo PSI. Nesse modo, todas as peças que estiverem na estação serão descartadas, de forma a permitir que o sistema possa reiniciar sem peças e retornar ao seu estado inicial. Concluída a inicialização do sistema, a variável interna *PSready* é ativada e o sistema pode entrar no modo Supervisionado (SUP) ou Manual (Man). O modo de emergência (Emergency) é ativado quando pressionando o botão *Stop*. Nesse modo, todas as atividades são suspensas imediatamente e, na sequência, o sistema retorna para o modo SI, realizado novamente todo o processo de inicialização do sistema.

5.3.6 SCADA

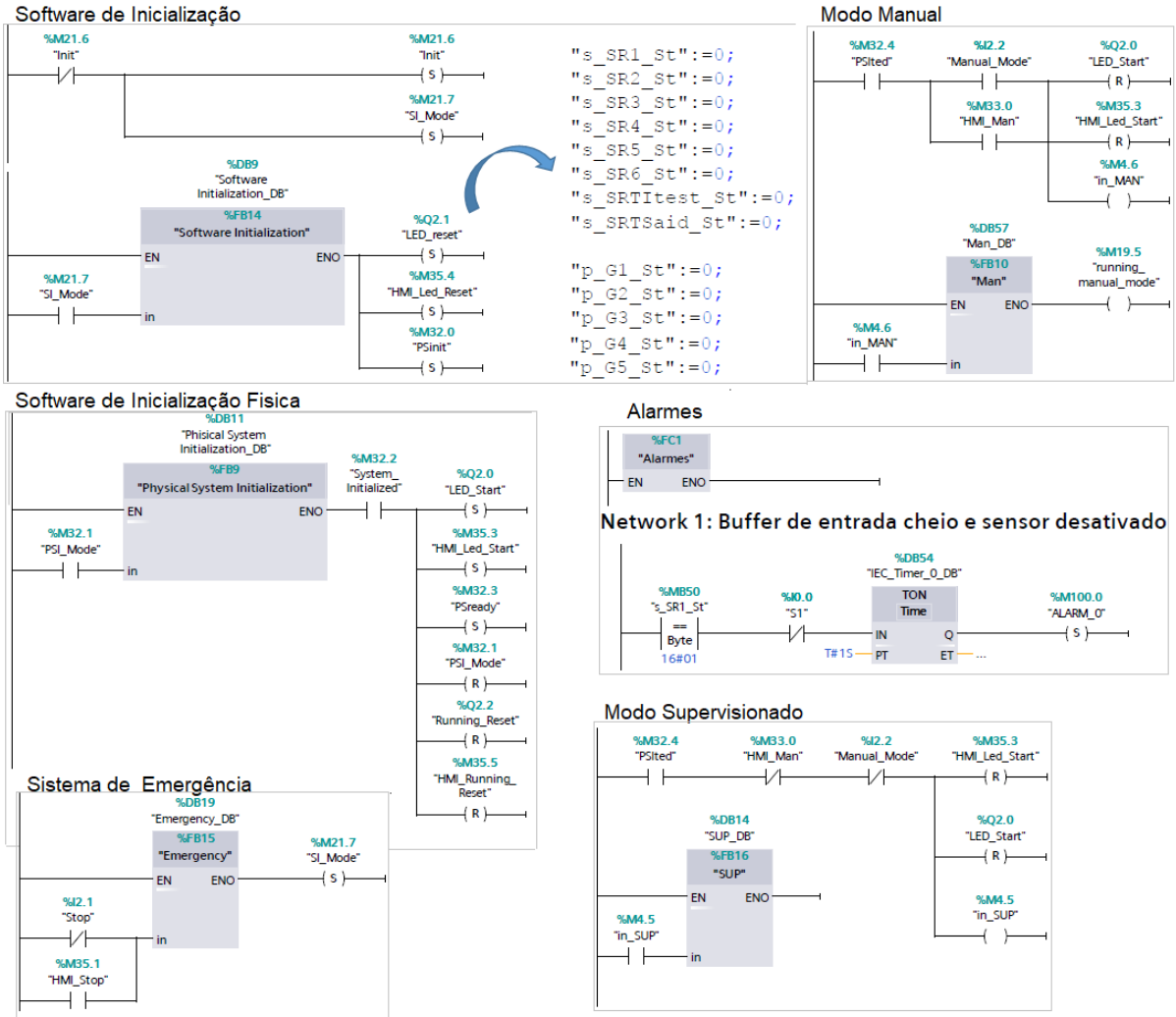
Para a construção do sinótico, foram analisados os estados necessários para visualizar, corretamente, o funcionamento da planta. Na sequência, foram criadas as variáveis como *tags* no *software* SCADA, de forma a serem inseridas na tela do sinótico. Logo, é possível monitorar os estados dos supervisores locais e as principais variáveis do sistema, como por exemplo: o resultado do teste; *buffer* do descarte cheio/livre; e saída ocupada/livre. Assim, é possível, também, comandar remotamente o sistema, por meio dos comandos iniciar, parar, reiniciar e forçar eventos no modo manual. A Figura 72 mostra a tela principal do sistema supervisório executando a operação de inicialização do sistema.

5.4 CONCLUSÃO DO CAPÍTULO

A principal contribuição deste capítulo consiste em uma proposta que estende a arquitetura de Queiroz e Cury (2002), que seja viável para a implementação dos CMLTs em CLPs, com a introdução do evento *tick* para o tratamento do relógio digital global e, também, o conjunto de eventos forçáveis.

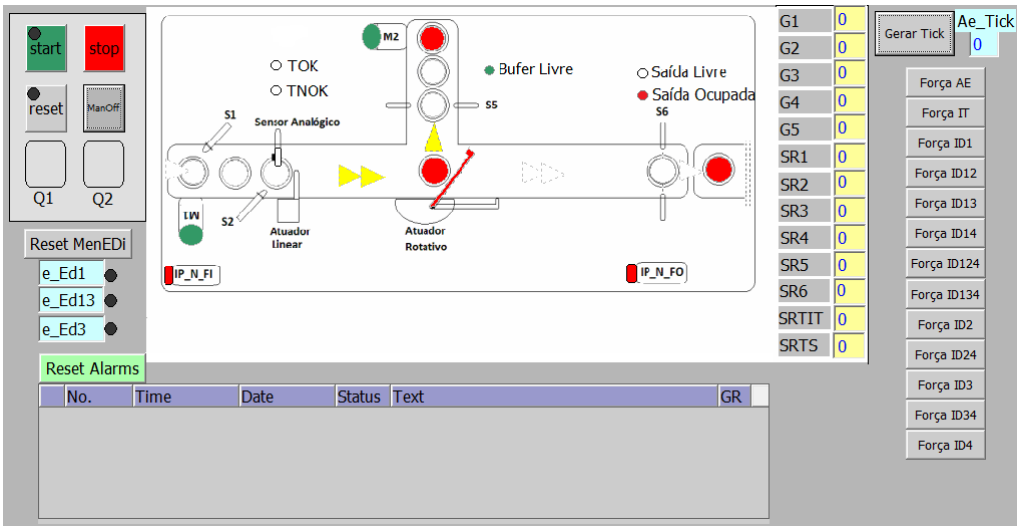
A partir de uma revisão dos principais problemas de implementação definidos por Fabian e Hellgren (1998), foi possível analisar como esses se comportaram na ar-

Figura 71 – Código CLP - Sistema de Inicialização



Fonte: Elaborado pelo autor (2021).

Figura 72 – Sinótico Geral - Estação Separating



Fonte: Elaborado pelo autor (2021).

quitetura temporizada e, assim, obtendo algumas soluções para alguns dos problemas apresentados.

Outra contribuição é a implementação das plantas locais como GTAs, resultando na economia do código gerado para a implementação dos problemas. Uma vez que mesmo refinando o período do relógio digital (*tick*), o número de estados dos modelos da planta não aumenta.

A implementação dos supervisores temporizados no sistema real se comportou, de uma maneira geral, como esperado e, portanto, a arquitetura proposta pode ser utilizada como uma ideia inicial para implementação da lógica de CMLT. Nesta aplicação foi possível, também, verificar o forçamento de eventos através do sistema SCADA. Como o código gerado com a solução para o coordenador ultrapassou a memória da CLP, optou-se por implementar apenas a especificação temporizada, que prioriza a saída de peças, tornando a implementação da lógica de CMLT possível e implementável.

Portanto, a arquitetura proposta atende às características dos eventos apresentados, e é suficientemente geral para lidar com uma ampla classe de problemas reais.

6 MODELAGEM DE SISTEMAS DE MANUFATURA SOB INCERTEZAS DE PROCESSAMENTO E RESTRIÇÕES DE TEMPO

Em sistemas de manufatura, os tempos de processamento podem ser incertos e os eventos podem ser associados a intervalos contínuos. Além disso, os sistemas de manufatura estão sujeitos a eventos não-controláveis, sob os quais não se pode decidir, e as restrições de tempo estabelecem relacionamentos entre esses eventos. Nesse caso, uma solução representada por uma sequência fixa, de ações de controle, como aquelas obtidas pelos métodos clássicos de escalonamento de processos, torna-se conservadora e acaba não sendo eficiente.

Neste capítulo, uma abordagem reativa, baseada na TCS para SEDT, como a apresentada anteriormente, é utilizada para resolver uma classe de problemas de controle de sistemas de manufatura, sob incertezas de processamento e restrições temporais. Para isso, são propostos modelos para alguns componentes e especificações tradicionalmente encontrados nos sistemas de manufatura. Ainda, propõe-se um método para traduzir os intervalos de tempo contínuos, normalmente associados aos processos e especificações temporais de um sistema, em intervalos discretos associados a um relógio digital compatível com a TCS. É apresenta, também, uma análise da influência da escolha de valores para a unidade temporal *tick*, no desempenho da solução obtida e na complexidade dos modelos e, consequentemente, do processo de síntese.

6.1 PLANTA

Nesta seção, apresenta-se como modelar a planta para um sistema de manufatura sob incertezas de processamento, como um modelo de SEDT. Apresenta-se, ainda, a discretização dos intervalos contínuos, bem como sua aplicação em um pequeno sistema de manufatura.

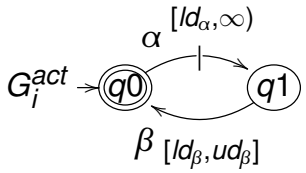
6.1.1 Modelo SEDT

Embora uma máquina em um sistema de manufatura geralmente tenha um comportamento composto por várias operações e sub-operações, seu modelo considerado por uma abordagem baseada na TCS é geralmente uma abstração em que apenas os eventos relevantes para expressar as propriedades desejadas da máquina devem ser mantidos. Neste estudo assume-se que um modelo com dois eventos representando o início e o término da operação é suficiente para resolver o problema de coordenação entre máquinas. Este modelo pode, sempre que necessário, ser complementado com outros eventos que representem quebras, falhas ou múltiplos tipos de operações. Por outro lado, o comportamento detalhado da máquina é sempre devidamente levado em

consideração na implementação como apresentado no Capítulo 5.

A Figura 73 apresenta o modelo de uma máquina, composta por dois estados ($q0$ e $q1$) e dois eventos (α e β) representando respectivamente, o início e fim da operação da máquina, e marcados com seus intervalos de tempo. α é considerado um evento remoto (Σ_{rem}), marcado com um tempo mínimo ld_α, representando um atraso necessário no início de operação do equipamento, e um tempo máximo (∞) significando que sua ocorrência pode ser mantida indefinidamente. Por outro lado, β é considerado um evento esperado (Σ_{esp}), marcado por um tempo mínimo ld_β e um tempo máximo ud_β, que modelam os limites no tempo de processamento da operação incerta.

Figura 73 – Autômato - Representação Modelo Planta



Fonte: Elaborado pelo autor (2021).

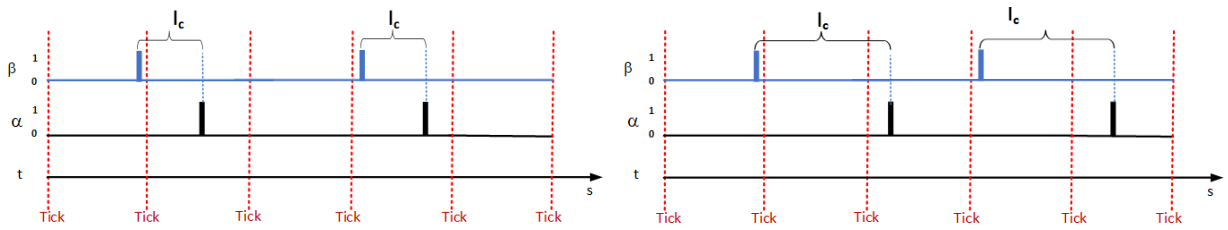
Como visto na Seção 4.1.1 e representado na Figura 73, o evento controlável α, com intervalo discreto $[ld_\alpha;\infty)$, onde $ld_\alpha \in \mathbb{N}^+$, é um evento remoto que pode ocorrer após ld_α *ticks* no estado $q0$ e pode ser adiado indefinidamente pelo supervisor, e o evento não-controlável β, uma vez ativado, está limitado a ocorrer dentro de um horizonte de tempo finito $[ld_\beta;ud_\beta]$, com ld_β, $ud_\beta \in \mathbb{N}^+$.

6.1.2 Discretização de intervalos contínuos

Os eventos que podem ocorrer em um sistema de manufatura, normalmente, estão associados a intervalos contínuos, porém, esses intervalos que são associados aos modelos de SEDs de BW, precisam estar discretizados e expressos, como múltiplos do período do relógio digital global (*tick*). Assim, para obter o intervalo discreto (minimamente conservador), associado aos eventos α e β, como apresentado no modelo da Figura 73, parte-se do tempo de processamento de um equipamento, onde o evento α está associado a um intervalo de tempo contínuo $I_{c_\alpha} = [lc_\alpha;\infty)$ e outro evento β a um intervalo $I_{c_\beta} = [lc_\beta;uc_\beta]$, em que lc_α, lc_β,$uc_\beta \in \mathbb{R}^+$.

Para a discretização do tempo associado ao evento remoto α, o que se deseja obter é o menor número de *ticks* possíveis de ocorrer antes da ocorrência de α, considerando que o evento α não está sincronizado com o relógio digital. O pior caso se dá quando a entrada que ativa o evento α ocorrer imediatamente após o *tick*, gerando o menor resultado, conforme observado na Figura 74.

Figura 74 – Planta - Número de ocorrências de *tick* para Σ_{rem}



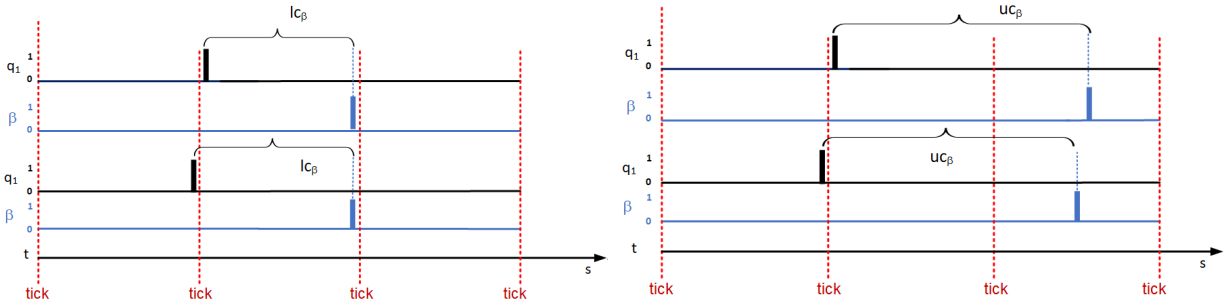
(a) $lc_\alpha \leq \delta$ (b) $lc_\alpha \geq \delta$

Fonte: Elaborado pelo autor (2021).

Assim, o valor discreto associado ao evento remoto, ld_σ será dado pelo menor valor inteiro que satisfaça lc_σ/δ, ou seja $ld_\sigma = \lfloor lc_\sigma/\delta \rfloor$, onde δ corresponde ao período do relógio digital (*tick*).

Quanto à discretização do evento esperado, ld_β é determinado pelo número mínimo e ud_β pelo número máximo de *ticks*, que podem ocorrer dentro dos tempos contínuos lc_σ e uc_σ, respectivamente. O pior caso para o tempo mínimo (lc_β) acontece quando a entrada que ativa o evento β ocorrer imediatamente após o *tick*, e para o tempo máximo (uc_β), quando o evento β ocorrer na iminência do *tick*, como pode ser visto na Figura 75

Figura 75 – Planta - Número de ocorrências de *tick* para Σ_{esp}



(a) lc_β (b) uc_β

Fonte: Elaborado pelo autor (2021).

Nesse caso, o que se deseja obter é o menor intervalo discreto (*ld*) para um dado evento σ, de forma minimamente conservadora e que não viole o intervalo contínuo, originalmente dado. Portanto, obter o intervalo ld_σ, corresponde a calcular o maior valor de ld_σ e o menor valor de ud_σ, tal que o intervalo correspondente *lcr*, contenha o intervalo contínuo original *lc*. Para isso ld_σ e ud_σ devem satisfazer:

$$\begin{cases} 1.\ lc_\sigma \geq \delta^* ld_\sigma > lc_\sigma - \delta \\ 2.\ uc_\sigma \leq \delta^* ud_\sigma < uc_\sigma + \delta \end{cases} \tag{4}$$

Qualquer intervalo discreto em que o evento σ ocorra dentro do intervalo original seria uma abstração conservadora, porém, não necessariamente será mínima. Assim, o que se deseja é que o modelo discreto não desrespeite os intervalos de tempos obtidos no modelo contínuo. Dessa forma, a melhor aproximação de I_d é tal que: i) o evento σ nunca ocorra antes de $ld_\sigma - 1$ *ticks*; e ii) o evento σ sempre ocorra antes de $ud_\sigma + 1$ *ticks*, pois, uma vez que ud_σ ocorra, o evento σ ainda pode acontecer antes do próximo *tick*.

