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

MARCELO ROSA

EXPLORANDO O USO DE ABSTRAÇÕES NA IMPLEMENTAÇÃO DE CONTROLADORES PARA SISTEMAS A EVENTOS DISCRETOS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2016

MARCELO ROSA

EXPLORANDO O USO DE ABSTRAÇÕES NA IMPLEMENTAÇÃO DE CONTROLADORES PARA SISTEMAS A EVENTOS DISCRETOS

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

Orientador: Prof. Dr. Marcelo Teixeira

Coorientador: Prof. Dr. Gustavo Weber Denardin

PATO BRANCO

2016

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

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

Às 8 horas e 30 minutos do dia 08 de dezembro de 2016, na sala V103, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, reuniu-se a banca examinadora composta pelos professores Marcelo Teixeira (orientador), Gustavo Weber Denardin (coorientador), Cesar Rafael Claure Torrico e Marco Antonio de Castro Barbosa para avaliar o trabalho de conclusão de curso com o título **Explorando o uso de abstrações na implementação de controladores para sistemas a eventos discretos**, do aluno **Marcelo Rosa**, matrícula 1436244, do curso de Engenharia de Computação. Após a apresentação o candidato foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____	_____
Marcelo Teixeira Orientador (UTFPR)	Gustavo Weber Denardin Coorientador (UTFPR)
_____	_____
Cesar Rafael Claure Torrico (UTFPR)	Marco Antonio de Castro Barbosa (UTFPR)
_____	_____
Beatriz Terezinha Borsoi Coordenador de TCC	Pablo Gauterio Cavalcanti Coordenador do Curso de Engenharia de Computação

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

RESUMO

ROSA, Marcelo. Explorando o uso de abstrações na implementação de controladores para sistemas a eventos discretos. 2016. 57 f. Trabalho de Conclusão de Curso 2 - Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2016.

Na *Teoria de Controle Supervisório* (TCS) de *Sistemas a Eventos Discretos*, o refinamento de eventos tem sido explorado para simplificar a tarefa de modelagem. As *Aproximações* complementam essa abordagem como uma alternativa para reduzir o esforço na síntese. Apesar das vantagens na etapa de modelagem e síntese, refinamentos não cobrem diretamente a fase de implementação. Em geral, controladores obtidos com ou sem refinamentos são implementados com o mesmo custo de hardware. Este trabalho propõe uma arquitetura descentralizada que estende os ganhos provenientes dos refinamentos da síntese para implementação. A abordagem proposta separa supervisor e distinguidor em duas estruturas distintas, as quais se comunicam de tal forma que o resultado da ação de controle sobre a planta é equivalente a versão centralizada, porém com custo de implementação reduzido. Além disso, é apresentada uma alternativa para implementar o comportamento do distinguidor utilizando um modelo genérico composto por dois estados, o qual espera-se estender para qualquer distinguidor.

Palavras-chave: Sistema a eventos discretos, Controle Supervisório, Distinguidores, Implementação.

ABSTRACT

ROSA, Marcelo. Exploring the use of abstractions in the implementation of controllers for discrete event systems. 2016. 57 f. Trabalho de Conclusão de Curso 2 - Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco,

In the Supervisory Control Theory (SCT) of Discrete Event Systems, the refinement of events has been explored to simplify the modeling task. Approximations complement this approach as an alternative to reduce effort in synthesis. In spite of the advantages in the modeling and synthesis step, refinements do not directly cover the implementation phase. In general, controllers obtained with or without refinements are implemented with the same hardware cost. This paper proposes a decentralized architecture that extends the gains from the refinements of the synthesis to implementation. The proposed approach separates supervisor and distinguisher in two distinct structures, which communicate in such a way that the result of the control action on the plant is equivalent to the centralized version, but with reduced implementation cost. In addition, an alternative to implement the distinguisher behavior is presented using a generic two-state model, which is expected to extend to any distinguisher.

Keywords: Discrete-Event Systems, Supervisory Control, Distinguishers, Implementation.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este de Trabalho de Conclusão de Curso (TCC) explora um aspecto pertinente em projetos de automação a eventos discretos, que é a complexidade enfrentada na etapa de síntese e implementação de um controlador. A literatura vem abordando vários aspectos relacionados à simplificação do processo de síntese e, neste trabalho, o objetivo é propagar esses métodos de simplificação também para as etapas de implementação.

Este capítulo apresentará uma visão geral sobre o assunto e a contextualização em termos de pesquisas e estudos na área, bem como a inserção deste trabalho no assunto. Em seguida, são apresentados os objetivos e a justificativa da pesquisa

1.1 CONSIDERAÇÕES INICIAIS

Dentre as várias metodologias existentes para a obtenção de lógicas de controle para *Sistemas a Eventos Discretos* (SEDs) (CASSANDRAS; LAFORTUNE, 2008), a *Teoria do Controle Supervisório* (TCS), proposta por Ramadge e Wonham (1989), ganha destaque. Sua estrutura fundamental é baseada na *teoria de Autômatos Finitos* (AFs) e *Linguagens* e sua essência é fornecer um mecanismo formal capaz de descrever a síntese de controladores ótimos, ou seja, controladores que possuem a propriedade de atuar sobre o modelo de um SED de forma minimamente restritiva, não-bloqueante e em consonância com um conjunto de especificações.

Na TCS, a planta do sistema é modelada por um conjunto de AFs, cuja composição resulta no modelo do comportamento do sistema sem qualquer intervenção externa ou ação de controle. Esse modelo é denominado como planta em *malha aberta*.

De maneira análoga, os requisitos do sistema são modelados por

um conjunto de AFs, cuja composição é denominada *especificação*. Da associação entre planta e especificação emerge um modelo que representa o comportamento desejado para o sistema sob controle, isto é, o sistema em *malha fechada*.

Em alguns casos, o comportamento que se espera do sistema sob controle pode não ser consistente com o comportamento que pode ser controlado na prática em um SED. Isso porque alguns eventos, como o final de operação de um equipamento pode ser proveniente de um impulso sobre o qual não se tem controle, por exemplo, uma queda de energia. Se o controlador não é capaz de distinguir a natureza da ocorrência de tal evento, então ele pode mascarar uma situação de controlabilidade que na verdade é inconsistente em relação ao sistema real. Nesse sentido, a TCS define o particionamento do conjunto de eventos de um SED conforme a natureza como ocorrem na planta. Esse particionamento permite que se calcule, do comportamento esperado, o subcomportamento que mais se aproxima do esperado e, ao mesmo tempo, que respeita o conjunto de especificações. Se esse subcomportamento for ainda não bloqueante, então ele é dito ser ótimo.

A entidade da TCS responsável por aplicar a ação de controle sobre a planta, de tal forma que implementa o comportamento controlável do sistema, é denominada *supervisor*. O supervisor observa os eventos possíveis na planta e define quais devem ser, de fato, habilitados. Desse modo, os eventos possíveis na planta e que não pertence ao conjunto de eventos habilitados do supervisor, são inibidos pela ação de controle.

Apesar das vantagens da TCS no processo de obtenção de controladores ótimos, alguns aspectos limitam a aplicação da TCS em escala industrial. Dois desses aspectos serão abordados neste trabalho e estão intimamente relacionados ao conceito de informação. O grau de informação disponível no modelo de um SED, quando elevado (TEIXEIRA *et al.*, 2014; CURY *et al.*, 2015), interfere diretamente na complexidade computacional

envolvida no processo de síntese do supervisor enquanto que, por outro lado, um baixo grau de informação (CUNHA; CURY, 2007) tende a aumentar a dificuldade enfrentada na tarefa de modelar as especificações a serem cumpridas pelo sistema em malha fechada (CURY *et al.*, 2015).

Esses aspectos fomentam extensões da TCS. Uma delas é baseada no refinamento de informações sobre um conjunto de eventos que pertencem ao sistema. Implicitamente, dispor de mais informações acerca de um mesmo evento tende a simplificar a tarefa de modelagem (TEIXEIRA, 2013; TEIXEIRA *et al.*, 2014; CURY *et al.*, 2015).