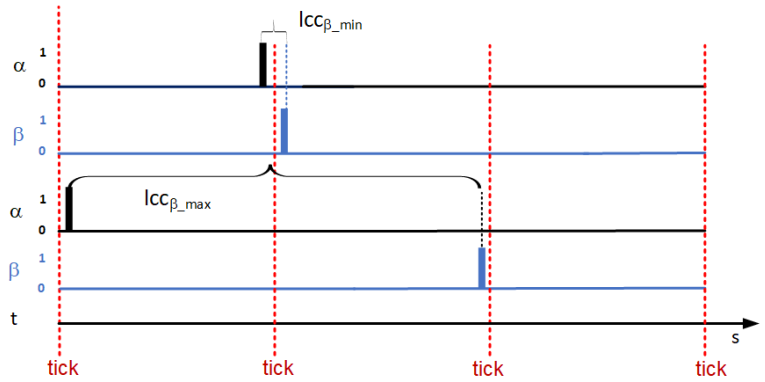
Portanto:

$$ld_\sigma = \lfloor lc_\sigma/\delta \rfloor \ \ e \ \ ud_\sigma = \lceil uc_\sigma/\delta \rceil \tag{5}$$

Onde, ld_σ é o maior inteiro menor ou igual a lc_σ/δ e ud_σ é o menor inteiro maior ou igual a uc_σ/δ.

Ao traduzir um intervalo discreto de volta para o tempo contínuo, isso representa obter um intervalo correspondente igual a $[lcc_\sigma, ucc_\sigma)$, que deve ser maior do que o original dado ($lcc_\sigma \supset lc_\sigma$), e que seja igual a $(\delta^*(ld_\sigma - 1); \delta^*(ud_\sigma + 1))$.

Figura 76 – Planta - Mínimo (lcc_σ) e máximo (ucc_σ) intervalo de tempo



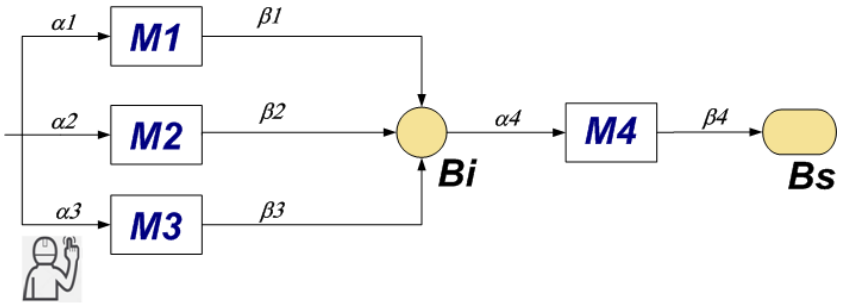
Fonte: Elaborado pelo autor (2021).

6.1.3 Exemplo de aplicação: Pequeno Sistema de Manufatura

Traz-se como ilustração um sistema de manufatura composto por quatro máquinas M_1, M_2, M_3 e M_4, em que M_1 produz peças do tipo A, M_2 peças do tipo B, M_3 do tipo C e M_4 é responsável por retirar as peças do *buffer* intermediário B_i e realizar um processo de pintura antes de colocá-las no *buffer* de saída B_s (Figura 77).

Para exemplificar o método de discretização proposto, assume-se aqui um período de δ = 5s para o relógio digital global, uma vez que é um valor que leva à obtenção de autômatos com pequeno número de estados.

Figura 77 – Representação do Sistema de Manufatura Composto por M_1, M_2, M_3, *Buffer* intermediário (B_i) e M_4



Fonte: Elaborado pelo autor (2021).

Modelos das Plantas: para os eventos propostos na Figura 77, dados por: $a_i \in \Sigma_{rem}$ - início de operação da máquina M_i; e $b_i \in \Sigma_{esp}$ - fim de operação da máquina M_i; em que i = {1,2,3,4}, apresenta-se a Tabela 5 com os intervalos de tempo contínuos associados as máquinas, bem como, os limites de tempo mínimo (lc_σ) e máximo (uc_σ), atribuídos a cada operação.

Tabela 5 – Intervalos de Tempo Contínuos Associados aos Eventos de Σ_{act}.

Máquinas	Operações	Intervalo Contínuo $[lc_\sigma; uc_\sigma]$
M_1	α_1 β_1	$[0s; \infty)$ $[11{,}3s; 14{,}5s]$
M_2	α_2 β_2	$[0s; \infty)$ $17{,}4s; 20{,}1s]$
M_3	α_3 β_3	$[5{,}3s; \infty)$ $[13{,}1s; 19{,}5s]$
M_4	α_4 β_4	$[0s; \infty)$ $[5{,}2s; 6{,}4s]$

Fonte: Elaborado pelo autor (2021).

Uma vez obtido os intervalos contínuos, é possível discretizá-los. Dessa forma, para exemplificar a metodologia proposta, apresenta-se o cálculo do intervalo discreto, associado ao evento esperado de M1 (β_1). Seja o evento esperado, dado pelo intervalo de tempo contínuo $lc_{\beta_1} = [lc_{\beta_1}; uc_{\beta_1}] = [11{,}3s; 14{,}5s]$, o intervalo discreto associado a β_1 é determinado por $[ld_\sigma = \lfloor lc_\sigma/\delta \rfloor; ud_\sigma = \lceil (uc_\sigma/\delta) \rceil] = [\lfloor 11{,}3/5 \rfloor; \lceil (14{,}5/5) \rceil] = [2, 3]$. Nesse caso, pode-se observar que o modelo discreto é conservador, uma vez que, o intervalo contínuo resultante obtido [5s ; 20s), contém o intervalo contínuo fornecido originalmente.

Conforme os intervalos de tempo contínuos apresentados anteriormente, a Tabela 6 exibe os intervalos discretos associados aos eventos não-controláveis, obtidos mediante a aplicação do método de discretização proposto.

Tabela 6 – Intervalos Discretos Associados aos Eventos Esperados (Σ_{esp})

Operações	Intervalo Discreto $[ld_\sigma; ud_\sigma]$
β_1	[2 ; 3]
β_2	[3 ; 5]
β_3	[2 ; 4]
β_4	[1 ; 2]

Fonte: Elaborado pelo autor (2021).

Para a discretização dos eventos remotos, apresenta-se o cálculo do intervalo discreto, associado ao evento (β_1) de M3. Seja o intervalo de tempo contínuo dado pelo intervalo $Ic_{\alpha_3} = [Ic_{\alpha_3}; \infty)] = [5{,}3s; \infty)$, o intervalo discreto associado a α_3 = é dado por $[ld_\sigma = \lfloor 1{,}06 \rfloor; ud_\sigma = \infty)$, logo $I_d = [1, \infty)$. A Tabela 7 apresenta os intervalos de tempo discretos, associados aos eventos controláveis.

Tabela 7 – Intervalos Discretos Associados aos Eventos Remotos (Σ_{rem})

Operações	Intervalo Discreto $[ld_\sigma; ud_\sigma]$
α_1	[0 ; ∞)
α_2	[0 ; ∞)
α_3	[1 ; ∞)
α_4	[0 ; ∞)

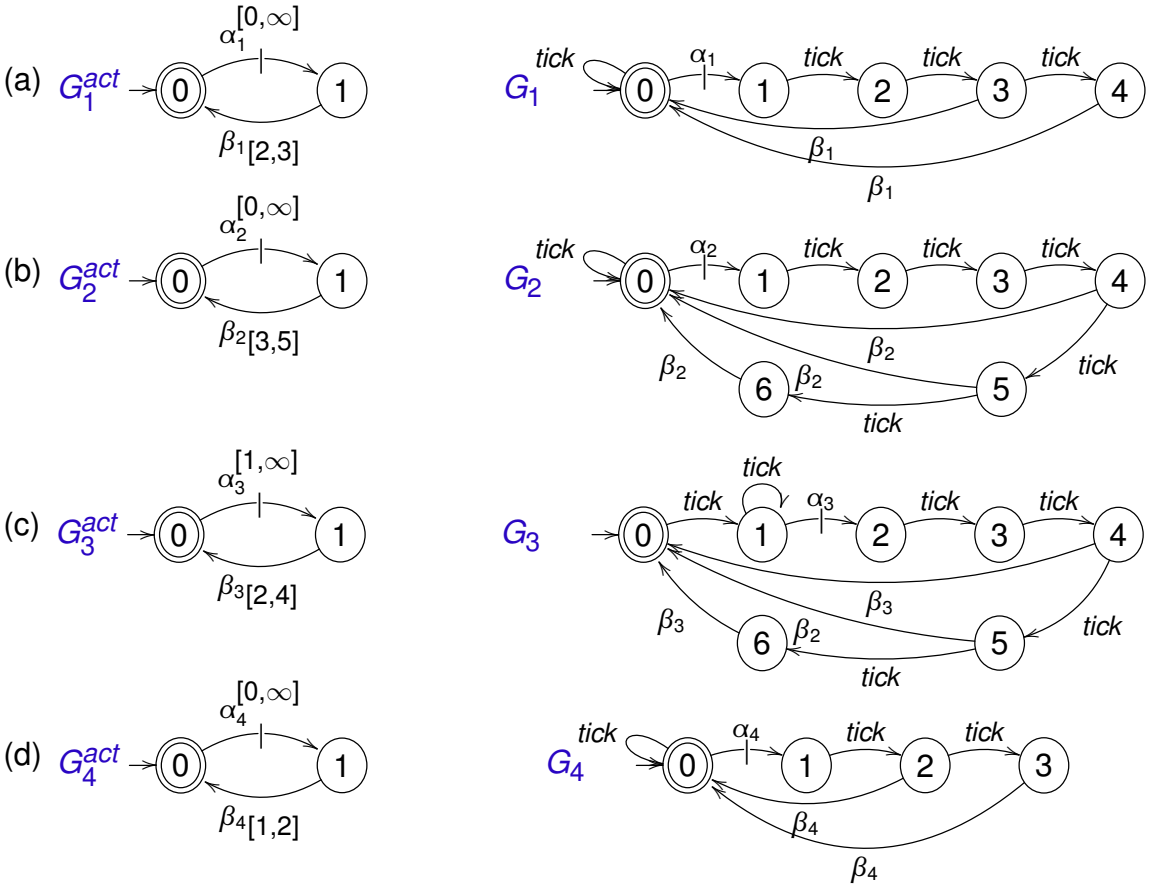
Fonte: Elaborado pelo autor (2021).

Uma vez realizada a discretização desses intervalos contínuos, é possível obter os modelos das plantas. Logo, o sistema a ser controlado é modelado por quatro subsistemas concorrentes, com o *tick* sendo o único evento compartilhado. A seguir são apresentados os autômatos das plantas para o exemplo proposto. Os respectivos *GTAs* $G_1^{act}, G_2^{act}, G_3^{act}, G_4^{act}$ e *GTTs* G_1, G_2, G_3 e G_4 são mostrados na Figura 78.

6.2 ESPECIFICAÇÕES

Nesta seção, apresenta-se como modelar as especificações para um sistema de manufatura sobre restrições de tempo, como um modelo de SEDT. Apresenta-se, ainda, a discretização dos tempos contínuos, bem como sua aplicação em um pequeno sistema de manufatura.

Figura 78 – Modelos SEDT para M1 (a); M2 (b); M3 (c) e M4 (d) com os correspondentes GTAs na esquerda e GTTs na direita.



Fonte: Elaborado pelo autor (2021).

6.2.1 Modelo SEDT

Em geral, as especificações temporais limitam o funcionamento do sistema, evitando que estados indesejados sejam alcançados, por exemplo, uma colisão entre um robô e um dispositivo pneumático de posicionamento de peças. Para ilustrar as especificações temporais, definem-se três modelos que representam o comportamento restritivo, comumente encontrado em sistemas de manufatura, quais sejam: de tempo máximo de espera (*Deadline*); de tempo mínimo de espera (*delay*); e de tempo mínimo e máximo de espera.

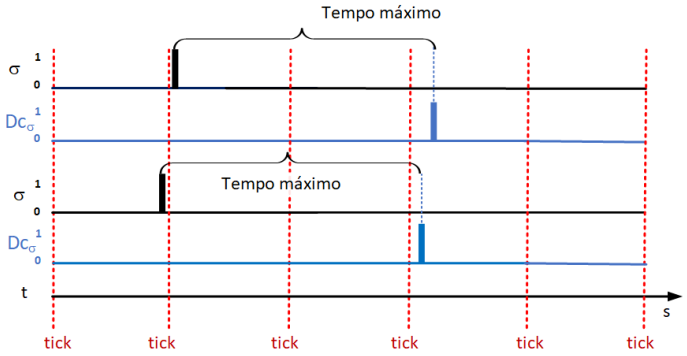
Tempo máximo para ocorrência *(Deadline)*: a especificação de *deadline* é uma restrição que limita a ocorrência de um evento σ, a partir de um determinado estado e a contar do momento em que o evento esteja elegível.

Cita-se como exemplo, o *deadline* para um dado sistema de manufatura em que uma peça com um certo aditivo anticorrosivo poderá aguardar em um buffer de espera, antes de ser processada, ou também, o tempo máximo que uma determinada peça

sujeita a duas ou mais operações, seja por temperatura, pintura, ou outras condições, possa aguardar. É preciso respeitar esses tempos para garantir a qualidade final da peça.

Para garantir que nenhum tempo máximo seja perdido, deseja-se obter o maior número de *ticks* entre dois eventos, que assegure que o *deadline* (Dc_σ) nunca seja maior do que o tempo máximo especificado. A Figura 79 ilustra o número mínimo e máximo de *ticks* possíveis de ocorrer, em que o pior caso acontece quando o evento σ ocorrer logo após o *tick*.

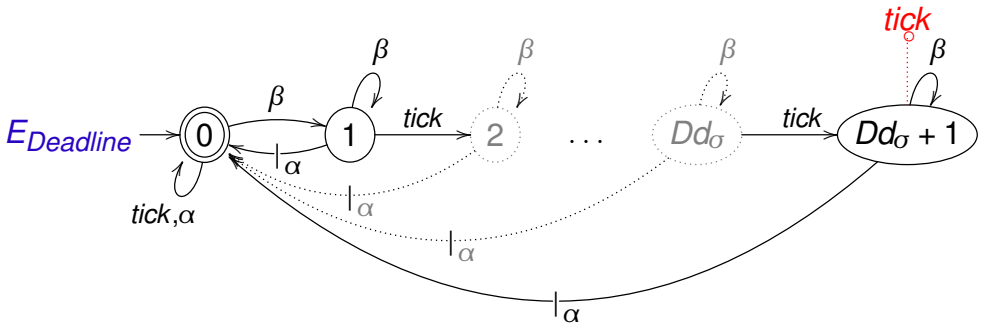
Figura 79 – Especificação de Tempo Máximo - Número mínimo e máximo de *ticks*



Fonte: Elaborado pelo autor (2021).

A Figura 80 define o autômato que representa o modelo geral do comportamento para uma especificação de tempo máximo para a ocorrência de um evento α, a partir da ocorrência de um evento β. Para garantir que o *deadline* seja respeitado, deve-se diminuir um *tick* do número mínimo.

Figura 80 – Especificação de tempo máximo Dd_σ para ocorrência de α a partir de β



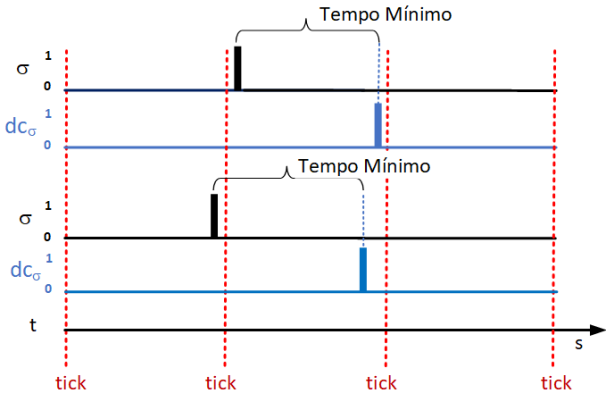
Fonte: Elaborado pelo autor (2021).

Tempo mínimo de espera *(Delay)*: define-se como *delay* (d), o tempo mínimo que um determinado evento σ habilitado deverá aguardar até a sua ocorrência. Essa especificação estipula um limite mínimo, mas não garante um limite máximo, o que pode causar situações indesejáveis aos sistemas. Cita-se, como exemplo, o tempo mí-

nimo que uma peça deve aguardar em um *buffer* intermediário para o seu resfriamento, antes de seguir para o processamento seguinte.

Para garantir que a especificação seja respeitada, deve-se determinar o menor número de *ticks* que pode ocorrer, de modo a garantir que se tenha um valor maior que o tempo mínimo especificado. Pode-se observar na Figura 81, que o pior caso (para $dc_\beta < \delta$), ocorrerá quando o evento σ na iminência do *tick* e, neste caso, o tempo mínimo não será respeitado. Já para garantir que seja respeitado, deve-se somar um *tick* a mais.

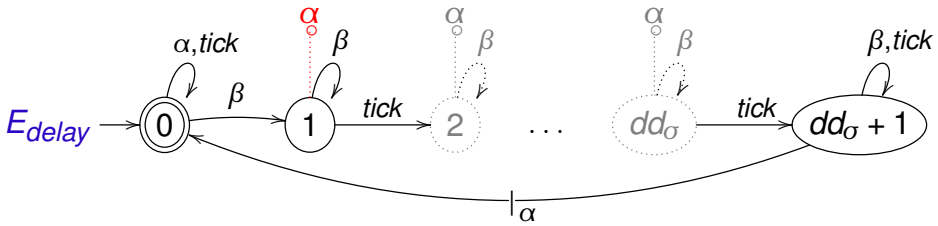
Figura 81 – Especificação de Tempo Mínimo - Número mínimo e máximo de *ticks*



Fonte: Elaborado pelo autor (2021).

Então, na Figura 82 é definido o autômato de um modelo geral de uma especificação de *delay*, para a ocorrência de um evento α, a partir da ocorrência de um evento β.

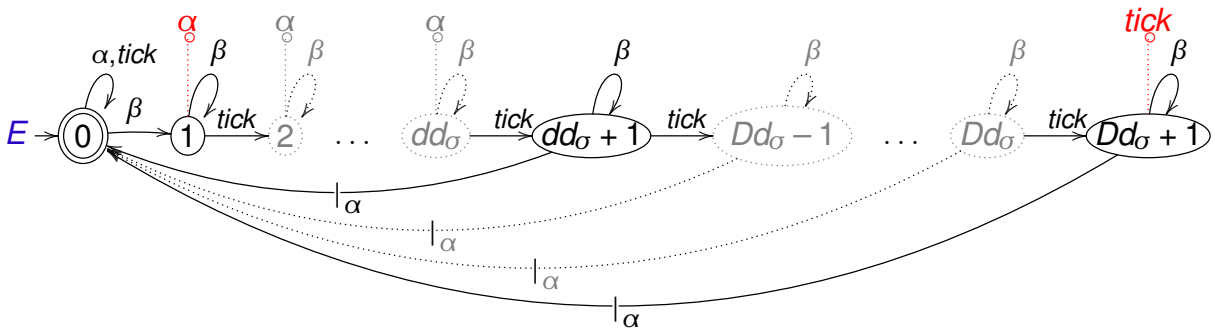
Figura 82 – Especificação de tempo mínimo de espera dd_σ, para a ocorrência de α a partir de β



Fonte: Elaborado pelo autor (2021).

Intervalo de tempo mínimo e máximo: a especificação de tempo mínimo e máximo é dada pela composição das especificações de tempo mínimo de espera e de tempo máximo para ocorrência de um evento. Assim, o intervalo é dado por $[dc_\beta; Dc_\beta]$. A Figura 83 ilustra o modelo geral para uma especificação de tempo mínimo e máximo para a ocorrência de um evento α, a partir da ocorrência de um evento β.

Figura 83 – Especificação de intervalo de tempo mínimo dd_σ e máximo Dd_σ para a ocorrência de α a partir de β.



Fonte: Elaborado pelo autor (2021).

6.2.2 Discretização dos intervalos contínuos

Para a discretização das especificações associadas a valores de tempos contínuos, assim como nos modelos das plantas, é necessário manter uma forma, minimamente, conservadora, em termos de *ticks* de relógio digital. Dessa maneira, ser conservador dependerá do tipo da especificação a ser aplicada.

Para a discretização do *deadline*, o evento σ nunca poderá acontecer depois do tempo máximo especificado. Deve ser conservadora, no sentido de garantir que o intervalo discreto obtido, seja mais restritivo do que o contínuo estabelecido ($Icr \subseteq Ic$), e a discretização de *delay* deve garantir que o intervalo discreto obtido, seja menos restritivo que o contínuo estabelecido ($Icr \supseteq Ic$).

Assim, a discretização do *deadline* é dada como segue: seja $Dc_\sigma \in \mathbb{R}^+$ e o tempo máximo para ocorrência de um evento σ e δ o período do relógio digital, deseja-se obter o maior valor do *deadline* discreto $Dd_\sigma \in \mathbb{N}^+$, que multiplicado pelo período do relógio digital, seja menor ou igual a Dc_σ. Assim, o maior valor do *deadline* discreto Dd_σ que satisfaça $(Dd_\sigma + 1)^*\delta \leq D_\sigma$, seja igual a $\lfloor (Dc_\sigma/\delta) - 1 \rfloor$.

Para a discretização do *delay*, considera-se que: seja $dc_\sigma \in \mathbb{R}^+$ o tempo mínimo esperado e δ o período do relógio, deseja-se encontrar o tempo mínimo discreto de espera ($dd_\sigma \in \mathbb{N}^+$), tal que $\delta^*(dd_\sigma) \geq dc_\sigma$, onde, o tempo mínimo discreto será o menor inteiro contido em $dd_\sigma = \lceil (dc_\sigma/\delta) \rceil + 1$.

E, por fim, para o tempo mínimo e máximo esperado, considera-se o cálculo do intervalo discreto dado por $[\ dd_\sigma = \lceil (dc_\sigma/\delta) \rceil + 1\ ;\ \lfloor (Dc_\sigma/\delta) - 1 \rfloor\ ]$.

6.2.3 Exemplo de aplicação: Pequeno Sistema de Manufatura

Modelos das especificações temporizadas: deseja-se pelo controle supervisório temporizado que: i) não ocorra *underflow* nem *overflow* no *buffer* de entrada B_i;

ii) que os tempos de resfriamento das peças sejam respeitados; e iii) o tempo mínimo de reinício de M_3 seja cumprido. Com o intuito de garantir as restrições apresentadas, definem-se cinco especificações, quais sejam:

- E_1 **– especificação de tempo Máximo (*Deadline*) -** trata do tempo máximo de permanência das peças do tipo A no buffer intermediário B_i. Uma vez finalizada a operação de M1 (β_1), o início de operação de $M4$ (α_4) deve ocorrer antes de 10s;
- E_2 **– especificação de tempo mínimo e máximo -** deve garantir que as peças do tipo B, mantenham-se durante um intervalo de tempo no buffer B_i. Quando ocorrer o fim de operação de M2 (β_2), o evento de início de operação de M4 (α_4) deve acontecer dentro do intervalo contínuo [4s ; 16s);
- E_3 **– especificação de tempo mínimo (*delay*)-** deve garantir que as peças do tipo C, fiquem por um tempo mínimo de 3s no buffer B_i. Logo, sempre que o fim de operação de M3 (β_3) ocorrer, o evento de início de operação de M4 (α_4) não deve ocorrer antes de 3s;
- E_4 **– especificação de tempo de preparação -** deve garantir um tempo mínimo de preparação para o início de processamento de $M3$. Toda vez que ocorrer o final de operação de M3 (β_3), o reinício de operação (α_3) deve respeitar o tempo mínimo de 2s.
- E_5 **– especificação atemporal de underflow e overflow -** deve garantir que só possa ter uma peça no buffer intermediário e que M4 não inicie sua operação sem peças. Assume-se que B_i tem capacidade unitária.

Quando se tem especificações com valores contínuos, do mesmo modo que nas plantas, o problema que se quer resolver é, como representá-las de forma minimamente conservadora, em termos de *ticks* do relógio.

Para exemplificar o método proposto aplicado ao exemplo, assume-se o período do relógio digital δ de 5s, em que a especificação $E1$ de *Deadline* deve respeitar o tempo máximo para a ocorrência de 10s, o valor do tempo discreto máximo de espera Dd_σ será igual a $\lfloor (Dc_\sigma/\delta) - 1 \rfloor = \lfloor 1 \rfloor = 1$. O *deadline* discreto sempre ocorrerá antes do segundo *tick*, isto é, estritamente antes de 10s. Para o caso de $D_{d\sigma} > 1$, o *deadline* real (Dc_σ) pode ser perdido.

Para a especificação de *delay* (E_4), a máquina $M3$ deve ter um tempo mínimo de preparo de 2s. Dessa forma, o tempo mínimo discreto obtido é $dd_\sigma = \lceil (dc_\sigma/\delta) \rceil + 1 = \lceil (2/5) \rceil + 1 = 2$. Pode-se observar que o tempo mínimo de preparação é respeitado, uma vez que após $M3$ ter finalizado o início de operação, não irá ocorrer antes do segundo

tick (antes de 2s). Cabe ressaltar que, para o valor do *tick* escolhido, a especificação torna-se bastante conservadora, já que o início de um novo processamento de $M2$ só ocorrerá após 10s.

Já para a especificação E_2, a discretização do intervalo de tempo mínimo e máximo $[6s; 16s]$ é dada por $[\ \lceil(dc_\sigma/\delta)\rceil + 1\ ;\ \lfloor(Dc_\sigma/\delta) - 1\rfloor\ ] = [\lceil 0{,}8\rceil + 1; \lfloor 2{,}2\rfloor] = [2;2]$. Nesse caso, a peça B permanecerá no buffer B_i por um tempo mínimo de $5s[$ e um tempo máximo de $15s[$.

A Tabela 8 apresenta os intervalos de tempos discretos obtidos para as especificações acima descritas.

Tabela 8 – Discretização dos Tempos Associados a Σ_{Ej}

Especificações	Intervalos Contínuo	Intervalo Discreto
E1	[0s ; 10s]	[0 , 1]
E2	[4s ; 16s]	[2 , 2]
E3	[3s ; ∞)	[2 , ∞)
E4	[2s ; ∞)	[2 , ∞)

Fonte: Elaborado pelo autor (2021).

Para o exemplo proposto, foram modeladas cinco especificações, as quais resolvem os problemas de interação entre os subsistemas apresentados anteriormente, como ilustrado na Figura 84.

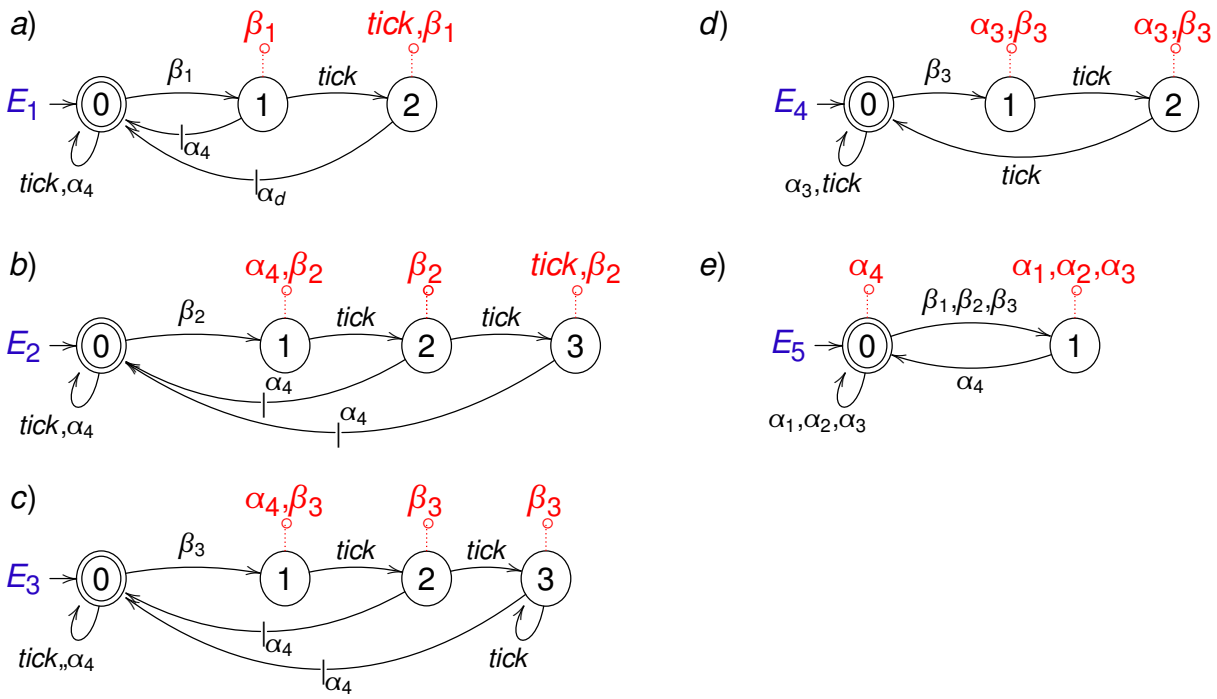
Conforme a Figura 84a, a especificação $E1$ estabelece que, sempre que o evento β_1 ocorrer, o início de operação de M4 (α_4) deve ocorrer antes do segundo *tick*. A especificação $E2$ (Figura 84b) determina que a peça B permaneça no buffer por, no mínimo, um *tick* e seja retirada antes do terceiro *tick*. Na especificação $E3$ (Figura 84c) sempre que o evento fim de operação de $M3$ ocorrer (β_3), o início de operação de M4 (α_4) deve aguardar, no mínimo, a ocorrência de um *tick*. A especificação $E4$ (Figura 84d) garante um tempo mínimo de preparação de um *tick* para o reinício de operação de $M3$ (α_3) e, por último, a especificação $E5$ (Figura 84e) trata o problema de *overflow* e *underflow* para o buffer (b_i) de capacidade unitário.

A partir dos intervalos de tempos discretos, passa-se a uma análise dos tempos associados aos modelos das plantas, em função do refinamento do período do relógio digital.

6.3 INTERVALOS DE TEMPO X GRANULARIDADE DO PERÍODO DO RELÓGIO

Considerando o exemplo da seção anterior, a Tabela 9 mostra os intervalos discretos obtidos, a partir dos tempos associados aos modelos das plantas e o seu intervalo contínuo equivalente para diferentes valores de *ticks*. Essa tabela ilustra os

Figura 84 – Especificações de *Deadline* (E_1); Intervalo Mínimo e Máximo (E_2); Delay (E_3); Setup (E_4) e Underflor/Overflow (E_5)



Fonte: Elaborado pelo autor (2021).

desvios, ou seja, o quão longe os intervalos contínuos resultantes estão dos valores reais. Nota-se que os Intervalos I_{cr} contêm os intervalos originais I_c.

Tabela 9 – Intervalos de Tempo Associados aos Modelos das Plantas x Granularidade do Período do Relógio Digital

δ \ σ , I_c		β_1 [11,3s; 14,5s]	β_2 [17,4s; 20,1s]	β_3 ([13,1s; 19,5s])	β_4 [5,2s; 6,4s]	α_3 [5,3s; ∞)
1s	$\frac{I_d}{I_{cr}}$	[11 , 15] [10s ; 16s)	[17 , 21] [16s ; 22s)	[13 , 20] [12s ; 21s)	[5 , 7] [4s ; 8s)	[5 , ∞) [4s ; ∞)
2s	$\frac{I_d}{I_{cr}}$	[5 , 8] [8s ; 18s)	[8 , 11] [14s ; 24s)	[6 , 10] [10s ; 22s)	[2 , 4] [2s ; 10s)	[2 , ∞) [2s ; ∞)
3s	$\frac{I_d}{I_{cr}}$	[3 , 5] [6s ; 18s)	[5 , 7] [12s ; 24s)	[4 , 7] [9s ; 24s)	[1 , 3] [0s ; 12s)	[1 , ∞) [0s ; ∞)
5s	$\frac{I_d}{I_{cr}}$	[2 , 3] [5s ; 20s)	[3 , 5] [10s ; 30s)	[2 , 4] [5s ; 25s)	[1 , 2] [0s ; 15s)	[1 , ∞] [0s ; ∞)
6s	$\frac{I_d}{I_{cr}}$	[1 , 3] [0s ; 24s)	[2 , 4] [6s ; 30s)	[2 , 4] [6s ; 30s)	[0 , 2] [0s ; 18s)	[0 , ∞] [0s ; ∞)
10s	$\frac{I_d}{I_{cr}}$	[1 , 2] [0s ; 30s)	[1 , 3] [0s ; 40s)	[1 , 2] [0s ; 30s)	[0 , 1] [0s ; 20s)	[0 , ∞] [0s ; ∞)

Fonte: Elaborado pelo autor (2021).

Já a Tabela 10 apresenta os intervalos de tempo obtidos para as especificações temporizadas. Observa-se que as aproximações são sempre internas aos intervalos

originais, ou seja, o intervalo contínuo resultante I_{cr} deve estar contido no intervalo original I_c e dependendo da escolha do *tick*, pode se aproximar mais ou menos.

Tabela 10 – Tempo Associados aos Modelos das Especificações x Granularidade do Período do Relógio Digital

δ \ *Esp.*, I_c		E_1 $[0s; 10s]$	E_2 $[2s; 16s]$	E_3 $[3s; \infty)$	E_4 $[2s; \infty)$
1s	I_d	[0 , 9]	[3 , 15]	[4 , ∞)	[3 , ∞)
	I_{cr}	[0s ; 10s)	[2s ; 16s)	[3s ; ∞)	[2s ; ∞)
2s	I_d	[0 , 4]	[1 , 7]	[3 , ∞)	[2 , ∞)
	I_{cr}	[0s ; 10s)	[2s ; 16s)	[4s ; ∞)	[2s ; ∞)
3s	I_d	[0 , 2]	[2 , 4]	[2 , ∞)	[2 , ∞)
	I_{cr}	[0s ; 9s)	[3s ; 15s)	[3s ; ∞)	[3s ; ∞)
5s	I_d	[0 , 1]	[2 , 2]	[2 , ∞)	[2 , ∞)
	I_{cr}	[0s ; 10s)	[5s ; 15s)	[5s ; ∞)	[5s ; ∞)
6s	I_d	[0 , 0]	[2 , 1] *	[2 , ∞)	[2 , ∞)
	I_{cr}	[0s ; 6s)	-	[6s ; ∞)	[6s ; ∞)

Fonte: Elaborado pelo autor (2021).

O intervalo discreto obtido $I_d = [2, 1]$ é inconsistente (* conforme especificação $E2$ para um $\delta = 6s$), uma vez que o tempo mínimo discreto (dd_σ) obtido pela função teto $\lceil dc_\sigma/\delta \rceil + 1 = \lceil 0{,}33 \rceil + 1 = 2$ é maior que o tempo máximo discreto (Dd_σ), obtido pela função piso $\lfloor (uc_\sigma/\delta) - 1 \rfloor = \lfloor (2{,}67) - 1 \rfloor = 1$.