No entanto, para utilizar esse conjunto de eventos refinados, é necessário agregar ao sistema refinado uma estrutura extra denominada distinguidor, proposto inicialmente por Bouzon *et al.* (2008). O distinguidor é uma entidade capaz de mapear cada cadeia de eventos originais em cadeias compostas por eventos refinados, e dar a elas uma semântica particular (TEIXEIRA, 2013).

A literatura (CURY *et al.*, 2015) demonstra que o uso de distinguidores permite modelar especificações complexas de controle, de uma forma mais simples e, ainda assim, levar a uma solução de controle equivalente a solução do *Problema de Controle Supervisório* (PCS) original. Mas, mostrou-se que somente o uso do distinguidor não gera diretamente vantagens computacionais na síntese, uma vez que os ganhos obtidos com a simplificação das especificações são compensados com a adição do modelo do distinguidor à planta.

Indiretamente, contudo, é possível simplificar também o processo de síntese ao utilizar uma aproximação para o modelo distinguido (TEIXEIRA, 2013). O conceito de aproximação se utiliza da ideia de que, em certas condições, o modelo do distinguidor pode ser removido momentaneamente da síntese do controlador, provendo assim ganhos computacionais, e devolvido ao modelo do controlador para fins de implementação (TEIXEIRA, 2013). É imaginável,

então, que possíveis economias computacionais se limitem ao procedimento de síntese, apenas.

1.2 OBJETIVOS

Os objetivos apresentados nessa proposta se dividem em objetivo geral que representa o resultado final desse trabalho, e objetivos específicos que define os passos necessários para que se alcance a conclusão do TCC.

1.2.1 OBJETIVO GERAL

Investigar como o refinamento de informações providas por sensores pode beneficiar a modelagem, a síntese e a implementação de controladores para sistemas de automação a eventos discretos.

1.2.2 OBJETIVOS ESPECÍFICOS

- Fundamentar os conceitos de Sistemas a Eventos Discretos;
- Fundamentar a Teoria de Controle Supervisório de Sistemas a Eventos Discretos;
- Fundamentar o problema de síntese de controladores para SEDs envolvendo o refinamento e a distinção de eventos;
- Fundamentar o problema de síntese de controladores para SEDs envolvendo o uso de aproximações para modelos refinados;
- Avaliar os aspectos teóricos que se permeiam entre o uso de distinguidores e a versão do problema resolvidas por meio de aproximações;
- Aplicar abordagem no contexto de um exemplo de um sistema de controle para um processo industrial;

- Quantificar teoricamente os ganhos provenientes da abordagem em termos de esforço de modelagem e complexidade computacional;
- Explorar os aspectos de implementação de um controlador sintetizado através da abordagem investigada;
- Implementar o sistema de controle para o exemplo de modo tal que reflita a economia computacional provida em teoria.

1.3 JUSTIFICATIVA

Pesquisas recentes (TEIXEIRA *et al.*, 2014; CURY *et al.*, 2015) têm mostrado que a obtenção de controladores para alguns tipos de processos industriais esbarra em quesitos de complexidade, tais que tornam o problema intransponível (intratável) sem o uso de inovações teóricas e tratamento algorítmico avançado. Diante disso, uma alternativa apresentada para simplificar o processo de síntese de controladores para SEDs é o uso de distinguidores aliado ao conceito de aproximação (CURY *et al.*, 2015; BOUZON *et al.*, 2008).

Porém, mesmo simplificando o processo de síntese de supervisores, a implementação do sistema de controle ainda requer a interseção completa entre o supervisor e o modelo distinguidor. Ou seja, o supervisor é obtido através de um processo simplificado, mas precisa, inevitavelmente, ser associado ao distinguidor para fins de implementação. Isso porque, sem o modelo do distinguidor, a estrutura do supervisor é não-determinística em relação aos eventos (sinais) que ocorrem na planta. Isso implicaria que, após a ocorrência de determinado evento do sistema, o supervisor não saberia qual ação de controle tomar perante a instância desse evento sem o auxilio do modelo do distinguidor. Diante disso, é sugerido em Teixeira (2013), compor o supervisor com o modelo do distinguidor, o que de fato resolve o problema de implementação de um supervisor determinístico. Contudo, isso acaba gerando

um novo problema, que é a complexidade computacional da implementação, já que a composição entre supervisor e distinguidor é um procedimento cuja complexidade é de ordem exponencial, no pior caso. Assim, os benefícios provenientes do uso do distinguidor não se propagariam para o processo de implementação.

Dessa forma, esse trabalho investigou alternativas de como supervisor e distinguidor poderiam ser implementados em estruturas distintas.

Se ambos os modelos não compartilham da mesma estrutura de implementação, a tendência é a de que não haja necessidade de compô-los a priori, o que reduziria substancialmente a complexidade computacional das etapas pré-implementação, que é evidenciada na literatura (CURY *et al.*, 2015; TEIXEIRA, 2013).

Entretanto, se beneficiar de duas estruturas distintas para implementar dois modelos que a rigor são dependentes, implica em comunicar tais estruturas, a fim de que a dependência de dados entre eles seja tratada. Por exemplo, dada a ocorrência de um evento original na planta, a estrutura do supervisor pode habilitar mais de um refinamento desse mesmo evento. Como ele não incorpora o mecanismo de distinção, necessita consultar a estrutura do distinguidor para definir qual dos refinamentos é de fato elegível.

Nesse sentido, esse trabalho também desenvolve um mecanismo de comunicação entre o hardware que implementa os comandos do supervisor e o hardware que implementa o distinguidor. A ideia é a de que, dada a ocorrência de evento original na planta, o supervisor informa se as máscaras de tal evento são ou não elegíveis sob controle, tal como ocorre na TCS convencional. Caso não sejam, o evento original é desabilitado. No entanto, caso sejam elegíveis, se faz necessário saber quais máscaras sobrevivem à distinção. Para isso, um *request* é disparado ao distinguidor, que então retorna de maneira preditível qual dos eventos de fato sobrevive.

Após definida a estrutura de comunicação entre supervisor e

distinguidor, pretende-se ainda estender tal abordagem para o contexto da implementação de n módulos distinguidores. A construção de um distinguidor, ao contrário da representação de especificações complexas, leva a uma estrutura modular. Assim, este trabalho pretende tirar proveito dessa modularidade para reduzir a complexidade de implementação do sistema de controle. A ideia é a de que a coordenação entre o supervisor e cada módulo do distinguidor ocorra de maneira análoga a uma comunicação *broadcast*. Ou seja, toda a vez que o supervisor requisitar o distinguidor, ele lança um *broadcast* com eventos a serem distinguidos, o que por sua vez vai acionar todos e somente os módulos para os quais tais eventos são elegíveis no atual contexto de distinção (após o presente prefixo). O módulo (que, por construção, é único) capaz de distinguir tal evento fornece então a distinção necessária a estrutura do supervisor. Desse modo, espera-se o mesmo efeito do distinguidor sem, contudo, envolver a composição com o supervisor, em nenhum momento.

Como resultado estima-se que os benefícios do uso de distinguidores se propaguem, também, para as etapas de implementação.

2 REFERENCIAL TEÓRICO

2.1 SISTEMAS A EVENTOS DISCRETOS

A definição de sistema é apresentada de diversas maneiras na literatura. Dentre elas, Ogata (2010, p. 3) define que um sistema é a associação de componentes que atuam em conjunto, a fim de realizar um determinado objetivo, não sendo limitado a algo físico. Logo o conceito de sistema pode ser estendido a fenômenos abstratos e dinâmicos. Já para HAYKIN e Veen (2001, p. 22), um sistema é uma entidade capaz de manipular um ou mais sinais de forma a realizar uma determinada função, resultando em novos sinais. Alguns exemplos de aplicações que compõem um sistema são: a automação da manufatura, a robótica, redes de comunicação e de computadores, sistemas operacionais, entre outros.

De um modo geral, os sistemas têm em comum a propriedade de serem compostos por um conjunto de estados, denominado de *espaço de estados*, tal que cada estado representa o status do sistema em determinada circunstância, e *transições de estados* responsáveis pela evolução do sistema entre os estados. A distinção entre diferentes sistemas recai sobre o fato de como ocorrem essas transições de estados, podendo essas serem dirigidas pelo tempo ou a eventos (TEIXEIRA, 2013).

Os sistemas de interesse deste trabalho têm a característica de perceberem o ambiente no qual estão inseridos através de estímulos/sinais assíncronos, denominados *eventos*, e seu comportamento compõem um conjunto de estados enumerável. Um evento pode ser identificado como sendo uma ação especifica como, por exemplo, o início e o fim de uma tarefa, mas não o tempo transcorrido. Desta forma, estando em um estado, a ocorrência de um evento implica na transição para um novo estado ou não, no qual o sistema permanece até a ocorrência de um novo evento e assim sucessivamente. Estas mudanças de estados se caracterizam por serem abruptas, podendo ser visível

ou não a um observador externo (CURY, 2001, p. 9). Sistemas que apresentam essas características são denominados com *Sistemas a Eventos Discretos* (SEDs) (CASSANDRAS; LAFORTUNE, 2008, p. 30).

2.2 MODELAGEM DE SISTEMAS A EVENTOS DISCRETOS

O processo de se realizar experimentos sobre a estrutura real é a maneira mais eficaz de análise de um sistema, tendo como base o grau de precisão do resultados obtidos. Contudo, esse processo nem sempre é possível, seja devido a complexidade ou mesmo a indisponibilidade da estrutura real do sistema. Desse modo, uma alternativa é a de representar um sistema através de um modelo, que permite a abstração de características irrelevantes de seu comportamento em determinado contexto, de modo a facilitar a compreensão e manipulação do mesmo (OGATA, 2010; TEIXEIRA, 2013).

Os sistemas que possuem sua dinâmica guiada pelo tempo, geralmente são representados por equações diferenciais, enquanto que os SEDs são mais naturalmente representáveis por meio de diagramas de transição de estados, que permitem representar formal e intuitivamente o enlace entre os elementos *evento, transição e estado*, descritores fundamentais do comportamento de um SED.

Dentre os vários formalismos existentes para se representar SEDs destacam-se as *Redes de Petri*, a *Teoria de Filas* e a *Teoria de Autômatos e Linguagens*. Porém, nenhum desses formalismos é aceito de forma unânime ou como padrão na tarefa de se estabelecer modelos para SEDs, tendo em vista que cada formalismo possui características e finalidades distintas nessa tarefa (CURY, 2001; TEIXEIRA, 2013).

Este trabalho adota como formalismo de modelagem a *Teoria de Autômatos e Linguagens*, também usado na síntese de controladores ótimos para SEDs (CURY, 2001, p. 12).

2.2.1 TEORIA DE AUTÔMATOS E LINGUAGENS

Em um SED modelado por um autômato, a evolução no modelo mapeia a trajetória do sistema e é dada pela ocorrência consecutiva e finita de eventos, o que gera uma *cadeia*. Desta forma, o conjunto de cadeias geradas descrevem o comportamento do sistema. Com isso, seria lógico pensar que modelar um SED nada mais é do que descrever tais cadeias, e que por sua vez, processar operações sobre o modelo consiste em manipular suas cadeias (TEIXEIRA, 2013).

O conjunto de todos os eventos que compõem um SED é denominado *alfabeto*, denotado por Σ um alfabeto finito e não-vazio. E Σ^* denota o conjunto de todas as cadeias finitas possíveis de serem composta por eventos de Σ, incluindo a *cadeia vazia* denotada por ε, a qual representa a sequência com nenhum evento (RAMADGE; WONHAM, 1989).

Uma *linguagem* L sobre um alfabeto Σ, é um subconjunto de cadeias em Σ^*, isto é, $L \subseteq \Sigma^*$, que determina quais são as cadeias possíveis de serem formadas com eventos em Σ (CASSANDRAS; LAFORTUNE, 2008).

De modo geral, as classes de linguagens *regulares* e *não-regulares*, se distinguem de acordo com o grau de recursos necessário para expressar tais linguagens. Uma linguagem é dita ser regular quando pode ser expressa por um conjunto finito de elementos, mesmo que o número de cadeias que compõem essa linguagem possa ser infinito. No caso contrário, uma linguagem é não-regular quando nem sempre é possível expressa-lá por meio de um conjunto finito de elementos (CASSANDRAS; LAFORTUNE, 2008).

Uma vez que a modelagem de um SED geralmente compreende um conjunto finito de elementos, onde o comportamento desses elementos podem ser representados por um conjunto de cadeias finitas, a escolha das linguagens regulares se torna natural (TEIXEIRA, 2013).

Uma das formas de representar linguagens regulares é por meio

de um *Autômato Finito* (AF) (CASSANDRAS; LAFORTUNE, 2008). Um autômato finito é um diagrama de transição de estados, que permite modelar diversos problemas, através de um método prático e intuitivo (TEIXEIRA, 2013; CASSANDRAS; LAFORTUNE, 2008). A definição formal de um AF se da por meio de uma 5-tupla $\langle \Sigma, Q, q^{\circ}, Q^{\omega}, \rightarrow \rangle$, tal que:

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

Denota-se por $q_1 \xrightarrow{\sigma} q_2$, a transição do estado q_1 para o estado q_2, proveniente a ocorrência do evento $\sigma \in \Sigma$.

Uma das possíveis maneiras de se representar graficamente um autômato é através de um grafo dirigido, onde os nós representam os estados e as arestas representam as transições entre os estados. A transição de estado é desencadeada diante a ocorrência de um determinado evento, associado a ela. O estado inicial é identificado por uma seta uniconectada apontando para ele e os estados marcados por círculos duplos, cuja semântica define um estado de aceitação. De modo geral, o termo estado de aceitação está associada a idéia de tarefa completa (TEIXEIRA, 2013). Um exemplo de AF representado por meio de um grafo é mostrado na Figura 1, tal que $\Sigma = \{\alpha, \beta\}$, $Q = \{q_0, q_1\}$, $q^o = q_0$, $Q^w = \{q_0\}$ e a função de transição de estados é dada por $q_0 \xrightarrow{\alpha} q_1$, $q_1 \xrightarrow{\alpha} q_1$ e $q_1 \xrightarrow{\beta} q_0$.

A linguagem definida sobre um autômato G se divide em *linguagem gerada* e *linguagem marcada*, definidas respectivamente por

$$L(G) = \{s \in \Sigma^* | q^o \xrightarrow{s} q \in Q\} \text{ e}$$

$$L^w(G) = \{s \in \Sigma^* | q^o \xrightarrow{s} q \in Q^w\}.$$

A linguagem $L(G)$ representa o conjunto de todas as cadeias possíveis de serem geradas em G, enquanto que $L^w(G)$ o conjunto de todas as cadeias que alcançam um estado marcado, tal que $L^w(G) \subseteq L(G)$ (CASSANDRAS; LAFORTUNE, 2008). Para o exemplo apresentado na Figura 1, a linguagem gerada é o conjunto das cadeias ϵ e todas que iniciam com uma sequência de α terminando ou não por um β, e a linguagem marcada é o conjunto que contém as cadeias da linguagem gerada com exceção das cadeias que não terminam com β.

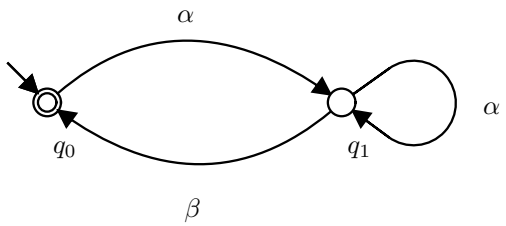

Figura 1 – Exemplo de Autômato Finito

2.2.2 OPERAÇÕES SOBRE AUTÔMATOS E LINGUAGENS

Uma vez que linguagens são conjuntos, por definição, então todas as operações sobre conjuntos, como por exemplo a união e intersecção, estão definidas. Além dessas, outras operações podem ser definidas sobre as linguagens a fim de lidar com os elementos do tipo cadeias de eventos (CURY, 2001; CASSANDRAS; LAFORTUNE, 2008; TEIXEIRA, 2013).