Neste caso, se $dd_\sigma = 0$, o valor mínimo contínuo resultante é 0s, que esta abaixo do tempo mínimo de 4s especificado, e no caso de $Dd_\sigma = 2$, o valor máximo contínuo resultante é estritamente menor que 18s, o que transgride o tempo máximo de 16s. Para qualquer valor de $\delta >= 5{,}4s$ não é possível garantir que a especificação E_2 seja respeitada, já que o intervalo discreto obtido sempre será inconsistente.

6.3.1 Cálculo do Índice de Conservadorismo (IC)

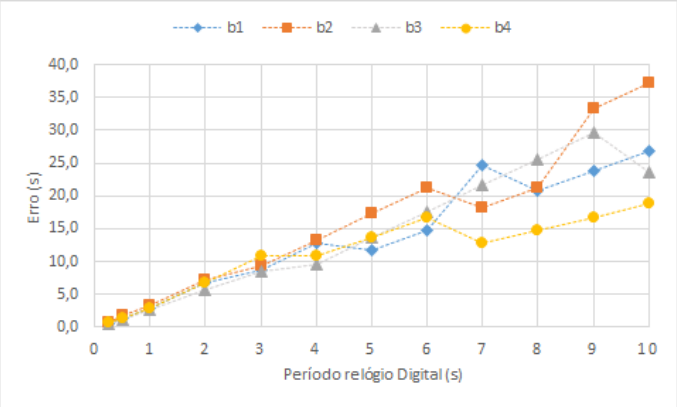
Esta seção tem como objetivo realizar uma análise do quanto o intervalo contínuo correspondente I_{cr} é conservador em relação ao intervalo original I_c, expressando o quão longe os intervalos contínuos resultantes estão dos valores originais. Dessa forma, apresentam-se os desvios dos tempos associados aos modelos das plantas e especificações, em virtude da variação do período do relógio digital.

IC dos tempos associados aos modelos da planta: para a análise do Índice de Conservadorismo, a normalização do erro é realizada em relação ao valor do tamanho do intervalo contínuo dado. Para cada valor do período do relógio, é possível verificar quanto tempo a mais foi adicionado ao intervalo original. Por exemplo, para o evento b_1 em que $I_c = [11{,}3s; 14{,}5s]$ tem-se que o tamanho do intervalo é igual

3,2s. Considerando o *tick* igual a 1s, ao discretizar I_c, o intervalo discreto calculado será I_d = [11,15] e o intervalo contínuo correspondente I_{cr} = [10s; 16s), gerando um erro total de 2,8s adicionado ao tamanho do intervalo original. Assim, o erro total é de (2,8s/3,2s) $*$ 100 = 87,5%, significando que o intervalo contínuo correspondente tem uma margem de 87,5% maior do que o intervalo original.

A Figura 85 apresenta o grau de conservadorismo dos eventos esperados pelo refinamento dos valores do *tick*, que mostra o quanto aumentou a margem de tolerância do erro individual para cada evento, pois em determinados períodos, podem ocorrer eventos mais críticos. Assim, é possível observar, que o evento b_1 será mais crítico (terá um erro maior) para um *tick* de 7s, do que para um *tick* de 8s ou 9s, já os eventos b_2 e b_4 serão mais críticos para um *tick* de 6s, do que para 7s ou 8s.

Figura 85 – Erro Absoluto Associado aos Eventos Remotos em Função do Refinamento de δ



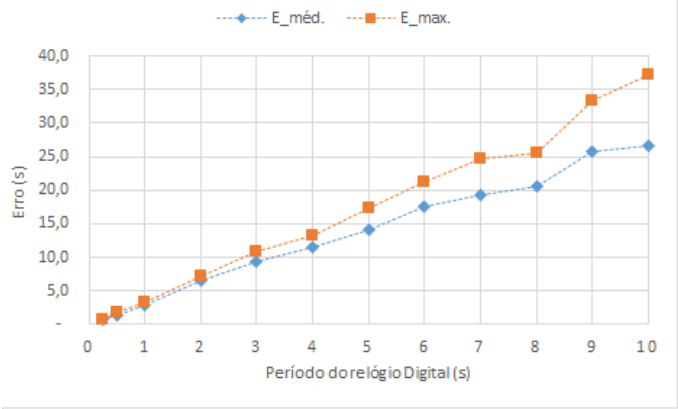
Fonte: Elaborado pelo autor (2021).

A Figura 86 mostra o índice de conservadorismo dos erros médio e máximo associados aos tempos de operações das máquinas, onde é possível observar que, nesse caso, os valores dos erros aumentam de forma linear.

IC dos tempos associados aos modelos das especificações: para o caso das especificações, a normalização do erro deve ser realizada em relação ao valor absoluto do tempo mínimo. Nesse caso agora, a aproximação deve ser interna e não externa como para a planta, ou seja, $I_{cr} \subset I_c$ e dependendo da escolha do *tick*, essa aproximação pode ser maior ou menor.

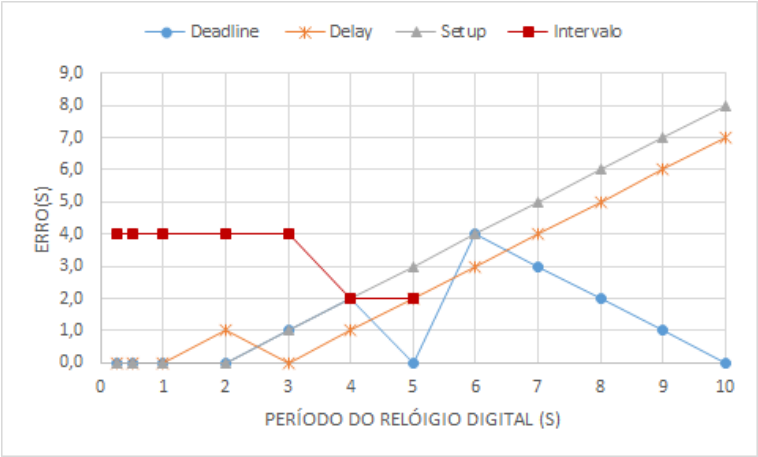
A Figura 87 mostra o erro dos tempos associados aos modelos das especificações temporizadas. Pode-se observar que a especificação de intervalo máximo e mínimo é a mais crítica e, para δ superior a 5s, esse intervalo discretizado é inconsistente e não aparece no gráfico.

Figura 86 – Erro Real Associado aos Eventos Remotos em Função do Refinamento do Período do Relógio Digital



Fonte: Elaborado pelo autor (2021).

Figura 87 – Erro Real Associado aos Tempos das Especificações pelo Refinamento do Período do Relógio Digital



Fonte: Elaborado pelo autor (2021).

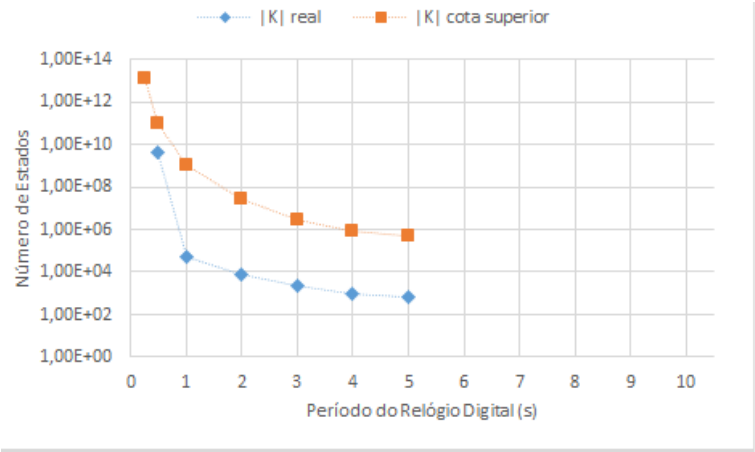
6.3.2 Complexidade dos Modelos x Granularidade do período do Relógio

Além de analisar o Índice de Conservadorismo dos modelos das plantas e das especificações, um ponto importante é, o quanto esses erros afetarão o desempenho do sistema? Do ponto de vista da complexidade, os fatores limitantes são o tempo de processamento da síntese, a memória para o cálculo da síntese dos supervisores, e a memória para a implementação no CLP.

Para um software fazer o cálculo do produto síncrono este tem de reservar uma memória do tamanho do produto cartesiano dos modelos, para o *TTCT* assume-se que este limite é da ordem de 10^6, então os valores devem ser calculados próximo desse número de estados, a partir de uma cota superior estimada $|K|$. Para este estudo de caso, consegue-se chegar até 1s de período do relógio.

A Figura 88 ilustra o número de estados do comportamento desejado para diferentes valores de *tick*, onde o $|K|$ real representa o autômato obtido pela ferramenta computacional *TTCT* e o $|K|$ cota superior, pelo produto do número de estados das plantas G_i com as especificações E_j. O pior caso é representado pela cota superior, em que mostra como a estimativa será boa ou ruim. Como pode ser observado, o *TTCT* não consegue realizar o cálculo para valores de *tick* < 0,5s. Já para valores *tick* $\geq$ 5,4s não é possível realizar o processo de síntese, uma vez que a especificação de intervalo ($E2$) não pode ser garantida.

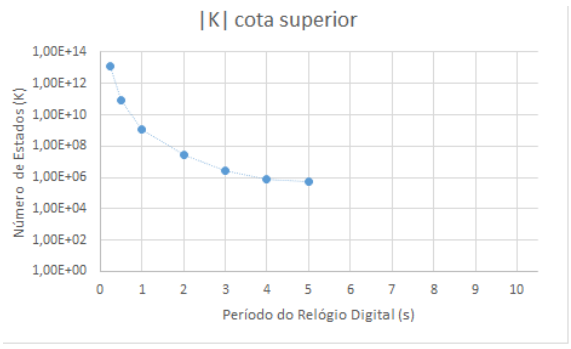
Figura 88 – Número de Estados do Automato K em Relação a Granularidade do Período do Relógio Digital



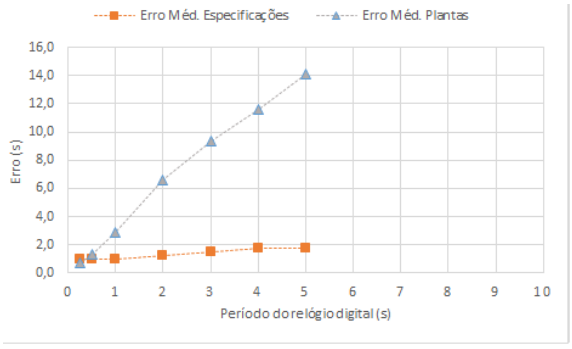
Fonte: Elaborado pelo autor (2021).

A Figura 89a mostra a complexidade do modelo do comportamento estimado $|K|$ cota superior e a Figura 89b ilustra os erros médios das plantas e das especificações, em relação a variação do período do relógio digital. Para este gráfico, pode-se definir um fator de custo por número de estados e pelo erro médio, obtendo um ponto de equilíbrio, mas isso deve ser baseado na experiência do projetista.

Figura 89 – Complexidade Autômato |K| e Erros Médios Reais



(a) Complexidade Autômato |K| Cota Superior



(b) Erros Médios Reais

Fonte: Elaborado pelo autor (2021).

Um aspecto importante é a qualidade da solução, quanto mais conservador (maior o erro) mais se perde em desempenho.

Portanto, objetiva-se trabalhar com o menor erro, desde que a síntese dos supervisores e coordenadores reduzidos, sejam possíveis e implementáveis no CLP (em termos de memória). Logo, o maior autômato possível de se implementar junto com um erro admissível, provavelmente será a melhor solução em termos de desempenho.

Nesse contexto, primeiramente pode-se analisar o limite de complexidade possível de se lidar. Na sequência, observa-se como os erros estão se comportando em torno desse ponto, ou seja, se quisermos ganhar em relação à complexidade, como o erro vai aumentar. Assim, é possível realizar uma análise qualitativa em torno do ponto limite, pois aumentando-se o erro, é possível verificar o quanto se está ganhando em complexidade.

Cabe ressaltar que a desvantagem de usar o relógio pequeno está relacionada, tanto ao processo de síntese como a implementação em CLPs. No caso da implementação, o tamanho do código gerado não mudará no SPT, uma vez que as plantas são implementadas como GTAs (só altera o tamanho do contador), porém, os tamanhos dos supervisores reduzidos e dos coordenadores podem ser alterados. Assume-se que a ordem do coordenador terá a mesma grandeza de K e que o software (*TTCT*) consegue reduzir supervisores até 2000 estados.

Assim, a partir do gráficos mostrados anteriormente relacionados aos erros e mais o gráfico da complexidade, o projetista pode avaliar e definir um valor para o período do relógio digital (δ). De uma forma geral, o *tick* escolhido será o menor valor em que é possível realizar a síntese do CMLT, implementar no CLP e conter um erro aceitável.

A Tabela 11 apresenta o tamanho dos modelos monolíticos, em virtude dos refinamentos do *tick*. Pode-se observar que para $\delta > 5$ (- *) a especificação $E2$ não pode ser garantida, e para $\delta < 1$ o supervisor não pode ser calculado (explosão de estados). Já na Tabela 12 é possível ver os tamanhos dos modelos para o CMLT, em função do refinamento do *tick*, onde para $\delta < 2s$ não é possível obter o coordenador reduzido, o que torna a complexidade da implementação em CLPs, inviável.

Tabela 11 – Tamanho do Modelo Monolítico x Período do Relógio Digital (*tick*)

δ	0,25s	0,5s	1s	2s	3s	4s	5s	6s	7s	8s	9s	10s
G	-	1019745	95013	10920	3150	1536	980	- *	- *	- *	- *	- *
E	-	6237	1122	270	96	60	48	- *	- *	- *	- *	- *
K	-	-	71469	11237	3724	2015	1417	- *	- *	- *	- *	- *
S	-	-	67823	6781	309	265	160	- *	- *	- *	- *	- *

Fonte: Elaborado pelo autor (2021).

Tabela 12 – Tamanho do Modelo CMLT x Período do Relógio Digital (*tick*)

δ	0,25s	0,5s	1s	2s	3s	4s	5s	6s	7s	8s	9s	10s
G_{loc1}	-	-	153	60	35	24	20	- *	- *	- *	- *	- *
G_{loc2}	-	-	207	78	45	32	28	- *	- *	- *	- *	- *
G_{loc3}	-	-	243	84	50	32	28	- *	- *	- *	- *	- *
G_{loc4}	-	-	27	14	10	8	7	- *	- *	- *	- *	- *
G_{loc5}	-	-	95013	10920	3150	1536	980	- *	- *	- *	- *	- *
S_{r1}	-	-	11	6	4	3	3	- *	- *	- *	- *	- *
S_{r2}	-	-	17	9	6	5	4	- *	- *	- *	- *	- *
S_{r3}	-	-	19	12	1	8	7	- *	- *	- *	- *	- *
S_{r4}	-	-	4	3	3	3	3	- *	- *	- *	- *	- *
S_{r5}			2	2	2	2	2	- *	- *	- *	- *	- *
G_{conf}	-	-	71469	6781	3924	1671	1222	- *	- *	- *	- *	- *
$Coord$	-	-	67823	6781	2924	265	160	- *	- *	- *	- *	- *
$Coord_{red}$	-	-	-	-	180	34	21	- *	- *	- *	- *	- *
$Total_{Imp}$	-	-	-	-	206	65	50	- *	- *	- *	- *	- *

Fonte: Elaborado pelo autor (2021).

A Tabela 13 apresenta o número de linhas de código estimado para a implementação no CLP de S e $Coord_{red}$. Esta estimativa foi apoiada no estudo de caso que será apresentada na Seção 7.4, em que cada transição implementada gerou, em média, três linhas de código para os supervisores reduzidos e quatro linhas para as plantas locais.

Tabela 13 – Tamanho do Código Implementação Monolítico x Modular Local (*tick*)

δ	3s	4s	5s
S	309/584	265/458	160/272
Nº Cod. CLP	1168	916	544
$Coord_{red}$	180/848	34/134	21/80
Nº Cod. CLP	1696	268	160

Fonte: Elaborado pelo autor (2021).

Pode-se concluir que a abordagem modular mostrou-se mais vantajosa em termos de números de estados, apesar do cálculo do coordenador ter a mesma ordem de complexidade do cálculo do supervisor monolítico. Do ponto de vista de implementação, sempre que o CMLT conseguir encontrar uma solução, teremos um código mais leve (consumo menor de memória do CLP), como se pode observar na Tabela 13, para os valores de *tick* igual a 3s e 4s. Porém, quando o CMLT não conseguir gerar o coordenador reduzido, a solução será a monolítica e, neste caso, o número de estados

aumentará consideravelmente.

A decisão de qual período do relógio escolher, depende do hardware disponível. Conforme Silva (2007), os CLPs podem ser classificados como pequeno, médio ou grande porte, como descritos a seguir:

- 1 - CLPs de Pequeno Porte (Nano e Micro) - máximo 16 entradas e 16 saídas, normalmente só digitais, baixo custo e reduzida capacidade de memória (máximo 512 passos);
- 2 - CLPs de Médio Porte - capacidade de entrada e saída de até 256 pontos, digitais e analógicas, e costumam permitir até 2048 passos de memória; e
- 3 - CLPs de Grande Porte - se caracterizam por uma construção modular, permitindo a utilização de até 4096 pontos de E/S e mais de 2048 passos de memória.

Nesse exemplo, para o coordenador obtido de 2567 transições, o código gerado será maior que 8000 linhas para implementá-lo como uma máquina de estado, o que provocaria um estouro de memória do CLP, considerando que a CPU 1214C utilizada para o estudo, possui 50KB de memória interna alocada.

A partir da análise realizada em função do refinamento do relógio digital, passa-se, na próxima seção, a proposta de um método que possa orientar o projetista sobre a escolha do período do relógio digital.