Um aspecto importante definido sobre as cadeias, são as *operações morfológicas*, que tem como principio estudar a forma/estrutura de cadeias. Desta forma, seja Σ um alfabeto e $s = pqr$ uma cadeia qualquer, com $p, q, r \in \Sigma^*$, então:

- p é um prefixo de s;

- q é uma subcadeia de s;
- r é um sufixo de s.

A concatenação entre duas linguagens L_1 e L_2, tal que $L_1, L_2 \subseteq \Sigma^*$, é denotada por L_1L_2, e definida por

$$L_1L_2 = \{s \in \Sigma^* | (s = s_1s_2) \wedge (s_1 \in L_1) \wedge (s_2 \in L_2)\}.$$

O prefixo-fechamento de uma linguagem $L \subseteq \Sigma^*$, cuja notação é $\overline{L}$, tal que

$$\overline{L} = \{s \in \Sigma^* | (\exists t \in \Sigma^*), st \in L\}.$$

O prefixo-fechamento $\overline{L}$ consiste na linguagem que contém todos os prefixos de todas as cadeias contidas em L. De modo geral $L \subseteq \overline{L}$.

Logo, se L for prefixo-fechada, então $L = \overline{L}$ e qualquer prefixo de qualquer cadeia de L, também é um elemento de L. Por exemplo, seja o alfabeto $\Sigma = \{a, b\}$, e as linguagens $L_1 = \{\varepsilon, a, b, ab, aba, ba\}$ e $L_2 = \{\varepsilon, a, b, aba\}$. Logo, L_1 é prefixo-fechada, pois todos os prefixo de L_1 são elementos de L_1. Já L_2 não é prefixo-fechada, uma vez que nem todos prefixos de L_2 são elementos de L_2, exemplo, o prefixo ab.

Outra operação útil no contexto de autômatos e linguagens é a *composição síncrona*, cuja notação é $\|$. Essa operação pode ser definida tanto para autômatos quanto para linguagens e espera-se que ela preserve a equivalência de resultados nas duas versões (WONHAM, 2002). Por praticidade esse trabalho define somente a operação $\|$ para autômatos.

Dado dois autômatos $A = \langle \Sigma_A, Q_A, q_A^\circ, Q_A^\omega, \rightarrow_A \rangle$ e $B = \langle \Sigma_B, Q_B, q_B^\circ, Q_B^\omega, \rightarrow_B \rangle$, a composição síncrona resulta no autômato

$$A \parallel B = (\Sigma_A \cup \Sigma_B, Q_A \times Q_B, (q_A^o, q_B^o), Q_A^w \times Q_B^w, \rightarrow),$$

em que a função de transição de estados é dada por:

- $(q_A, q_B) \xrightarrow{\sigma} (q'_A, q'_B)$, se $\sigma \in \Sigma_A \cap \Sigma_B$;
- $(q_A, q_B) \xrightarrow{\sigma} (q'_A, q_B)$, se $\sigma \in \Sigma_A \setminus \Sigma_B$;
- $(q_A, q_B) \xrightarrow{\sigma} (q_A, q'_B)$, se $\sigma \in \Sigma_B \setminus \Sigma_A$.

Em palavras, dada a ocorrência de um evento habilitado em ambos autômatos, a evolução de estado ocorre de maneira síncrona em ambos os modelos. Caso contrário, quando o evento está habilitado em somente um dos autômatos a evolução de estado ocorre de maneira assíncrona, ou seja, de maneira independente em cada autômato e somente o autômato que reconhece o evento evolui de estado (CURY, 2001; TEIXEIRA, 2013).

A definição de produto síncrono pode ser naturalmente estendida a n autômatos. Assim, sejam os autômatos $G_i = \langle \Sigma_i, Q_i, q_i^\circ, Q_i^\omega, \rightarrow_i \rangle$, para $i = 1, 2, \cdots, n$, um modelo global G é dado por meio da composição síncrona

$$G = \bigparallel_{i=1}^{n} G_i, \text{ tal que } \Sigma = \bigcup_{i=1}^{n} \Sigma_i.$$

Já os comportamentos gerado e marcado resultantes da operação de composição entre n autômatos são mostrados em Wonham (2002), como

$$L(G) = \bigparallel_{i=1}^{n} L(G_i) \text{ e } L^w(G) = \bigparallel_{i=1}^{n} L^w(G_i).$$

Dessa forma, a composição síncrona permite que se trabalhe com cada modelo G_i separadamente, de modo que esses modelos possam ser compostos posteriormente, e por consequência levando ao modelo global do sistema (G).

Algumas propriedades importantes sobre um dado autômatos $G = \langle \Sigma_G, Q_G, q_G^\circ, Q_G^\omega, \rightarrow_G \rangle$, estão relacionadas ao conceito de acessibilidade. Essas propriedades são definidas como:

- *Estado acessível*: Um estado $q \in Q_G$ é dito ser acessível se $\exists s \in \Sigma^*$, tal

que $q_G^o \xrightarrow{s} q$.

- *Autômato acessível*: G é dito ser acessível se q é acessível, $\forall q \in Q_G$.
- *Autômato co-acessível*: G é dito ser co-acessível se cada cadeia $s \in L(G)$ é prefixo de uma cadeia marcada. Ou seja, se cada $s \in L(G)$ pode ser completado por algum $t \in \Sigma^*$ tal que $st \in L^w(G)$, isto é, $q_G^o \xrightarrow{st} q$ tal que $q \in Q_G^w$. Ou ainda, se $L(G) = \overline{L^w(G)}$.
- *Autômato não-bloqueante*: G é dito ser não-bloqueante se é co-acessível.
- *Autômato Trim*: G é dito ser trim, se ele é acessível e co-acessível.

A propriedade de *não-bloqueio* de autômatos, esta relacionado ao conceito de acessibilidade, desta forma o comportamento gerado por um autômato G, é dito ser não-bloqueante se e somente se o autômato é co-acessível.

2.2.3 MODELAGEM DE SED USANDO AUTÔMATOS

Tendo em mente a noção de sistema, e tomando autômatos como forma de modelar SEDs, o primeiro passo a ser realizado é a modelagem dos componentes do sistema, geralmente um conjunto de itens de uma máquina, resultando em um conjunto de AFs. Logo, a composição síncrona dos elementos desse conjunto leva ao comportamento geral do sistema, sendo este denominado como *planta*. A planta do sistema está em *malha aberta* quando o comportamento do sistema não sofre nenhuma ação de controle.

Um exemplo simples de um SED modelado por AFs (Figura 2) seria considerar a operação de duas máquinas, M_1 e M_2, modeladas respectivamente por G_1 e G_2, tal que a composição síncrona entre as máquinas resulte no modelo da planta em malha aberta $G = G_1 \parallel G_2$. O comportamento das máquinas M_1 e M_2 é descrito em termos de eventos de início (a e c) e final de operação (b e d), conforme a seguir.

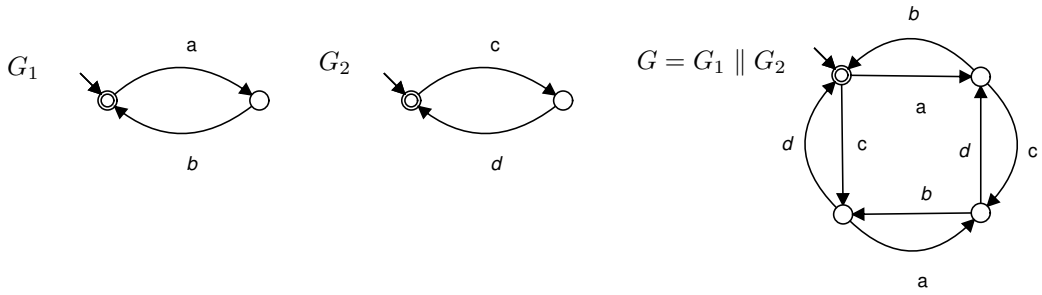

Figura 2 – Modelagem da planta de um SED

O comportamento de uma planta em malha aberta, em geral, não atende ao comportamento que se espera na prática, quando em operação, ou seja, $L(G)$ permite cadeias indesejáveis na planta. Uma maneira de resolver isso seria evitar a ocorrência de tais cadeias, por meio de um conjunto de restrições sendo este denominado de *especificação*. As especificações de controle são modeladas de forma análoga à planta, mas com o intuito de representar uma ação proibitiva no sistema, de modo que observa os eventos possíveis na planta e desabilita os eventos que são considerados proibidos em determinado contexto.

Nesse sentido, Teixeira *et al.* (2015) apresentam uma metodologia para auxiliar no processo de modelagem das especificações. Essa metodologia parte da definição de um conjunto de especificações para uma planta G, denotado por $E = \mathop{\Vert}_{i=1}^{n} E_i$, onde cada modelo de especificação E_i pode ser obtido através dos seguintes passos:

(i) Interpretar a especificação descrita em forma textual;

(ii) identificar o evento $\sigma \in \Sigma$ a ser desabilitado. Neste caso, σ representa a ação que deve ser proibida na planta. Uma boa prática, tendo em mente a simplicidade e modularização, é que cada modelo da especificação desabilite somente um evento;

(iii) identificar o(s) evento(s) $\sigma' \in \Sigma$ que imediatamente precedem σ;

(iv) construir o AF E_i que, somente após σ', habilite σ;

(v) habilitar σ' em todos os estados $q \in Q_{R_i}$.

Vale salientar que o processo de modelagem do conjunto de especificações para um planta G, denotado por $E = \mathop{||}_{i=1}^{n} E_i$, é um processo empírico e está diretamente associado ao grau de experiência e a familiaridade que o engenheiro/projetista possui do sistema.

Um fato importante a se observar, quando se está modelando E_i, é que as cadeias que pertencem ao comportamento do sistema não têm que necessariamente ser mapeadas no AF a partir do estado inicial, umas vez que essas, de fato, estão habilitadas em $L(G)$. Logo, tudo que se deve saber a priori dessas cadeias, é o último evento do prefixo de σ, ou seja, um $\sigma' \in \Sigma$.

Desta forma, a associação das especificações ao modelo da planta que pode ser dada por composição síncrona, resulta em um modelo que representa o comportamento esperado do sistema sob controle, isto é, o sistema em *malha fechada*.

Um exemplo de especificação relacionada a planta G (Figura 2), seria a restrição de precedência de início de operação entre as máquinas (a partir do estado inicial), de forma que M_2 só possa iniciar após o fim de operação de M_1. Em outras palavras, a especificação deve proibir a ocorrência de cadeias que contenham prefixo que iniciam com o evento c. A Figura 3 contextualiza essa especificação.

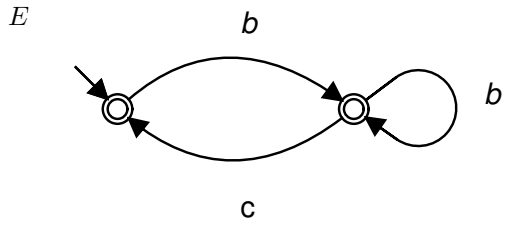

Figura 3 – Modelagem de uma especificação

No estado inicial, o evento c está sendo proibido já que se deseja estabelecer a precedência de inicio de M_1 em relação a M_2, e o evento que

precede c é o evento b que está habilitado. Por analogia, $b \equiv \sigma'$ e $c \equiv \sigma$.

2.3 TEORIA DE CONTROLE SUPERVISÓRIO

O comportamento esperado de um sistema sob controle, obtido pela composição da planta (G) e suas especificações (E), modela a ação de controle de E sobre G, que em alguns casos pode não ser consistente com o comportamento controlável de um SED na prática. Essa inconsistência se dá pelo fato de E poder estar desabilitando eventos que não podem ser diretamente desabilitados em G.

A falha/quebra de um equipamento em uma linha de produção é um exemplo desse tipo de evento, que tem a particularidade de ser espontâneo e não depender de qualquer política de controle. Logo, para evitar essa inconsistência seria necessário estabelecer uma maneira de distinguir a natureza da ocorrência dos eventos que compõem o sistema, de tal forma que a ação de controle tenha conhecimento de quais eventos podem ser desabilitados ou não na planta.

Nesse sentido, a TCS (RAMADGE; WONHAM, 1989) particiona o conjunto de eventos da planta de um SED, tal que $\Sigma = \Sigma_{\mathrm{c}} \cup \Sigma_{\mathrm{u}}$, onde Σ_{c} denota o conjunto de *eventos controláveis*, cuja a ocorrência pode ser inibida na planta, e Σ_{u} denota o conjunto de todos os *eventos não-controláveis*, os quais não podem ser diretamente desabilitados.

A entidade da TCS que efetivamente implementa a ação de controle na planta é o *supervisor*. Formalmente, um supervisor S é um mapa $S : L(G) \rightarrow 2^{\Sigma}$, associada a uma linguagem $L_S \subseteq L^w(G)$ que, após qualquer cadeia $s \in L(G)$, observa eventos elegíveis em G e informa, dentre eles, quais devem ser, de fato, habilitados. Assim, a ação de controle de S sobre G, denotada por S/G consiste em habilitar eventos do conjunto $S(s) \in \Sigma$. A estrutura de supervisão que interage com a planta de forma a fechar a malha de controle, é ilustrada na Figura 4.

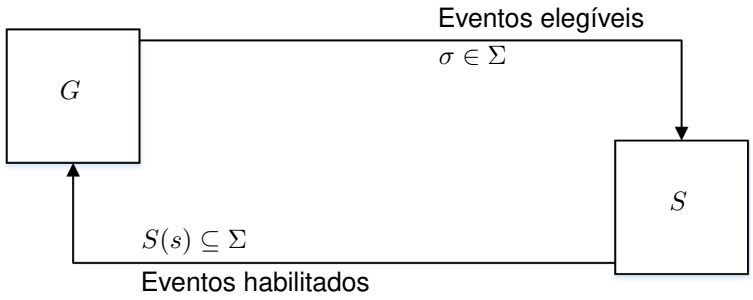

Figura 4 – Fluxo de controle

Assume-se que eventos $\sigma \in \Sigma$ ocorrem espontaneamente na planta. Quando σ é observado, após uma cadeia qualquer $s \in L(G)$, o supervisor S atualiza o conjunto $S(s)$ de eventos habilitados. Os eventos possíveis na planta que não pertencem a $S(s)$ são justamente aqueles que estão sendo inibidos pela ação de controle.

O conjunto de cadeias de $L(G)$ que sobrevive sob controle representa o *comportamento gerado em malha fechada*, e é dado pela linguagem $L(S/G)$. O *comportamento marcado em malha fechada*, por sua vez, é dado por $L^{\omega}(S/G) = L(S/G) \cap L_{\mathrm{S}}$. Nesse caso, S é definido como um tipo especial de supervisor, denominado *marcador*, cuja ação de controle sobre G (S/G) além de habilitar eventos do conjunto $S(s) \subseteq \Sigma$, também marca cadeias $s \in L_{\mathrm{S}}$.

Um supervisor S é dito ser *não-bloqueante* quando $L(S/G) = \overline{L^{\omega}(S/G)}$. Dessa maneira, sempre que uma cadeia sobrevive sob controle, essa cadeia é um prefixo de uma cadeia marcada. Assim, garante-se que a evolução do sistema sob controle sempre leva a completar uma tarefa e, logo, o sistema controlado nunca bloqueia.

Diante a essas definições Ramadge e Wonham (1989) enunciam o *Problema de Controle Supervisório* como:

Seja uma planta G, com eventos em Σ, e uma especificação $E \subseteq \Sigma^{*}$, definindo um comportamento desejado $K = E \cap L^{\omega}(G)$, deve-se encontrar um supervisor não-bloqueante S tal que $L^{\omega}(S/G) \subseteq K$.