6.4 CONCLUSÃO DO CAPÍTULO

Neste capítulo, foi apresentado um método para modelagem da planta e das especificações em sistemas de manufatura sujeitos a incertezas de operação e a restrições temporais, que considera a discretização dos tempos associados aos eventos de um problema estabelecido no mundo real. Desta forma, os tempos discretos são expostos em função de um período do relógio digital (*tick*), como nos modelos de BW. Foram apresentados os passos para a discretização dos modelos das plantas e das especificações e um exemplo para ilustrar o método proposto. Além disso, foi proposta uma metodologia que se fundamenta no erro máximo aceitável, na complexidade da síntese do CMLT e no tamanho do código implementado no CLP, para auxiliar o projetista na escolha do período do relógio digital a ser utilizado no problema real que se deseja controlar. Com base no exemplo proposto, pode-se concluir que os métodos propostos apresentaram resultados satisfatórios.

7 MODELAGEM E IMPLEMENTAÇÃO DO CONTROLE SUPERVISÓRIO APLICADO A UM SISTEMA MODULAR DE PRODUÇÃO SOB INCERTEZAS DE PROCESSAMENTO E RESTRIÇÕES DE TEMPO

Este capítulo apresenta um estudo de caso completo que abrange a modelagem, a síntese e a implementação em CLP do controle supervisório para um sistema de fabricação real, com tempos de processamento incertos, devido à intervenção humana direta, operação flexível e restrições de tempo, como e *time lag* e tempo de preparação. Esse estudo incia-se com a modelagem das plantas e especificações, conforme o método proposto no Capítulo 6. Para a síntese de controle supervisório temporizado, usa-se a abordagem modular local de Schafaschek, Queiroz e Cury (2017), apresentada no Capítulo 4. Além disso, neste estudo de caso, é aplicada a nova arquitetura para a implementação de controle supervisório temporizado em controladores lógicos programáveis (CLP), proposta no Capítulo 5.

Para um melhor entendimento, o capítulo está estruturado da seguinte forma: na Seção 7.1 é apresentada a estação MPS utilizada no estudo de caso, com o levantamento dos dados obtidos experimentalmente; na Seção 7.2 mostra-se como modelar o exemplo motivador; é realizada a síntese dos supervisores locais na Seção 7.3; a implementação do CMLT em CLPs com base na arquitetura proposta na Seção 5.1 é contemplada na Seção 7.4; e, por fim, apresentam-se os resultados e a conclusão.

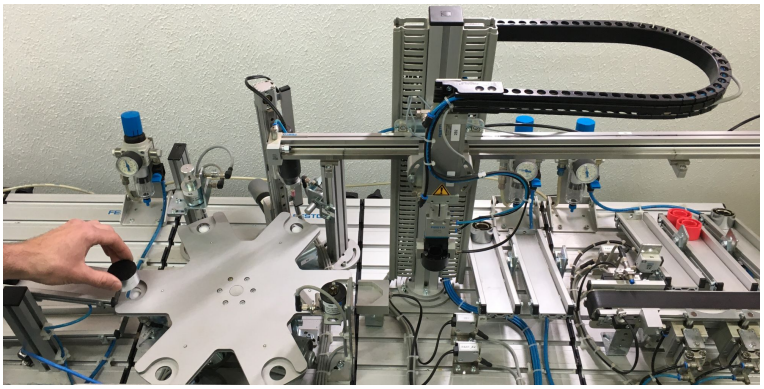
7.1 APLICAÇÃO: ESTUDO DE CASO

Esta seção apresenta um caso real, que será utilizado como referência do exemplo motivador para sistemas de manufatura, com incertezas de processamento e restrições temporais. Sistemas Modulares de Produção (MPS) são compostos por máquinas modulares e controladores reconfiguráveis, que combinam a alta taxa de produção com a flexibilidade de sistemas de manufatura. A planta MPS da Festo localizada no Laboratório de Automação do Instituto Federal de Santa Catarina (IFSC) *campus* Chapecó, é constituída pelas estações *PROCESSING* e *HANDLING* controladas individualmente por um CLP (Siemens da série S7-1200)(Figura 90).

As peças são inseridas manualmente na estação *Processing* ($M1$) através de um operador (evento $a1$) e a seguir são transportadas por meio de uma mesa giratória para a posição de teste, que conforme o resultado obtido, pode sofrer diversos processos de usinagem. Na sequência, as peças são encaminhadas para um *buffer* intermediário B (evento $b1$) e são armazenadas até que a próxima máquina inicie suas operações.

A estação *Handling* ($M2$) objetiva a separação de peças para diferentes *buffers* de acordo com os critérios de seleção. Nessa estação, as peças são retiradas de B, por

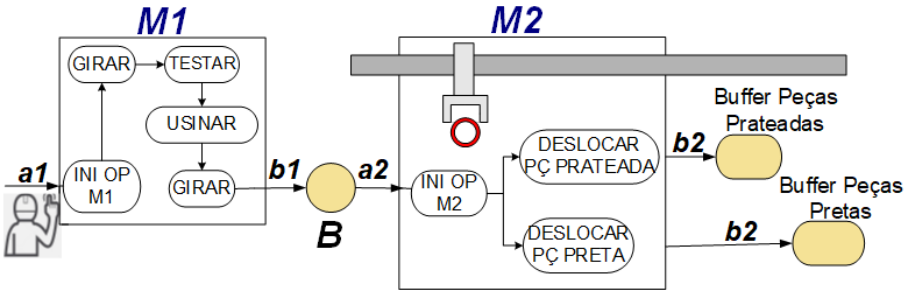
Figura 90 – MPS® Estações Processing e Handling



Fonte: Elaborado pelo autor (2021).

meio de um manipulador eletropneumático (evento *a*2) e deslocadas para os diferentes *buffers* de saída, de acordo com sua cor (evento *b*2), conforme ilustrado na Figura 91.

Figura 91 – Representação das estações *Processing* (M1), *Buffer* (B) e *Handling*(M2)



Fonte: Elaborado pelo autor (2021).

A partir da apresentação do problema para o estudo de caso, passa-se, para a obtenção experimental dos intervalos de tempo contínuo. Para definir os intervalos de tempos associados aos eventos (Σ_{esp}, Σ_{rem}), fez-se uma análise do conjunto de valores de tempos amostrados, obtidos de forma experimental para as máquinas $M1$ e $M2$. O intervalo foi calculado utilizando-se a distribuição de *t-Student*[1], com uma probabilidade γ de 99%.

Assim, os dados obtidos experimentalmente mostraram que o tempo mínimo que o operador leva para posicionar uma peça na entrada de $M1$ e iniciar seu processo de operação é de 2,1s. O processo de operação $M1$ pode variar em um intervalo de [5,8s; 9,6s]. A máquina $M2$ passa a operar automaticamente, sem tempo de *setup*, e seu tempo de operação varia entre [9,8s; 15,4s].

O sistema apresentado deve ser controlado de forma a respeitar as restrições *underflow* e *overflow* para um buffer unitário *B*. Além disso, o sistema também deve obedecer a especificações de tempo, como tempo máximo de 11s para a permanência

[1] Em função de se ter amostras com tamanhos menores que 30.

da peça no buffer intermediário *B* e tempo mínimo de 1s para reiniciar a operação de *M*2.

Soluções clássicas que utilizam escalonamento fixo nem sempre resolvem problemas com as características de tempo descritas acima e, quando podem resolvê-los, também podem gerar soluções de baixo desempenho, por considerar apenas o pior tempo de processamento. Por exemplo, uma possível solução seria permitir que apenas uma peça seja processada por vez no sistema, iniciando a segunda operação assim que a peça estiver disponível. Embora essa restrição satisfaça as restrições de tempo, é uma solução conservadora e ineficiente. Para permitir a operação simultânea de duas máquinas, respeitando a especificação *time lag*, a estratégia de controle deve ser reativa à duração não determinística das operações humanas e da máquina.

Assim, para lidar com restrições de tempo e especificações incertas, este estudo explora a TCS de SEDT para a síntese de supervisores minimamente restritivos. Passa-se, então, a modelagem da planta e das especificações, aplicada ao estudo de caso do exemplo motivador.

7.2 MODELAGEM

Demonstra-se, nesta seção, como se pode modelar o exemplo motivador ilustrado na Seção 7.1, como modelo de SEDT, conforme apresentado nas seções 6.1 e 6.2

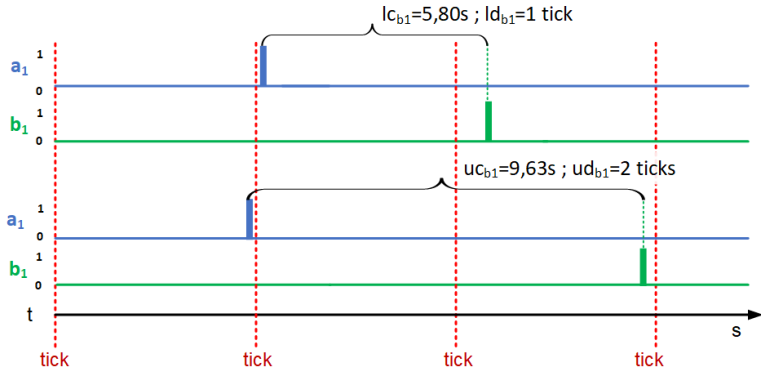
7.2.1 Planta

Para a discretização proposta para o estudo de caso, apresentado na Seção 7.1, assume-se aqui um período de δ = 5s para o relógio digital global, uma vez que é um valor que leva à obtenção de autômatos com pequeno número de estados. O impacto do período do relógio digital, na complexidade do modelo e na produtividade do sistema de manufatura, é analisado na Seção 7.5.

Seja o evento esperado, dado pelo intervalo de tempo contínuo lc_{b_1} = [lc_{b_1} ; uc_{b_1}] = [5,8s; 9,6s], seu intervalo discreto é determinado por [$\lfloor lc_{b_1}/\delta \rfloor$; $\lceil (uc_{b_1}/\delta) \rceil$] = [$\lfloor$5,8/5$\rfloor$; $\lceil$(9,6/5)$\rceil$] = [1, 2]. A Figura 92 ilustra a ideia de uma discretização minimamente conservadora. Nesse caso, pode-se ver que o modelo discreto é conservador, uma vez que, o intervalo contínuo resultante obtido por lcr = [$(ld_{b_1} - 1)^*\delta$; $(ud_{b_1} + 1)^*\delta$] = (0s; 15s), contém o intervalo contínuo originalmente dado.

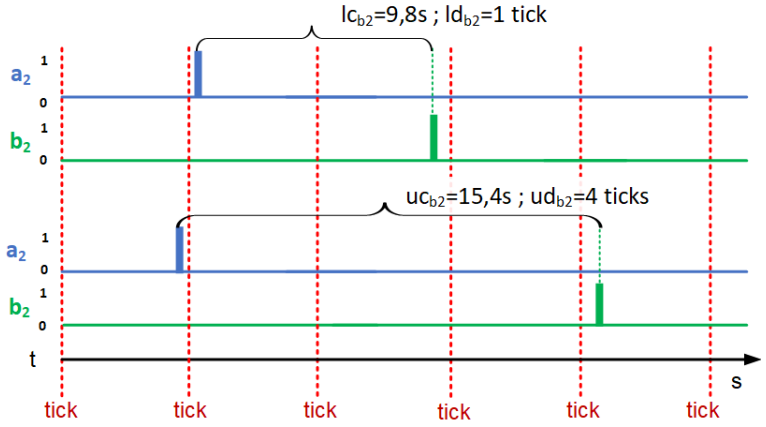
Para o intervalo de tempo associado ao evento esperado b_2 ([9,8 ; 15,4]) como pode ser observado na Figura 93, o intervalo discreto associado a b_2 é [1 , 4]. Pode-se observar que o modelo discreto é conservador, uma vez que, o intervalo contínuo resultante obtido (0s ; 25s) contém o intervalo contínuo originalmente dado.

Figura 92 – Discretização do intervalo [5,8s; 9,6s] para o evento b_1 na planta, com um período de relógio $\delta = 5s$



Fonte: Elaborado pelo autor (2021).

Figura 93 – Discretização do intervalo [9,8s; 15,4s] para o evento b_2 na planta, com um período de relógio $\delta = 5s$

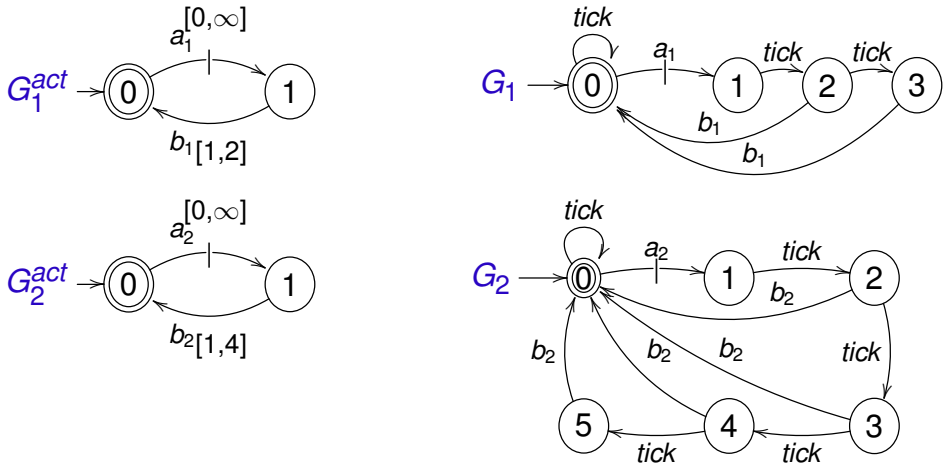


Fonte: Elaborado pelo autor (2021).

Seja o evento remoto, dado pelo intervalo $lc_{a_1} = [lc_{a_1}; \infty)] = [1{,}98s; \infty)$, o intervalo discreto associado a a_1 é dado por $[ld_{a_1} = \lfloor 1{,}98/5 \rfloor; ud_{a_1} = \infty)$, logo $ld = [0, \infty)$. Já para o evento a_2, o intervalo discreto obtido é $I_d = [0, \infty)$.

O sistema de manufatura apresentado é modelado por dois subsistemas concorrentes, correspondentes às estações $M1$ e $M2$, com o *tick* sendo o único evento compartilhado. Os respectivos *GTAs* G_1^{act} e G_2^{act} e *GTTs* G_1 e G_2 são mostrados na Figura 94. Os eventos associados são $\Sigma_u = \{b_1, b_2\}$; $\Sigma_c = \{a_1, a_2\}$; $\Sigma_{for} = \{a_2\}$. Como pode ser observado, os eventos a_1 e a_2 estão elegíveis em $[0, \infty)$. O evento b_1 está associado ao intervalo discreto $[1, 2]$, uma vez que exatamente um *tick* sempre precede a ocorrência de b_1 e o evento b_2 ao intervalo discreto [1,4], já que pelo menos um *tick* do relógio digital e no máximo três *ticks* sempre precedem b_2.

Figura 94 – Modelos SEDT para *M*1 (a) e *M*2 (b) com os GTAs correspondentes à esquerda e GTTs à direita



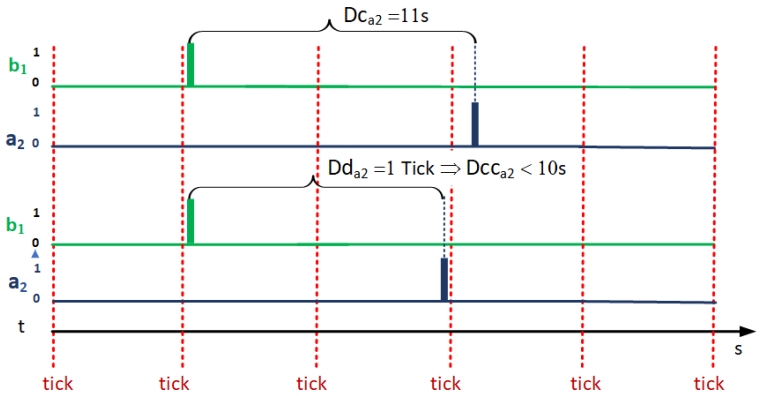
Fonte: Elaborado pelo autor (2021).

7.2.2 Especificações

O sistema apresentado deve ser controlado para respeitar as restrições *underflow* e *overflow* para um buffer unitário *B*. Além disso, o sistema também deve obedecer às especificações de tempo, como tempo máximo de 11s para a permanência da peça no *buffer* intermediário *B* e tempo mínimo de 1s para reiniciar a operação de *M*2.

Conforme apresentado na Seção 6.2, para a discretização proposta, considera-se aqui um período de δ de 5s, em que a especificação de *Deadline* deve respeitar o tempo máximo para a ocorrência de 11s. O valor do tempo máximo discreto Dd_{a2} será igual a $\lfloor (Dc_{a2}/\delta) - 1 \rfloor = \lfloor 11/5 - 1 \rfloor = 1$ (Figura 95). O deadline discreto sempre ocorrerá antes do segundo *tick*, isto é, estritamente antes que 10s.

Figura 95 – Discretização da especificação de tempo máximo - Número máximo de *ticks*, com um período de relógio $\delta = 5s$

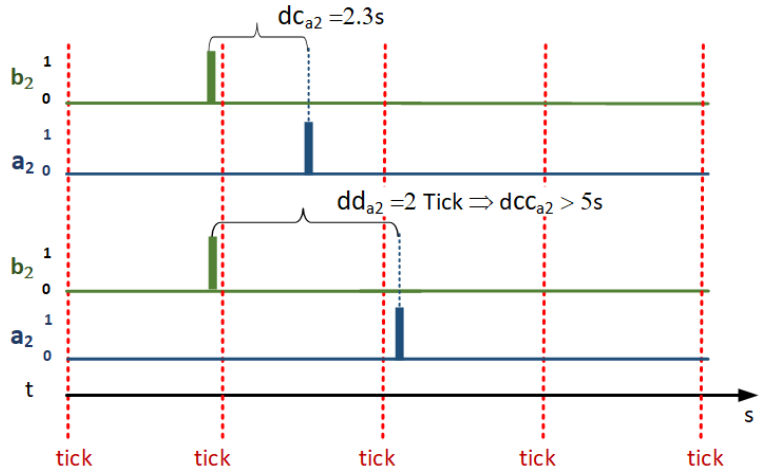


Fonte: Elaborado pelo autor (2021).