Encontrar uma solução para o PCS passa, sobretudo, pelo conceito de *controlabilidade*. Uma linguagem $K \subseteq \Sigma^*$ é dita ser *controlável* em relação a L quando

$$\overline{K}\Sigma_{\mathrm{u}} \cap L \subseteq \overline{K}.$$

Ou seja, após qualquer prefixo de K, se um evento não-controlável é observado em L, a cadeia resultante continua sendo um prefixo de K. Garante-se, então, que nenhum evento não-controlável, possível em L, esteja sendo desabilitado pela ação de controle.

A controlabilidade de uma linguagem não vazia $K \subseteq L^{\omega}(G)$ é uma condição necessária e suficiente para a existência de um supervisor marcador não-bloqueante S, tal que $L^{\omega}(S/G) = K$. Nesse caso, S, tal que $L^{\omega}(S/G) = K$, pode ser implementado por um autômato V, tal que $K = L^{\omega}(V) \cap L^{\omega}(G)$ e $\overline{K} = L(V) \cap L(G)$. A ação de controle de S é implementada pela desabilitação de eventos possíveis em $L(G)$, os quais não são possíveis em $L(V)$, após uma cadeia $s \in L(S/G)$. A função de marcação corresponde a marcar cadeias que são marcadas em ambos V e G.

Quando K não atende à condição de controlabilidade, torna-se necessário que se calcule a sub-linguagem controlável que mais se aproxima de K, i.e., a *máxima linguagem controlável*.

Seja $\mathcal{C}(K,G) = \{L \subseteq K \mid L$ é controlável em relação a $L(G)\}$. $\mathcal{C}(K,G)$ possui um elemento supremo único, denotado por $\sup\mathcal{C}(K,G)$, o qual representa a máxima linguagem controlável, i.e., a sub-linguagem controlável que mais se aproxima de K. Assim, $\sup\mathcal{C}(K,G)$ pode ser associado ao comportamento menos restritivo possível de ser implementado por um supervisor não-bloqueante S sobre G, de maneira a respeitar o conjunto de especificações.

Então S, tal que $L^{\omega}(S/G) = \sup\mathcal{C}(K,G)$ e $L(S/G) = \overline{\sup\mathcal{C}(K,G)}$, é uma solução ótima para o PCS. Caso $\sup\mathcal{C}(K,G)$ não possa ser obtido, implica que o PCS não possui solução.

O cálculo do elemento supremo $\sup\mathcal{C}(K, G)$ é um processo interativo que consiste em basicamente identificar e remover *maus estados* do autômato E que representa a especificação K. Um estado x é dito ser mau estado se, após a ocorrência de alguma cadeia $s \in \Sigma^*$, existe $\mu \in \Sigma_{\mathrm{u}}$, tal que $G \xrightarrow{s} x \xrightarrow{\mu}$ e $E \xrightarrow{s} x$. Ou seja, se após a ocorrência de uma cadeia s, definida tanto na planta quanto na especificação, resulte em um estado x em ambos os modelos, tal que somente a planta habilite um evento não-controlável a partir desse estado, então esse é dito ser um mau estado em E.

O algoritmo que calcula o $S = \sup\mathcal{C}(K, G)$, dado sobre os autômatos $G = \langle \Sigma_G, Q_G, q_G^\circ, Q_G^\omega, \rightarrow_G \rangle$ e $E = \langle \Sigma_E, Q_E, q_E^\circ, Q_E^\omega, \rightarrow_E \rangle$, tal que $L^\omega(E) = K \subseteq L^\omega(G)$, é definido pelos seguintes passos:

(i) Identificar maus estados em E. Caso não existam, $S = E$ e fim.

(ii) Remover maus estados em E. Caso existam, atualize E, eliminando os maus estados identificados.

(iii) Testar o bloqueio em E. Calcule a componente *trim* de E. Uma componente é trim se é acessível e co-acessível. Volte ao passo 1.

Como resultado obtém-se o comportamento em malha fechada minimamente restritivo e não-bloqueante, tal que $L^\omega(S) = \sup\mathcal{C}(E)$ e $L(S) = \overline{L^\omega(S)}$, logo, essa é uma solução ótima para o PCS.

2.3.1 EXEMPLO DE UM SISTEMA A EVENTOS DISCRETOS

Considere como exemplo de um SED uma pequena fábrica (Figura 5) composta por duas máquinas, M_1 e M_2, operando de maneira sequencial, tal que as duas máquinas são interligadas por um *buffer* B capaz de suportar a estocagem de até n peças empilhadas, adotando a política *Last In, First Out* (LIFO) de empilhamento.

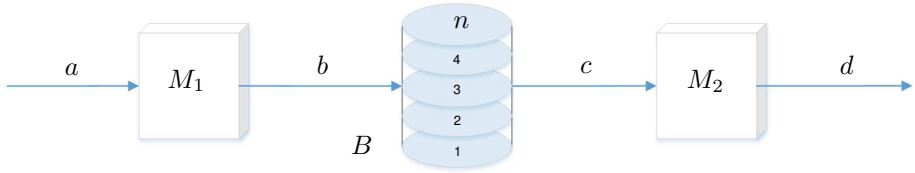

Figura 5 – Exemplo de um SED

O principio básico de funcionamento de cada máquina é que dada a ocorrência de um evento $\sigma_i \in \{a, c\}$, a máquina M_i carrega uma peça e realiza uma operação sobre a mesma, ao finalizar a operação a máquina descarrega automaticamente a peça e essa ação está associada a um evento $\beta_i \in \{b, d\}$. A Figura 6 apresenta os autômatos G_1 e G_2, modelando respectivamente M_1 e M_2.

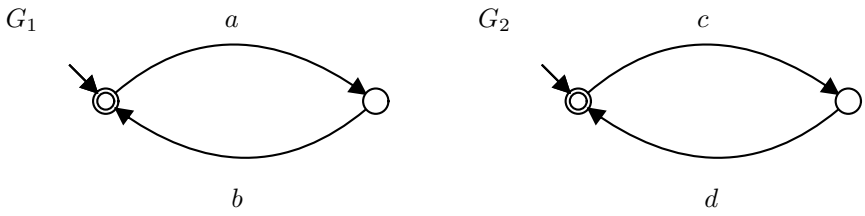

Figura 6 – Modelos para os subsistemas M_1 e M_2

Os eventos a e c modelam o início de operação de M_1 e M_2, respectivamente, enquanto b e d, o final. O modelo global do sistema, apresentado na Figura 7, é obtido pela composição síncrona $G = G_1 \parallel G_2$, com o conjunto de eventos $\Sigma = \{a, b, c, d\}$, onde é assumido que $\Sigma_{\mathrm{c}} = \{a, c\}$ e $\Sigma_{\mathrm{u}} = \{b, d\}$.

O objetivo de controle para esse exemplo, consiste em evitar o *underflow* e *overflow* no *buffer*, ou seja, que M_1 descarregue peças no buffer quando estiver cheio, e M_2 retire peças quando o *buffer* estiver vazio. Para isso obtém-se o modelo da especificação E com $n+1$ estados (Figura 8), onde cada estado marcado, exceto o inicial, representa a ocupação de uma posição de B.

No estado inicial, um evento c é proibido, já que a ocorrência desse antes de um evento b resultaria no *underflow*. Após o empilhamento de n peças, um evento b é desabilitado, pois a ocorrência de mais um evento b levaria à

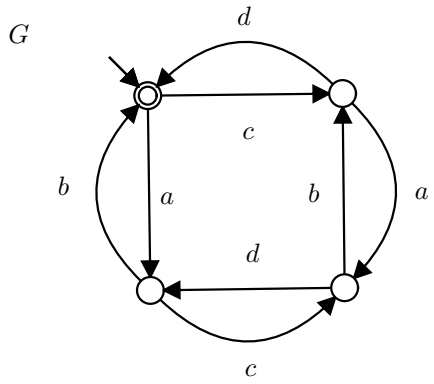

Figura 7 – Modelo global do sistema

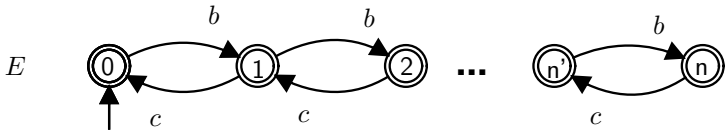

Figura 8 – Modelo da especificação de controle E

condição de *overflow*.