Para a especificação de *delay*, a máquina *M*2 deve ter um tempo mínimo de

preparo de 2,3s. Assim, o tempo mínimo discreto obtido é $dd_{a2} = \lceil (dc_{a2}/\delta) \rceil + 1 = \lceil (2{,}3/5) \rceil + 1 = 2$ (Figura 96). Pode-se observar que o tempo mínimo de preparação é respeitado, pois após $M2$ ter finalizado, o início de operação de $M1$ não ocorrerá antes do segundo *tick* (antes de 2,3s). Cabe ressaltar que, para o valor do *tick* escolhido, a especificação torna-se bastante conservadora, já que o início de um novo processamento de $M2$ só ocorrerá após 10s.

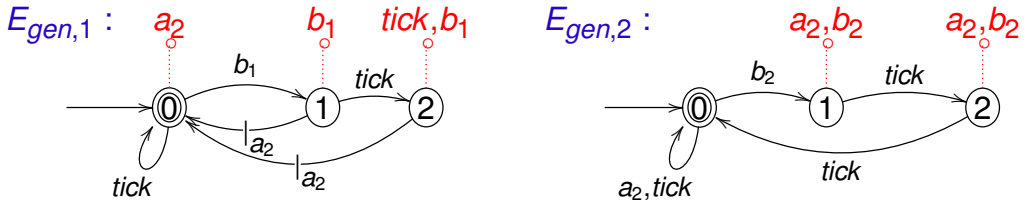
Figura 96 – Discretização da especificação de tempo mínimo - Número mínimo de *ticks*, com um período de relógio $\delta = 5s$



Fonte: Elaborado pelo autor (2021).

Duas especificações temporais são impostas ao sistema de manufatura apresentado. A Figura 97 apresenta seus modelos GTTs.

Figura 97 – Especificações para time-lag ($E_{gen,1}$) e tempo de preparação ($E_{gen,2}$)



Fonte: Elaborado pelo autor (2021).

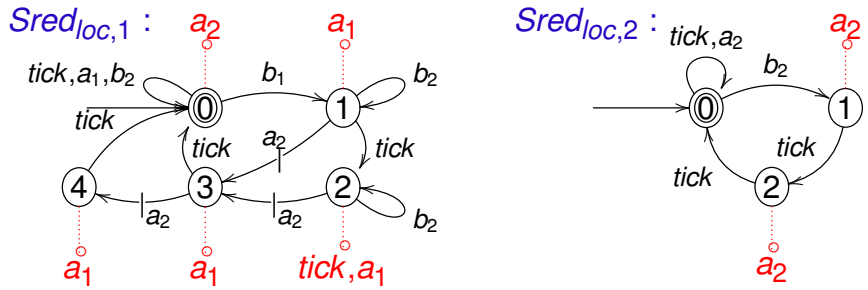
7.3 SÍNTESE

Nesta seção será realizada a síntese dos supervisores locais de acordo com Schafaschek, Queiroz e Cury (2017). Essa abordagem consiste em obter modelos locais pela composição dos subsistemas afetados por cada especificação e os supervisores, então, são calculados sobre esses modelos, impondo localmente o comportamento especificado. Para sintetizar os controladores, utiliza-se o TTCT Wonham e Cai (2019).

No exemplo aqui proposto, ambos os subsistemas G_1 e G_2 compartilham eventos com as especificações E_1, e a especificação E_2 afeta apenas G_2. Então, $G_{loc,1} = G_1||G_2$ e $G_{loc,2} = G_2$. Os comportamentos desejados das plantas locais são obtidos a partir da composição dos comportamentos marcados e suas respectivas especificações, denotados como $K_{loc,j}$.

Para a especificação não controlável E_1, um supervisor $S_{loc,1}$ para a linguagem controlável suprema $SupC(K_{loc,1}, G_{loc,1})$ é calculado e reduzido para $Sred_{loc,1}$. Já que $K_{loc,2} = E_2||L_m(G_{loc,2})$ é controlável em relação a $G_{loc,2}$, o modelo de especificação E_2 pode ser usado diretamente como um supervisor local reduzido ($Sred_{loc,2} = E_2$). Ambos os supervisores reduzidos são mostrados na Figura 98, onde as setas vermelhas tracejadas apontam os eventos a serem desabilitados em cada estado.

Figura 98 – Supervisores locais reduzidos Sr_1 and Sr_2.



Fonte: Elaborado pelo autor (2021).

A Tabela 14 mostra o número de estados/transições dos modelos dos autômatos gerados na síntese de supervisores modulares reduzidos.

Tabela 14 – Número de estados/transições - Síntese CMLT

j	$E_{gen,j}$	$G_{loc,j}$	$K_{loc,j}$	$S_{loc,j}$	$Sred_{loc,j}$
1	3/5	24/53	50/94	20/36	5/12
2	3/5	6/10	8/12	8/12	3/5

Fonte: Elaborado pelo autor (2021).

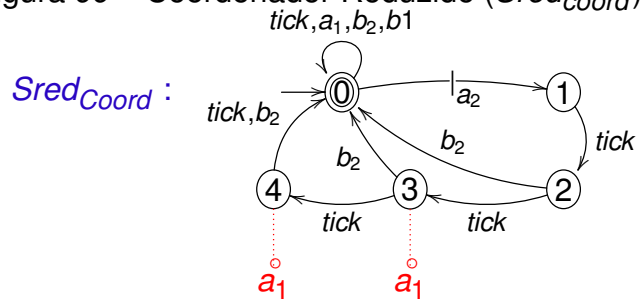
Para garantir que não haja conflito entre as ações dos supervisores locais quando combinados, deve-se verificar a existência de um conjunto de supervisores conflitantes que impeçam o controlador local modular de ser aplicado diretamente, caso não haja, o resultado da ação modular é o mesmo da ação centralizada. Deve haver, também, uma verificação da coercividade conjunta das linguagens resultantes. Caso alguma das condições seja violada, deve-se resolver o conflito.

Pode-se verificar que $G_{conf} = Sred_{loc,1}||Sred_{loc,2}||G_1||G_2$, com 24 estados, é bloqueante e, portanto, a conjunção $Sred_{loc,1}$ e $Sred_{loc,2}$ é conflitante para G. Além

disso, $L_m(G_{conf})$ não é controlável em relação a G, o que significa que $Sred_{loc,1}$ e $Sred_{loc,2}$ não são coercitivos em conjunto.

Os conflitos de bloqueio e coercividade podem ser resolvidos por um supervisor modular adicional, denominado coordenador, obtido computando-se um supervisor não bloqueante ($SupC(L_m(G_{conf}),G)$), que pode ser reduzido ao supervisor $Sred_{coord}$ (Figura 99), por meio do algoritmo de Su e Wonham (2004), tomando G_{conf} como planta do coordenador. A conjunção de $Sred_{loc,1}$, $Sred_{loc,2}$ e $Sred_{coord}$ corresponde ao supervisor minimamente restritivo e não bloqueante, que preserva o comportamento das estações de processamento e tratamento, dentro da segurança e do tempo de restrições.

Figura 99 – Coordenador Reduzido ($Sred_{coord}$)



Fonte: Elaborado pelo autor (2021).

Um ponto importante a ser analisado se dá em relação ao refinamento do período do relógio digital, no sentido de verificar o quão impactante a escolha da granularidade do período terá em relação a complexidade dos autômatos, como pode ser visto na Tabela 15. Para certos valores, o problema de controle não é solucionável pelo TTCT: para $\delta = 0{,}5s$, o coordenador reduzido não pode ser obtido, tornando a complexidade da implementação em CLPs inviável; para $\delta \geq 11s$, a especificação *time lag* não é garantida; e para o $\delta = 0{,}2s$ não é possível obter o supervisor reduzido (Sr_{loc1}). Vale ressaltar que para o *tick* com períodos maiores do que 11s, o problema de controle não é solucionável, uma vez que não é possível garantir a especificação *time lag*.

7.4 IMPLEMENTAÇÃO DE CONTROLE SUPERVISÓRIO MODULAR TEMPORIZADO EM CLPS

A seguir será implementado para o estudo de caso, o CMLT em CLPs com base na arquitetura proposta na Seção 5.1.

7.4.1 Implementação Relógio Digital Global

O relógio digital global é implementado através da função *Cycle Clock Memory*, que é um sinalizador que alterna seus valores binários periodicamente. Para esta

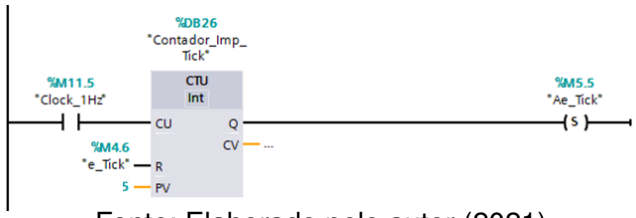
Tabela 15 – Tamanho do Modelo x Período do relógio digital (*tick*)

Valor do *tick*		**10S**	**5S**	**3S**	**2S**	**1S**	**0,5S**	**0,2S**
G_{loc1}	*E*	12	24	48	70	234	825	4661
	T	30	53	105	146	447	1481	7953
G_{loc2}	*E*	6	6	8	10	18	36	79
	T	9	10	12	15	26	46	108
S_{loc1}	*E*	9	20	32	63	229	912	5727
	T	15	36	60	123	437	1680	10370
S_{loc2}	*E*	6	8	10	12	20	36	82
	T	9	12	14	17	28	49	111
$Sred_{loc1}$	*E*	4	5	7	8	13	284	X
	T	10	12	15	20	37	702	X
$Sred_{loc2}$	*E*	3	3	3	3	3	4	4
	T	5	5	5	5	5	6	6
G_{conf}	*E*	16	32	49	87	275	1155	6804
	T	24	50	83	153	493	1950	11504
Coord	*E*	9	18	30	62	229	821	4871
	T	13	28	48	106	416	1456	8522
$Sred_{coord}$	*E*	4	5	6	5	4	X	X
	T	8	12	12	9	8	X	X

Fonte: Elaborado pelo autor (2021).

aplicação, uma memória de relógio com 1 *Hz* foi composta em série com um contador incremental para acionar uma variável *Ae_Tick* a cada 5 segundos, conforme mostrado na Figura 100.

Figura 100 – Implementação Relógio Digital Global



Fonte: Elaborado pelo autor (2021).

O ciclo de varredura do CLP deve estar sincronizado com o relógio digital, desta forma, deve-se estimar um valor múltiplo do relógio digital global. Para o exemplo apresentado, o ciclo mínimo de varredura é 40*ms*, o que significa que a cada 125 ciclos o evento *e_Tick* é gerado pelo CLP e, a variável *Ae_Tick* é definida. Portanto, para garantir essa sincronia, se o período do ciclo de varredura for menor que o período mínimo do ciclo escolhido, a CPU aguardará até que o período mínimo do ciclo seja atingido.

7.4.2 Implementação Supervisores Modulares

Os supervisores modulares (e coordenadores) obtidos na Seção 7.3 são implementados como máquinas de estados concorrentes, de forma que as desabilitações e preempções das respectivas variáveis internas (De_a_i, i = {1,2} e *Preempt_tick*) são sinalizados pelo SPT, de acordo com os estados ativos.

A Figura 101 ilustra a implementação em Texto Estruturado (ST) do supervisor $Sred_{loc,1}$. Na primeira parte do código, cada estado é representado por uma variável inteira (*Sr*1 *_St*) e cada transição seta o estado posterior e reseta o anterior. A variável *e_blk_Sr*1 garante que apenas uma transição ocorra em cada supervisor, no mesmo ciclo de varredura do CLP. Para que se tenha consistência na implementação, essa variável deve sempre ser zerada no início de cada ciclo e setada quando uma nova transição ocorrer. Observa-se que os auto-laços não são implementados, uma vez que não alteram os estados dos supervisores. Já na segunda parte, as ações de controle (desabilitações e preempção do tick) dependem dos estados dos supervisores reduzidos.

7.4.3 Implementação do Sistema Produto Temporizado

A implementação dos GTAs está dividida em três etapas: (i) transições com eventos não-controláveis; (ii) transições com eventos controláveis não-forçáveis; (iii) atualização dos cronômetros pelo evento *tick*; e (iv) transições com eventos forçáveis.

O trecho de código ilustrado na Figura 102, implementa em ST as transições dos eventos não-controláveis, para as máquinas de estados dos subsistemas G_1 e G_2. Pode-se observar que não é necessário implementar o tempo para os eventos b_1 e b_2, uma vez que as máquinas *M*1 e *M*2 devem terminar espontaneamente e de forma consistente, com os intervalos de tempo do GTA.

Da mesma forma, pode-se observar na primeira parte do código que implementa a máquina de estados para os eventos controláveis não-forçáveis (Figura 103), que a condição do tempo para o evento a_1 não é necessária também, uma vez que a máquina *M*1 deve iniciar espontaneamente e de forma consistente, com o intervalo de tempo do GTA. A segunda parte do código implementa a desabilitação do evento *a*1, uma vez que nessa aplicação foi inserido um indicador luminoso (verde/vermelho) na bancada, com o objetivo de informar ao operador em que momento é permitido o início da operação de *M*1. Assim, a implementação do tempo é necessária para habilitar o sinal de proibição (*Block_a*1), enquanto: o evento *a*1 não estiver elegível ($\tau_{a1} < \tau_{0la1}$); ou o sinal do evento *a*1 permanecer desabilitado (*De_a*1 >); ou estiver pausado pelo *Scada* (*Pause_a*1); ou, ainda, se o subsistema G_1^{act} não estiver no estado inicial (*p_G*1*_St* <> 0).

Figura 101 – Implementação do Supervisor Local Reduzido - Sr_1

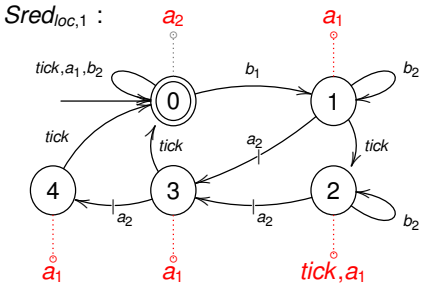
Máquina de Estados

```
IF (NOT e_blk_Sr1) AND Sr1_St=0 AND e_b1 THEN
    e_blk_Sr1 := 1; Sr1_St := 1;
END_IF;
IF (NOT e_blk_Sr1) AND Sr1_St=1 AND e_tick THEN
    e_blk_Sr1 := 1; Sr1_St := 2;
END_IF;
IF (NOT e_blk_Sr1) AND Sr1_St=1 AND e_a2 THEN
    e_blk_Sr1 := 1; Sr1_St := 3;
END_IF;
IF (NOT e_blk_Sr1) AND Sr1_St=2 AND e_a2 THEN
    e_blk_Sr1 := 1; Sr1_St := 3;
END_IF;
IF (NOT e_blk_Sr1) AND Sr1_St=3 AND e_tick THEN
    e_blk_Sr1 := 1; Sr1_St := 0;
END_IF;
IF (NOT e_blk_Sr1) AND Sr1_St=3 AND e_a2 THEN
    e_blk_Sr1 := 1; Sr1_St := 4;
END_IF;
IF (NOT e_blk_Sr1) AND Sr1_St=4 AND e_tick THEN
    e_blk_Sr1 := 1; Sr1_St := 0;
END_IF;
```

Desabilitações e Prrempção

```
(*e_a1:*) De_a1 := 0;
IF (Sr1_St=1 OR Sr1_St=2 OR Sr1_St=3 OR Sr1_St=4) THEN
    De_a1 := 1;
END_IF
(*e_a2:*) De_a2 := 0;
IF (Sr1_St=0) THEN
    De_a2 := 1;
END_IF

(*e_tick:*) Preempt_tick := 0;
IF (Sr1_St=2) THEN
    Preempt_tick := 1;
END_IF
```



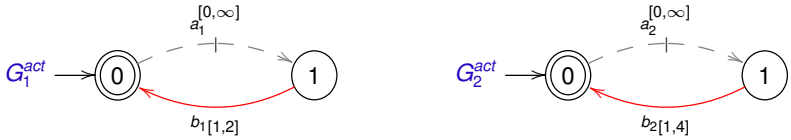
Fonte: Elaborado pelo autor (2021).

O trecho do código ilustrado na Figura 104 apresenta a implementação do evento *tick* gerado pelo RDG, tendo como função atualizar os cronômetros dos eventos controláveis, resetar e setar as variáveis *Ae_tick* e *e_tick*, respectivamente. Se nenhum evento não-controlável ou controlável não-forçável ocorreu na planta (no atual ciclo de varredura) e o evento *tick* for gerado pelo RDG, então todos os contadores ativos devem ser atualizados. Os cronômetros são implementados como contadores (CTU), onde seu valor é incrementado de forma síncrona com o relógio digital em uma unidade.

Figura 102 – Implementação transições eventos não-controláveis ($M1$ e $M2$)

Máquina de Estados (Σ_u)

```
IF (NOT e_blk_G) AND p_G1_St=1 AND Ae_b1 THEN
  Ae_b1 := 0;
  e_b1 := 1;
  p_G1_St:=0;
  e_blk_G := 1;
END_IF;
IF (NOT e_blk_G) AND p_G2_St=1 AND Ae_b2 THEN
  Ae_b2 := 0;
  e_b2 := 1;
  p_G2_St:=0;
  e_blk_G := 1;
END_IF;
```



Fonte: Elaborado pelo autor (2021).

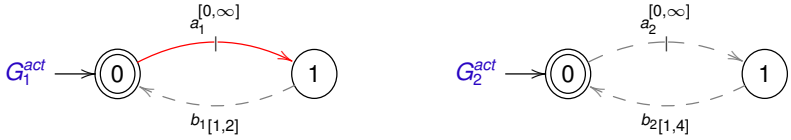
Figura 103 – Implementação transição evento controlável não-forçável ($M1$)

Máquina de Estados ($\Sigma_c \backslash \Sigma_{for}$)

```
IF (NOT e_blk_G) AND p_G1_St=0 AND Ae_a1 THEN
  Ae_a1 := 0;
  e_a1 := 1;
  p_G1_St:=1;
  e_blk_G := 1;
END_IF;
```

Desabilitação de evento controlável não-forçável

```
Block_a1 := De_a1 OR Pause_a1 OR (tau_a1 < tau_0la1 ) OR p_G1_St<>0;
```



Fonte: Elaborado pelo autor (2021).

Depois que todos os cronômetros estiverem atualizados, a variável *e_blk_G* deve ser setada, para impedir a atualização das máquinas de estados (dos eventos forçáveis) no SPT. No final da *Function Block* (FB) que implementa o SPT, atribui-se 0 a variável *Ae_tick*, e 1 a variável *e_tick* que será enviada ao SM.

A Figura 105 ilustra o código em ST associado a implementação da máquina de estados dos eventos forçáveis para o subsistema G_2. Para o evento forçável a_2, uma variável de tempo é necessária para verificar quando esse evento pode ocorrer, isto é, $\tau_{a2} \geq \tau_{0l_{a2}}$. O sinal de preempção do relógio digital deve ser implementado, para impedir que o início de operação de $M2$ possa ser desabilitado pelo sistema *SCADA*, enquanto o mesmo estiver ativo.

Cronômetros

Figura 104 – Implementação trecho código que atualiza os cronômetros, e seta e reseta variáveis *Ae_tick* e *e_tick*

```
IF ((NOT e_blk_G) AND Ae_tick AND (NOT Preemp_tick)) THEN

    (*Atualizacao cronometro - evento a1*)
    tau_0la1 := 0;
    CTU tau_a1 (CU:= p_G1_St=0, RESET:= e_a1, PV:= tau_0la1)
    tau_a1  := CTU_tau_a1.CV
    (*Atualizacao cronometro - evento a2*)
    tau_0la2 := 0;
    CTU tau_a2 (CU:= p_G2_St=0, RESET:= e_a2, PV:= tau_0la2)
    tau_a2 := CTU_tau_a2.CV

    Ae_tick := 0;
    e_tick := 1;
    e_blk_G := 1;
END_IF;
```

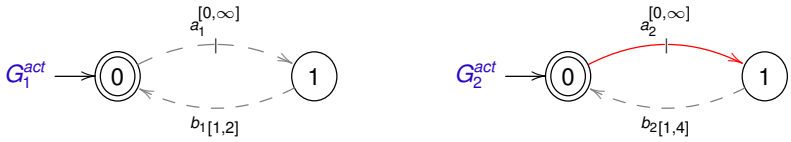
Fonte: Elaborado pelo autor (2021).

Figura 105 – Implementação transição evento forçável (*M*2)

Máquina de Estados (Σ_{for})

```
IF ((NOT e_blk_G) AND (p_G2_St=0) AND (NOT De_a2) AND (Preempt_tick or (NOT
    Pause_a2)) AND (tau_a2 >= tau0la2 )) THEN
    e_a2 := 1;
    p_G2_St:=1;
    e_blk_G := 1;
END_IF;

IF Preempt_tick AND (NOT evt_blk_G)  THEN
    error := 1;
END_IF;
```



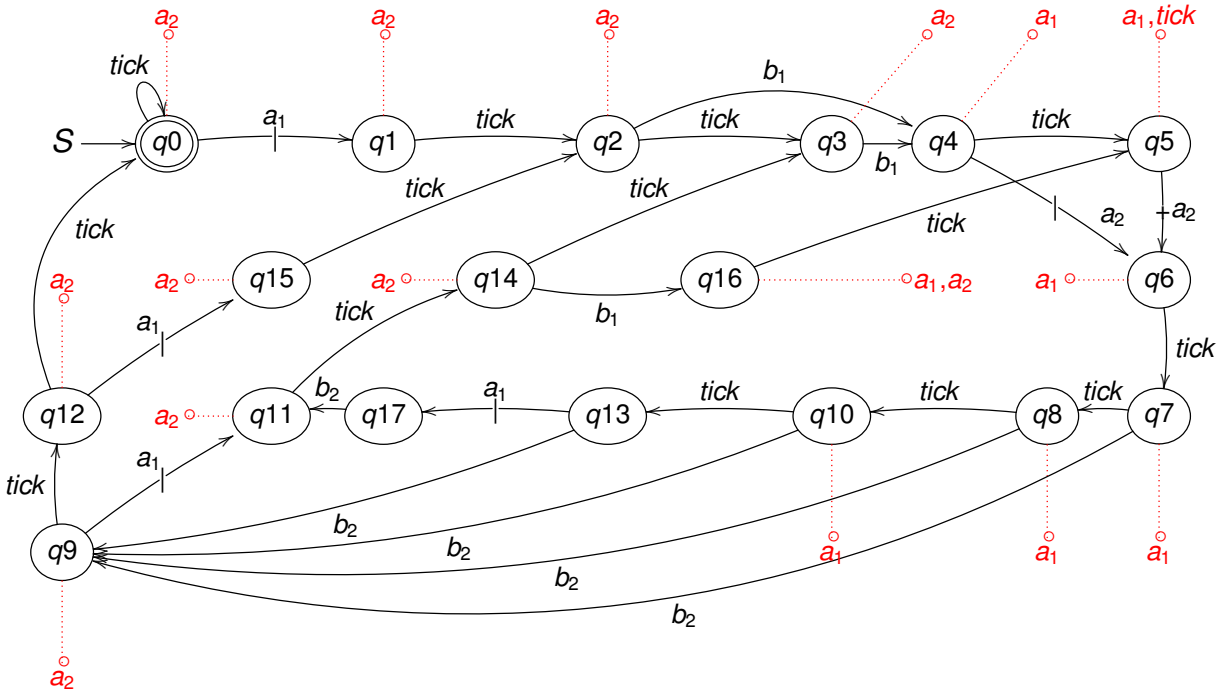
Fonte: Elaborado pelo autor (2021).

Importante salientar que, o trecho do código exposto nas Figuras 102; 103; 104 e; 105 são ordenadas no programa, conforme a ordem de prioridades definida na Figura 55, para o nível SPT.

Análise - Problemas de implementação: pode-se observar na Figura 106, que o supervisor monolítico resultante, obtido para este estudo de caso, não apresenta o problema de insensibilidade ao entrelaçamento, como por exemplo, no estado $q13$, a sequência de eventos $a1$-$b2$ ou $b2$-$a1$, leva o supervisor para o mesmo estado ($q14$). Pode-se observar, também, que no estado $q4$, conforme a ordem de prioridade definida na Seção 5.1.3, o *tick* será tratado primeiro, uma vez que não está sendo desabilitado e, que no estado $q5$ o *tick* será desabilitado e o evento $a2$ forçado. Já nos estados $q2$ e $q14$, como o *tick* tem menor prioridade que $b1$, se ambos acontecerem no mesmo

ciclo de varredura, então o evento *b*1 ocorrerá primeiro e o supervisor irá transitar para o estado *q*5, em que o *tick* será desabilitado e o evento *a*2 forçado. Como na estrutura proposta do SPT o evento *tick* tem a menor prioridade, quando ocorrer um evento não-controlável ou controlável não-forçável no mesmo ciclo, o relógio acabará atrasando por um período de *N scan*. Assim, é razoável assumir que esse pequeno atraso no período do relógio, esteja dentro da margem de tolerância do problema. Portanto, o supervisor satisfaz as condições para evitar os problemas analisados na Seção 5.2, logo, neste exemplo, não há problemas de implementação de escolha ou de ordem de eventos. Cabe ressaltar, que esses problemas de implementação da arquitetura temporizada em CLPs, também, não ocorrerão na implementação dos supervisores reduzidos.

Figura 106 – Supervisor monolítico (*S*)



Fonte: Elaborado pelo autor (2021).

7.5 RESULTADOS

Os experimentos a seguir têm como objetivo analisar o impacto da variação do período do relógio digital, no tempo de processamento das peças e no tamanho do código implementado em CLP. Vale ressaltar que, para períodos do *tick* maiores que 11s, o problema de controle não é solucionável, pois não é possível garantir a especificação *time lag*. O controle supervisório não foi implementado para períodos menores que 1s, porque o TTCT não permitiu calcular todo o conjunto de supervisores reduzidos.

Uma vez que o controle supervisório para o período de 5s foi programado no CLP S7-1200 e validado experimentalmente na bancada MPS, as demais soluções de controle supervisório indicadas na Tabela 15, também foram programadas. Para isso, foi apenas necessário substituir os supervisores do SM, e atualizar o RDG e os contadores do SPT de acordo com o período de relógio correspondente. O número de linhas de código para as diferentes implementações são apresentados na Tabela 16.

Por intermédio de valores obtidos experimentalmente para duas sequências de peças diferentes, pode-se verificar na Tabela 16, que a eficiência do sistema melhora quando a granularidade do *tick* aumenta. Reduzindo o período do relógio de 5s para 1s, o tempo de processamento de 4 peças consecutivas reduz de 80s para 56s (30%), e o tempo de processamento de 8 peças consecutivas reduz de 143s para 118s (17%), enquanto o número de linhas de código aumenta de 90 para 149 (66%).

Pode-se observar, também, um pequeno aumento no número de linhas de código, devido ao refinamento dos valores do *tick*, uma vez que as plantas locais são implementadas como GTAs, o que não varia o número de estados, já que a variável de tempo é implementada por meio de contadores. Cabe ressaltar, que essa estimativa foi apoiada na implementação do exemplo motivador, em que cada transição gerou, em média, três linhas de código para os supervisores reduzidos e quatro para as plantas locais.

Tabela 16 – Tempo total de processamento do sistema x Tamanho código CLP

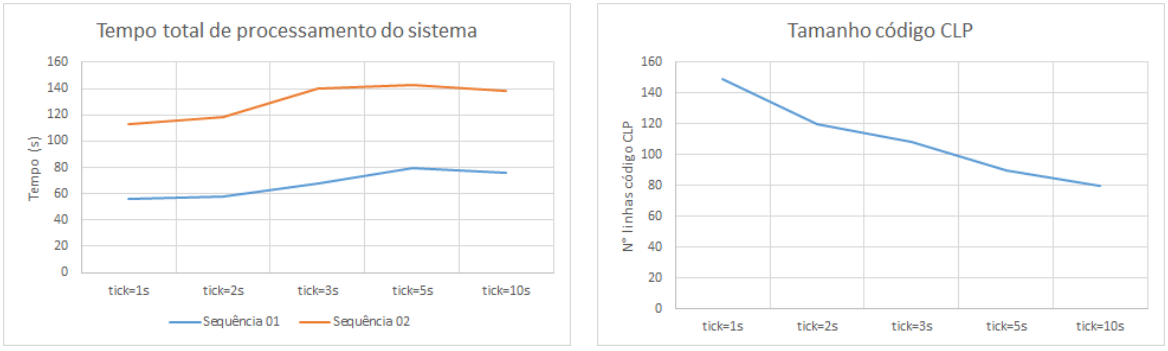
Período do Relógio Digital (valor do *tick*)	10s	5s	3s	2s	1s
Sequência 01 (4 peças)	76s	80s	68s	58s	56s
Sequência 02 (8 peças)	138s	143s	140s	118s	113s
Nº Linhas de Código CLP	80	90	108	120	149

Fonte: Elaborado pelo autor (2021).

A Figura 107 ilustra o tempo de processamento total do sistema em comparação com o tamanho do código do CLP, em função do refinamento do período do relógio digital global.

Para os autômatos usados na síntese formal, pode-se observar um pequeno aumento no número de estados, devido ao refinamento dos valores do *tick*. Consequentemente, o código de implementação não apresenta uma grande variação, como pode ser verificado na Tabela 4. Uma das razões para isso é que as plantas locais são implementadas como GTAs, que não variam o número de estados, uma vez que a variável de tempo é implementada por meio de temporizadores.

Figura 107 – Tempo total de processamento do sistema x Tamanho código CLP



(a) Tempo total de processamento (b) Tamanho código CLP

Fonte: Elaborado pelo autor (2021).

7.6 CONCLUSÃO DO CAPÍTULO

Uma aplicação bem-sucedida da teoria de controle de supervisório temporizada em uma bancada didática do mundo real foi apresentada. A arquitetura de implementação proposta permite estruturar o código do CLP de forma que a explosão do espaço de estados seja evitada. Os resultados experimentais indicam que os supervisores Ramage-Wonham são adequados para garantir um comportamento correto, não bloqueante e seguro para sistemas de manufatura inteligentes, respeitando as incertezas no tempo de operação das máquinas e operadores humanos.

Com a implementação do sistema de controle modular temporizado, o sistema se torna mais produtivo e flexível. A lógica de implementação permite que o sistema trabalhe com até três peças em paralelo e as máquinas trabalhem ao mesmo tempo. A estrutura do código torna-se flexível e compreensível, o que facilita alterações. Assim, a proposta de uma estrutura de controle temporizada apresentada e implementada, é satisfatória e atende aos objetivos iniciais propostos.

A partir dos experimentos da aplicação proposta, foi possível realizar uma análise sobre o impacto da variação do relógio digital no tempo de processamento das peças e no tamanho do código implementado no CLP. Assim, verificou-se que a eficiência e o número de linhas de código aumentam em função do refinamento da granularidade do *tick*. Pôde-se observar, também que, uma vez que as plantas locais são implementadas como GTAs, o número de estados não varia com o refinamento do *tick*, pois a variação de tempo é implementada por meio de contadores, gerando modelos mais compactos e, consequentemente, gerando menos memória do que se implementadas como GTTs. E, por fim, observa-se um pequeno aumento no código final implementado no CLP, devido ao refinamento do *tick*, em razão das plantas locais serem implementadas como GTAs.

8 CONSIDERAÇÕES FINAIS

Esta Tese de Doutorado teve como objetivo geral, o desenvolvimento de contribuições que fundamentem a implementação do Controle Supervisório Modular Local Temporizado em Controladores Lógicos Programáveis. Para dar conta do objetivo proposto, uma nova arquitetura de controle modular local temporizada foi apresentada, em conjunto com um método para a modelagem (discretização dos intervalos de tempos contínuos) de sistemas de manufatura com tempos de processamentos incertos e restrições de tempo.

Os resultados apresentados constituem uma proposta viável de uma arquitetura temporizada que permite a implementação do CMLT em CLPs e uma proposta para o cálculo dos intervalos de tempos discretos (planta e especificações), a partir dos intervalos de tempos contínuos obtidos experimentalmente.

Uma pequena contribuição desta pesquisa, a partir de uma revisão dos principais conceitos TCS com enfoque na abordagem modular local, foi a realização da modelagem e aplicação em um problema real, com o tratamento de falhas para eventos observáveis e não controláveis. Os resultados obtidos demonstraram que a metodologia mostrou-se eficiente, tornando a solução do projeto, segura, flexível e minimamente restritiva, além de permitir a estruturação do programa do CLP e uma maior sistematização para a integração do sistema SCADA. Com essa aplicação bem-sucedida, apresentou-se um artigo no congresso *Fluid Power Net International* (SZPAK; QUEIROZ, 2016).

Com base em uma revisão dos principais conceitos da TCS de SEDT, com enfoque principal na abordagem modular local temporizada, apresentou-se um problema real de controle para a aplicação dos conceitos e metodologias, que se mostrou satisfatório para a síntese dos supervisores modulares locais temporizados. A partir desses modelos e da arquitetura temporizada proposta, outra contribuição foi obtida com a implementação dos supervisores temporizados no sistema real para a Estação MPS Festo de Separação que se comportou, de uma maneira geral, como esperado e, portanto, pode ser utilizada para a implementação da lógica do CMLT. Assim, o código resultante é estruturado, de fácil interpretação e implementável na memória do CLP.

Cabe destacar que a implementação das plantas locais como GTAs resultou em economia de código gerado na implementação do problema, uma vez que o número de estados da planta não aumenta, em função do refinamento do período do relógio digital. Assim, a arquitetura proposta é bastante simples, permitiu estruturar o código do CLP de forma que a explosão do espaço de estados fosse evitada, atende às características dos eventos apresentados, e é suficientemente geral para lidar com uma ampla classe de problemas reais. Esses conceitos aplicados para a resolução de um

problema real foram utilizados para a construção de um artigo, o qual foi apresentado no Congresso Internacional *Workshop on Discrete-Event Systems (WODES)* (SZPAK; QUEIROZ; CURY, 2020). Uma versão expandida desse trabalho está sendo submetida para publicação em periódico internacional.

Uma análise da granularidade do período do relógio digital verificou o quanto esse índice afeta o desempenho do sistema, e que, o que se perde em conservadorismo, se ganha em complexidade dos modelos.

Como contribuições futuras, vislumbra-se a elaboração de um método sistematizado para a geração de códigos da implementação do controle temporizado em CLPs. Outra perspectiva para estudos futuros é estender a proposta para trabalhar com mais de um evento de varredura no CLP, além de realizar um estudo aprofundado sobre o uso de escalonadores para resolver os problemas das especificações temporizadas.

REFERÊNCIAS

AFZALIAN, A. A.; NOORBAKHSH, S.; WONHAM, W. Discrete-event supervisory control for under-load tap-changing transformers (ULTC): from synthesis to PLC implementation. *Discrete Event Simulations*, BoD–Books on Demand, p. 285–310, 2010.