Diante ao modelo da planta G e do modelo da especificação E, o próximo passo é obter o comportamento desejado para o sistema sob controle, ou seja, $K = G \parallel E$. Contudo, o comportamento desejado nesse caso é inconsistente com o comportamento controlável do sistema na prática, uma vez que E esta desabilitando b, um evento não-controlável, no estado que representa a n-ésima posição do buffer. Para resolver tal inconsistência, o supervisor obtido pelo cálculo de $\sup\mathcal{C}(K, G)$, desabilita a n-ésima primeira instância (contida em uma cadeia) do evento a na planta de forma a evitar os maus estados contido em K. Em outras palavras, como não é possível desabilitar diretamente a ocorrência de b que leva ao *overflow*, o supervisor desabilita o evento imediatamente anterior, que levaria à ocorrência de b, nesse caso, o a. Ainda em outras palavras: se não é possível evitar que uma ação termine, então nem a inicie.

2.4 TEORIA DE CONTROLE SUPERVISÓRIO COM DISTINGUIDORES

O uso de distinguidores na TCS tem como objetivo facilitar a tarefa de modelagem de especificações de controle, podendo ainda, em certos casos diminuir a complexidade da síntese de uma solução para o PCS. Os distinguidores foram inicialmente propostos por Bouzon *et al.* (2008), os quais foram pensados como tipos especiais de sensores que ao serem associados ao modelo de um SED, permitem prover mais informações sobre certos eventos que pertencem ao sistema. Já Teixeira (2013), apresentou uma versão estendida da abordagem de distinguidores na TCS.

Considere um SED modelado com eventos em Σ, refinar tais eventos na abordagem com distinguidores, em geral, é considerar que cada evento $\sigma \in \Sigma$ torna-se uma máscara para um conjunto não-vazio de eventos refinados, denotado por Δ^σ. Os eventos refinados que compõem esse conjunto são escolhidos de forma a identificar diferentes instâncias que a máscara (σ) pode ocorrer na planta. Então, Σ é um conjunto de máscara para o alfabeto refinado $\Delta = \Delta_c \cup \Delta_u$, onde $\Delta_c = \bigcup_{\sigma \in \Sigma_c} \Delta^\sigma$ e $\Delta_u = \bigcup_{\sigma \in \Sigma_u} \Delta^\sigma$ (TEIXEIRA, 2013; CURY *et al.*, 2015).

A relação entre os alfabetos Σ e Δ é definida por um *mapa mascarador* $\Pi : \Delta^* \to \Sigma^*$, definido indutivamente por

$$\Pi(\epsilon) = \epsilon,$$
$$\Pi(t\delta) = \Pi(t)\sigma, \ para\ t \in \Delta^*, \delta \in \Delta^\sigma\ e\ \sigma \in \Sigma.$$

Ou seja, Π é um mapa que relaciona cada cadeia de eventos refinados em Δ^*, à respectiva cadeia máscaras em Σ^*. A definição de Π, pode ser naturalmente estendida para qualquer linguagem $L_d \subseteq \Delta^*$ por

$$\Pi(L_{\mathrm{d}}) = \{s \in \Sigma^* | \exists t \in L_{\mathrm{d}}, \Pi(t) = s\}.$$

Por definição, o mapa Π é dito ser prefixo-preservante, ou seja, se

$s \leq t$, então $\Pi(s) \leq \Pi(t)$. Isto é, se s é um prefixo de t, após aplicar o mapa Π em ambas as cadeias de eventos refinados, a relação de prefixo entre as cadeias ainda permanece no domínio do alfabeto Σ. Além disso, Π comuta com o prefixo-fechamento, ou seja, para $L_d \subseteq \Delta^*, \overline{\Pi(L_{\mathrm{d}})} = \Pi(\overline{L_{\mathrm{d}}})$.

O *mapa mascarador inverso* $\Pi^{-1} : \Sigma^* \to 2^{\Delta^*}$ é definido por

$$\Pi^{-1}(s) = \{t \in \Delta^* | \Pi(t) = s\}.$$

Análogo ao mapa mascarador, o Π^{-1} também pode ser estendido para qualquer linguagem $L \subseteq \Sigma^*$ por

$$\Pi^{-1}(L) = \{t \in \Delta^* | \Pi(t) \in L\}.$$

Definidos os mapeamentos, o efeito de um distinguidor $D : \Sigma^* \to 2^{\Delta^*}$ sobre uma linguagem $L \subset \Sigma^*$ pode ser ilustrado por:

$$D(L) = \Pi^{-1}(L) \cap L_{\mathrm{d}}.$$

O distinguidor D leva as cadeias do domínio de Σ^* para cadeias refinadas no domínio de Δ^*, através do mapa Π^{-1}, e distingue a ocorrência das cadeias refinadas por meio de uma linguagem L_{d}.

Nesse caso, L_{d} é uma *linguagem distinguidora*, definida por $L_{\mathrm{d}} = \overline{L_{\mathrm{d}}} \subseteq \Delta^*$. Uma vez que L_{d} é prefixo-fechada, por definição, e o mapa Π^{-1} é um caso particular de D, então L_{d} e $\Pi^{-1}(L)$ são não-conflitantes, para qualquer linguagem $L \subseteq \Sigma^*$. Essa propriedade é formalizada pela condição necessária e suficiente, definida por $\overline{\Pi^{-1}(L) \cap L_{\mathrm{d}}} = \overline{\Pi^{-1}(L)} \cap \overline{L_{\mathrm{d}}}$ (TEIXEIRA, 2013).

Note que L_{d} pode estar associada, por exemplo, à distinção de cada cadeia em Σ^* por todas as possíveis cadeias em Δ^*; ou à distinção de cada cadeia em Σ^* por exatamente uma cadeia em Δ^*.

Isso leva à classificação de distinguidores, que está relacionada com a cardinalidade do conjunto de cadeias geradas em Δ^* para cada cadeia em

Σ^*. Essa classificação se divide em *preditível* e *não-preditível*. Dizer que um distinguidor $D : \Sigma^* \to \Delta^*$ é preditível, significa que para todo $s \in \Sigma^*$ é sempre verdade que $|D(s)| = 1$, ou seja, se após a ocorrência de uma cadeia $r \in L_{\mathrm{d}}$ exista um, e apenas um evento refinado habilitado para cada máscara $\sigma \in \Sigma$, de modo a permitir que L_{d} possa sempre *predizer* a próxima distinção de qualquer máscara. Quando D é não-preditível implica em $|D(s)| > 1$ para algum $s \in \Sigma^*$ (TEIXEIRA, 2013).

Considerando um SED, cujo comportamento é modelado por autômato G, o efeito de um distinguidor D sobre G é dado por

$$D(L(G)) = \Pi^{-1}(L(G)) \cap L_{\mathrm{d}};$$
$$D(L^{\omega}(G)) = \Pi^{-1}(L^{\omega}(G)) \cap L_{\mathrm{d}}.$$

Um diagrama de blocos que contextualiza a interação entre o distinguidor e a planta (G) é apresentada na Figura 9.

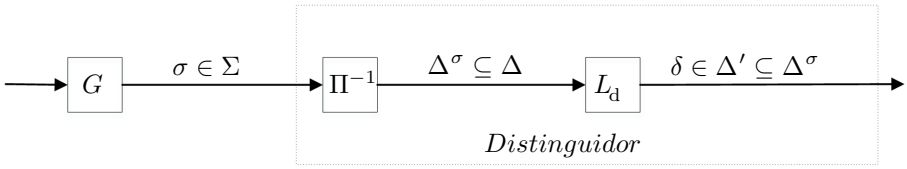

Figura 9 – Diagrama de blocos interação de D e G

Nesse diagrama, assume-se que a cada ocorrência de $\sigma \in \Sigma$ em G, esse evento é mapeado em um subconjunto Δ^{σ} de refinamentos, por meio do mapa Π^{-1}, que por sua vez são filtrados pela linguagem L_{d}, a qual define de fato quais deles são habilitados na planta.