AKESSON, K. *Methods and Tools in Supervisory Control Theory: Operator Aspects, Computational Efficiency, and Applications*. Göteborg: Institutionen för signaler och system, Chalmers tekniska högskola, 2002. (Technical report - School of Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden, no: 43). 200. ISBN 91-7291-200-6.

ANDERSON, S.; TOURLAS, K. Diagrams and programming languages for programmable controllers. In: *Proceedings of the 4th International Symposium of Formal Methods Europe on Industrial Applications and Strengthened Foundations of Formal Methods*. London, UK: Springer-Verlag, 1997. (FME '97), p. 1–19. ISBN 3-540-63533-5.

BALEMI, S. *Control of discrete event systems: theory and application*. Tese (Doutorado) — Automatic Control Laboratory, Swiss Federal Insitute of Technology, Zürich, Switzerland, 1992.

BALEMI, S. Input/output discrete event processes and communication delays. *Discrete Event Dynamic Systems*, Springer, v. 4, n. 1, p. 41–85, 1994.

BASILE, F.; CHIACCHIO, P. On the implementation of supervised control of discrete event systems. *IEEE Transactions on Control Systems Technology*, IEEE, v. 15, n. 4, p. 725–739, 2007.

BASILE, F.; CHIACCHIO, P.; VITTORINI, V.; MAZZOCCA, N. Modeling and logic controller specification of flexible manufacturing systems using behavioral traces and Petri Net building blocks. *Journal of Intelligent Manufacturing*, Springer, v. 15, n. 3, p. 351–371, 2004.

BRANDIN, B.; WONHAM, W. Modular supervisory control of timed discrete-event systems. In: IEEE. *Proceedings of 32nd IEEE Conference on Decision and Control*. [S.l.], 1993. p. 2230–2235.

BRANDIN, B. A.; WONHAM, W. M. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, IEEE, v. 39, n. 2, p. 329–342, 1994.

CAI, K.; WONHAM, W. M. Supervisor localization for large discrete-event systems. *The International Journal of Advanced Manufacturing Technology*, v. 50, n. 9, p. 1189–1202, 2010. ISSN 1433-3015.

CANTARELLI, M.; ROUSSEL, J.-M. Reactive control system design using the supervisory control theory: Evaluation of possibilities and limits. In: *In Proc. 9th Int. WODES*. [S.l.: s.n.], 2008. p. 200 – 205.

CHEN, Y.-L.; LIN, F. Modeling of discrete event systems using finite state machines with parameters. In: IEEE. *Control Applications, 2000. Proceedings of the 2000 IEEE International Conference on*. Anchorage, Alaska, USA, 2000. p. 941–946.

CONSTAIN, N. B. P. *Integração de sistemas SCADA com a implementação de controle supervisório em CLP para sistemas de manufatura*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina - UFSC, 2011.

CRUZ, D. L. L. d. Metodologia para implementação de controle supervisório modular local em controladores lógicos programáveis. Dissertação (Mestrado) - Universidade do Estado de Santa Catarina, 2011.

FABIAN, M.; HELLGREN, A. PLC-based implementation of supervisory control for discrete event systems. In: IEEE. *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*. Tampa, Florida, USA, 1998. v. 3, p. 3305–3310.

FARINES, J. M.; QUEIROZ, M. H. de; ROCHA, V. G. da; CARPES, A. M. M.; VERNADAT, F.; CRéGUT, X. A model-driven engineering approach to formal verification of PLC programs. In: *Proc of IEEE 16th conference on emerging technologies Factory automation*. Toulouse, France: ETFA, 2011. p. 1–8. ISSN 1946-0740.

FENG, L.; WONHAM, W. Tct: A computation tool for supervisory control synthesis. In: IEEE. *8th International Workshop on Discrete Event Systems*. Ann Arbor, MI, USA, 2006. p. 388–389.

GERGELY, E. I.; COROIU, L.; POPENTIU-VLADICESCU, F. Methods for validation of PLC systems. *Journal of Computer Science and Control Systems*, University of Oradea, v. 4, n. 1, p. 47, 2011.

GOHARI, P.; WONHAM, W. M. On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE, v. 30, n. 5, p. 643–652, 2000.

HASDEMIR, İ. T.; KURTULAN, S.; GÖREN, L. An implementation methodology for supervisory control theory. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 36, n. 3-4, p. 373–385, 2008.

HELLGREN, A.; FABIAN, M.; LENNARTSON, B. Modular implementation of discrete event systems as sequential function charts applied to an assembly cell. In: IEEE. *Control Applications, 2001.(CCA'01). Proceedings of the 2001 IEEE International Conference on*. Mexico City, 2001. p. 453–458.

HELLGREN, A.; FABIAN, M.; LENNARTSON, B. On the execution of sequential function charts. *Control Engineering Practice*, Elsevier, v. 13, n. 10, p. 1283–1293, 2005.

HELLGREN, A.; LENNARTSON, B.; FABIAN, M. Modelling and PLC-based implementation of modular supervisory control. In: IEEE. *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*. Zaragoza, Spain, 2002. p. 371–376.

HOFFMANN, G.; WONG-TOI, H. Symbolic synthesis of supervisory controllers. In: ACC'92. *American Control Conference, 1992*. Chigaco, IL, USA, 1992. p. 2789–2793.

HOLLOWAY, L. E.; KROGH, B. H. Synthesis of feedback control logic for a class of controlled petri nets. *IEEE Transactions on Automatic Control*, IEEE, v. 35, n. 5, p. 514–523, 1990.

HOLLOWAY, L. E.; KROGH, B. H.; GIUA, A. A survey of petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems*, Springer, v. 7, n. 2, p. 151–190, 1997.

JIAO, T.; GAN, Y. On plc implementation of decentralized and hierarchical supervisory control of discrete-event system. In: IEEE. *2013 UKSim 15th International Conference on Computer Modelling and Simulation*. [S.l.], 2013. p. 413–418.

JIMENEZ, I.; LOPEZ, E.; RAMIREZ, A. Synthesis of ladder diagrams from petri nets controller models. In: IEEE. *Proceeding of the 2001 IEEE International Symposium on Intelligent Control (ISIC'01)(Cat. No. 01CH37206)*. [S.l.], 2001. p. 225–230.

KLINGE, S. *Supervisory control of a manufacturing cell: modeling and implementation*. 92 p. Tese (Doutorado) — Fakultät für Elektrotechnik und Informationstechnik, Otto-von-Guericke-Universität Magdeburg, 2007.

LAMNABHI-LAGARRIGUE, F.; ANNASWAMY, A.; ENGELL, S.; ISAKSSON, A.; KHARGONEKAR, P.; MURRAY, R. M.; NIJMEIJER, H.; SAMAD, T.; TILBURY, D.; HOF, P. V. den. Systems control for the future of humanity, research agenda: Current and future roles, impact and grand challenges. *Annual Reviews in Control*, v. 43, p. 1 – 64, 2017. ISSN 1367-5788.

LEAL, A.; CRUZ, D. da; HOUNSELL, M. da S. *PLC-Based Implementation of Local Modular Supervisory Control for Manufacturing Systems*. Manufacturing System. 1ed: Intech, 2012.

LEAL, A. B.; CRUZ, D. L. L. da; HOUNSELL, M. da S. Supervisory control implementation into programmable logic controllers. In: *Proceedings of the 14th IEEE International Conference on Emerging Technologies and Factory Automation*. Palma de Mallorca, Spain: ETFA, 2009. p. 899–905.

LEDUC, R. J. *PLC implementation of a DES supervisor for a manufacturing testbed: an implementation perspective.* [S.l.]: University of Toronto, 1996.

LEDUC, R. J.; WANG, Y.; AHMED, F. Sampled-data supervisory control. *Discrete Event Dynamic Systems*, v. 24, n. 4, p. 541–579, 2014. ISSN 1573-7594.

LEE, G. B.; ZANDONG, H.; LEE, J. S. Automatic generation of ladder diagram with control petri net. *Journal of Intelligent Manufacturing*, Springer, v. 15, n. 2, p. 245–252, 2004.

LJUNGKRANTZ, O.; ÅKESSON, K.; FABIAN, M.; YUAN, C. A formal specification language for PLC-based control logic. In: *8th IEEE International Conference on Industrial Informatics*. .: INDIN, 2010. p. 1067–1072. ISSN 1067-1072.

MALIK, P. Generating controllers from discrete-event models. Proceedings of the Summer school in Modelling and Verification of Parallel processes (MOVEP), p. 1–6, 2002.

MIRZAEI, F.; POUYAN, A. A.; BIGLARI, M. Automatic controller code generation for swarm robotics using probabilistic timed supervisory control theory (PTSCT). *Journal of Intelligent & Robotic Systems*, Springer, v. 100, n. 2, p. 729–750, 2020.

MOKADEM, H. B.; BERARD, B.; GOURCUFF, V.; SMET, O. D.; ROUSSEL, J. M. Verification of a timed multitask system with uppaal. *IEEE Transactions on Automation Science and Engineering*, v. 7, n. 4, p. 921–932, Oct 2010. ISSN 1545-5955.

MONIRUZZAMAN, M.; GOHARI, P. Implementing supervisory control maps with PLC. In: IEEE. *2007 American Control Conference*. [S.l.], 2007. p. 3594–3599.

MOODY, J.; ANTSAKLIS, P. *Supervisory Control of Discrete Event Systems Using Petri Nets*. [S.l.]: Springer US, 1998. (The International Series on Discrete Event Dynamic Systems). ISBN 9780792381990.

MOREIRA, M. V.; BASILIO, J. C. Bridging the gap between design and implementation of discrete-event controllers. *IEEE Transactions on Automation Science and Engineering*, IEEE, v. 11, n. 1, p. 48–65, 2013.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, IEEE, v. 77, n. 4, p. 541–580, 1989.

MUŜIĈ, G.; MATKO, D. Combined synthesis/verification approach to programmable logic control of a production line. *IFAC Proceedings Volumes*, Elsevier, v. 38, n. 1, p. 98–103, 2005.

PINOTTI, A. J. Um método para projeto de sistemas embarcados baseado no controle supervisório modular local. Dissertação (Mestrado) Universidade do Estado de Santa Catarina, 2012.

PORTILLA, N. B.; QUEIROZ, M. H. de; CURY, J. E. Integration of supervisory control with scada system for a flexible manufacturing cell. In: IEEE. *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. [S.l.], 2014. p. 261–266.

PRENZEL, L.; PROVOST, J. PLC implementation of symbolic, modular supervisory controllers. *IFAC-PapersOnLine*, Elsevier, v. 51, n. 7, p. 304–309, 2018.

QUEIROZ, M. D. *Controle supervisório modular e multitarefa de sistemas compostos*. Tese (Doutorado) — Universidade Federal de Santa Catarina - UFSC, 2004.

QUEIROZ, M. H. D.; CURY, J. E. Modular supervisory control of large scale discrete event systems. In: *In Discrete Event Systems: Analysis and Control. Proc. WODES'00*. Massachusetts: Kluwer Academic, 2000. p. 103–110.

QUEIROZ, M. H. D.; CURY, J. E. R. Synthesis and implementation of local modular supervisory control for a manufacturing cell. In: IEEE. *Proceedings of 6th Int. Workshop on Discrete Event Systems*. Zaragoza, Spain, 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. G.; WONHAM, W. M. The control of discrete event systems. *Proceedings of the IEEE*, v. 77, n. 1, p. 81–98, Jan 1989. ISSN 0018-9219.

ROUSSEL, J.; GIUA, A. Designing dependable logic controllers using the supervisory control theory. *IFAC Proceedings Volumes*, v. 38, p. 56–61, 2005.

RUDIE, K. The integrated discrete-event systems tool. In: *2006 8th International Workshop on Discrete Event Systems*. Ann Arbor, MI, USA: IEEE, 2006. p. 394–395.

SAADATPOOR, A.; MA, C.; WONHAM, W. Supervisory control of timed state tree structures. In: IEEE. *2008 American Control Conference*. [S.l.], 2008. p. 477–482.

SCHAFASCHEK, G. *Uma abordagem local para o controle supervisório modular de sistemas a eventos discretos temporizados*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina - UFSC, 2014.

SCHAFASCHEK, G.; QUEIROZ, M. H. de; CURY, J. E. Local modular supervisory control of timed discrete-event systems. *IFAC Proceedings Volumes*, Elsevier, v. 47, n. 2, p. 271–277, 2014.

SCHAFASCHEK, G.; QUEIROZ, M. H. de; CURY, J. E. R. Local modular supervisory control applied to the scheduling of cluster tools. In: *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. Gothenburg, Sweden: IEEE, 2015. p. 1381–1388.

SCHAFASCHEK, G.; QUEIROZ, M. H. de; CURY, J. E. R. Local modular supervisory control of timed discrete-event systems. *IEEE Trans. Automat. Contr.*, v. 62, n. 2, p. pp. 934–940, 2017.

SCOTTI, W. A. F. *Arquitetura de sistema de controle supervisório integrando CLP, SCADA e roteamento de tarefas*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina - UFSC, 2015.

SILVA, M. E. *Apostila de Automação Industrial*. Piracicaba: Piracicaba, 2007.

SKOLDSTAM, M.; AKESSON, K.; FABIAN, M. Modeling of discrete event systems using finite automata with variables. In: IEEE. *2007 46th IEEE Conference on Decision and Control*. [S.l.], 2007. p. 3387–3392.

SU, R.; WONHAM, W. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems*, v. 14, n. 1, p. 31–53, 2004. ISSN 1573-7594.

SUESUT, T.; INBAN, P.; NILAS, P.; RERNGREUN, P.; GULPHANICH, S. Interpretation petri net model to iec 1131-3: Ld for programmable logic controller. In: IEEE. *IEEE Conference on Robotics, Automation and Mechatronics, 2004.* [S.l.], 2004. v. 2, p. 1107–1111.

SZPAK, R.; QUEIROZ, M. H. D. Design and implementation of supervisory control for an electropneumatic station subject to faults in material flow. In: . Florianópolis, SC: ASME. Fluid Power Systems Technology, 9th FPNI Ph.D. Symposium on Fluid Power. http://dx.doi.org/10.1115/FPNI2016-1542., 2016.

SZPAK, R.; QUEIROZ, M. H. de; CURY, J. E. R. Synthesis and implementation of supervisory control for manufacturing systems under processing uncertainties and time constraints. *IFAC-PapersOnLine*, Elsevier, v. 53, n. 4, p. 229–234, 2020.

TEXEIRA, C. A.; LEAL, A. B. Desenvolvimento de controles eletrônicos para refrigeradores: Uma abordagem baseada na teoria de controle supervisório. In: *XVII Congresso Brasileiro de Automática*. [S.l.: s.n.], 2008.

THÉVENOD-FOSSE, P.; WAESELYNCK, H.; CROUZET, Y. Software statistical testing. In: ____. *Predictably Dependable Computing Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995. p. 253–272. ISBN 978-3-642-79789-7.

UZAM, M. A general technique for the PLC-based implementation of rw supervisors with time delay functions. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 62, n. 5, p. 687–704, 2012.

UZAM, M.; JONES, A. Discrete event control system design using automation petri nets and their ladder diagram implementation. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 14, n. 10, p. 716–728, 1998.

VIEIRA, A. D. *Método de implementação do controle de sistemas e eventos discretos com aplicação da teoria de controle supervisório*. Tese (Doutorado) — Universidade Federal de Santa Catarina - UFSC, 2007.

Vieira, A. D.; Santos, E. A. P.; de Queiroz, M. H.; Leal, A. B.; de Paula Neto, A. D.; Cury, J. E. R. A method for plc implementation of supervisory control of discrete event systems. *IEEE Transactions on Control Systems Technology*, v. 25, n. 1, p. pp. 175–191, Jan 2017.

VIEIRA, A. D.; SANTOS, E. A. P.; QUEIROZ, M. H. de; LEAL, A. B.; NETO, A. D. de P.; CURY, J. E. A method for PLC implementation of supervisory control of discrete event systems. *IEEE Transactions on Control Systems Technology*, IEEE, v. 25, n. 1, p. 175–191, 2017.

WANG, G.-C.; LI, W. Expected schedulability based on stochastic model for tasks in soft real-time system. *Kongzhi Lilun Yu Yingyong/Control Theory and Applications*, v. 29, n. 1, p. 130–134, 2012. Cited By 0.

WONG, K. C.; WONHAM, W. M. Modular control and coordination of discrete-event systems. *Discrete Event Dynamic Systems*, Springer, v. 8, n. 3, p. 247–297, 1998.

WONHAM, W. M. *Supervisory control of discrete-event systems. Lecture notes. University of Toronto [Online]*. at: http://www.control.utoronto.ca/DES/.: [s.n.], 2011.

WONHAM, W. M.; CAI, K. *Supervisory control of discrete-event systems*. [S.l.]: Springer, 2019.

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.

WUJIE, C.; YONGMEI, G.; ZHAO'AN, W.; WONHAM, W. An optimal nonblocking modular supervisory control to real-time state tree structures. *Journal of Xi'an Jiaotong University*, p. 04, 2013.

ZAYTOON, J.; RIERA, B. Synthesis and implementation of logic controllers – a review. *Annual Reviews in Control*, , n. , p. , 2017. ISSN 1367-5788.

ZHANG, R.; CAI, K.; GAN, Y.; WANG, Z.; WONHAM, W. M. Supervision localization of timed discrete-event systems. *Automatica*, Elsevier, v. 49, n. 9, p. 2786–2794, 2013.

ZHANG, Z.; WONHAM, W. M. STCT: An efficient algorithm for supervisory control design. In: *Synthesis and control of discrete event systems*. [S.l.]: Springer, 2002. p. 77–100.

ZHOU, M.; DICESARE, F. *Petri net synthesis for discrete event control of manufacturing systems*. [S.l.]: Springer Science & Business Media, 2012. v. 204.