Nesse contexto, G_d é autômato tal que $L(G_d) = D(L(G))$ e $L^{\omega}(G_d) = D(L^{\omega}(G))$, e E_{d} denota uma especificação definida em Δ. Os requisitos expressos por E_{d} usando as informações adicionais providas pelo distinguidor devem se referir aos mesmos requisitos expressos por E em Σ, tal que $E_{\mathrm{d}} \cap L^{\omega}(G_d) = K_d = D(K)$, onde $K = E \cap L^{\omega}(G)$. Nesse caso, a especificação $K \subseteq L^{\omega}(G)$ seria dada por $K = \Pi(E_{\mathrm{d}} \cap \Pi^{-1}(L^{\omega}(G)) \cap L_{\mathrm{d}})$.

Então, o problema de controle com distinguidor (PCS-D) consiste em encontrar um supervisor não-bloqueante $S_d : \Delta^* \to 2^{\Delta}$, tal que $L^{\omega}(S_d/G_d) \subseteq K_d$. Pode ser demonstrado (TEIXEIRA, 2013) que, se um distinguidor D é preditível, então

$$\Pi(\sup\mathcal{C}(K_d, G_d)) = \sup\mathcal{C}(K, G);$$
$$\sup\mathcal{C}(K_d, G_d) = D(\sup\mathcal{C}(K, G)).$$

A Figura 10 contextualiza a arquitetura do problema de controle supervisório com distinção preditível de eventos.

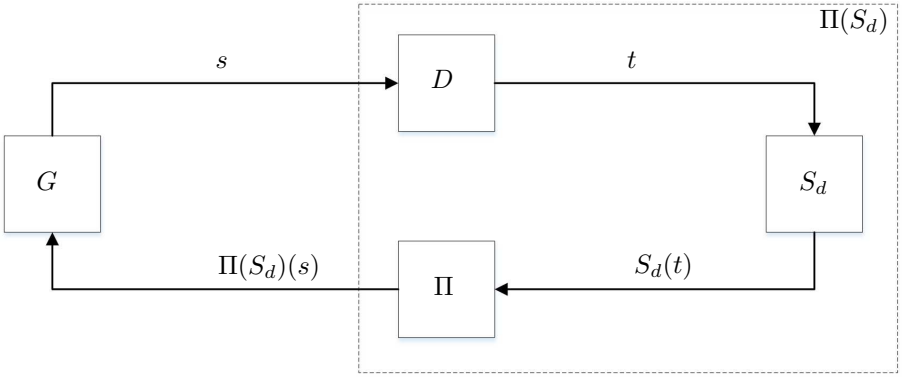

Figura 10 – Arquitetura de controle supervisório com distinguidor preditível

Após uma cadeia $s \in L(G)$ ser observada na planta G, o distinguidor preditível D identifica a cadeia $t \in L(G_d)$ correspondente a s, tal que $D(s) = t$. Então, o supervisor S_d define o conjunto $S_d(t)$ de eventos habilitados após a cadeia t. Por fim, $\Pi(S_d)$ mapeia esse conjunto de eventos refinados para os eventos originais correspondentes, o que permite que a ação de controle para o alfabeto original seja implementada sobre G.

2.4.1 CONSTRUÇÃO DE UM DISTINGUIDOR

Resta saber como representar um distinguidor D. Essa é uma tarefa de modelagem que, portanto, depende da capacidade do engenheiro, o que torna inconveniente esquematizá-las por completo. Ainda assim Teixeira

(2013), propõem um guia para o desenvolvimento de um distinguidor. Esse guia descreve os seguintes passos:

(i) *Identificação de um conjunto inicial de eventos $\sigma \in \Sigma$ a serem refinados*: identificar quais eventos em Σ, quando refinados, poderiam simplificar a modelagem de uma especificação de controle;

(ii) *Definição das instâncias de refinamentos $\delta \in \Delta^{\sigma}$*: a partir do passo anterior, deve-se determinar quais instâncias devem ser associadas a cada um dos eventos selecionados para o refinamento. Tais instâncias representam diferentes circunstâncias que o evento original pode ocorrer. Assim, cada instância também carrega uma semântica particular, a qual deve ser implementado pelo modelo do distinguidor;

(iii) *Complementação do conjunto Δ*: a partir de um dado conjunto de eventos a serem refinados (máscaras) e seus respectivos conjuntos de instâncias (refinamentos), a construção de um modelo que distingue a ocorrência de tais instâncias pode depender de outros refinamentos. De fato, o significado de uma certa instância de um evento pode se concretizar somente quando o sentido de outro evento é conhecido. Então, tal evento deve ser refinado e suas instâncias serem distinguidas. Logo, é necessário revisitar os passos (i) e (ii), e assim definir corretamente o alfabeto Δ;

(iv) *Modelagem do distinguidor*: neste passo, se constrói um modelo que estabelece as interdependências entre as instâncias de refinamentos em Δ. Esse processo consiste na definição de um autômato H_{d}, tal que a linguagem associada desse descreve a linguagem distinguidora, isto é, $L(H_{\mathrm{d}}) = L_{\mathrm{d}}$. O modelo H_{d}, é naturalmente simples no aspecto de modelagem e, em geral, pode ser construído de forma modular.

2.4.2 EXEMPLO DE UM SED COM DISTINGUIDOR

Para fins de comparação é utilizado o mesmo sistema apresentado na Figura 6. Abordar esse exemplo com o uso de distinguidor, consiste em primeira instância construir o modelo do distinguidor. Para essa tarefa é utilizado o guia supracitado, que resulta em:

(i) *Identificação de um conjunto inicial de eventos a serem refinados*: Definir um modelo para a especificação E, de modo que venha evitar o *underflow* e *overflow* consiste em saber quando o *buffer* está vazio ou cheio, ou seja, ter o conhecimento de quantas peças estão presentes no *buffer*. Diante disso, é natural a escolha dos eventos b e c, que respectivamente inserem e removem peças no *buffer* (B), para serem refinados, uma vez que isso permite a simplificação da especificação E.

(ii) *Definição das instâncias de refinamentos*: Baseado no passo anterior é essencial refinar os eventos b e c, de forma que cada instância refinada de b passe a identificar a ação de inserir uma peça em uma posição de B, enquanto que cada instância refinada de c passe identificar a ação de remover uma peça de determinada posição de B.

Desse modo, considere $\Delta^b = \{b_1, b_x, b_n, b_{n+1}\}$ e $\Delta^c = \{c_0, c_1, c_x, c_n\}$ tal que c_0 e b_{n+1} expressam o *underflow* e *overflow*, respectivamente. Os outros refinamentos representam as posições intermediárias.

(iii) *Complementação de* Δ: Os eventos a e d não requerem refinamentos, então $\Delta^a = \{a\}$ e $\Delta^d = \{d\}$. Logo, o alfabeto refinado de Σ é dado por $\Delta = \Delta^a \cup \Delta^b \cup \Delta^c \cup \Delta^d$.

(iv) *Modelagem do distinguidor*: Um distinguidor que implementa a relação entre os refinamentos em Δ é apresentado na Figura 11.

O modelo H^y distingue os eventos refinados dos conjuntos Δ^b e Δ^c, no estado inicial o modelo distingue o evento c_0 dos demais refinamentos da

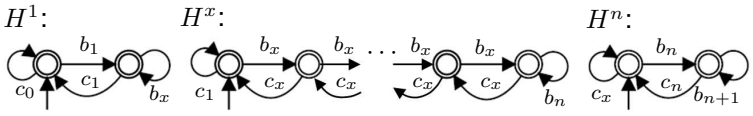

Figura 11 – Modelo do distinguidor $H^y = H^1 \,||\, H^x \,||\, H^n$

máscara c, ainda nesse estado, distingue b_1 dos outros refinamentos de b. No próximo estado, distingue o evento c_1, dos demais refinamentos de c, e b_x dos demais refinamentos de b. Este comportamento é estendido de maneira análoga, para os outros refinamentos de b e c nos demais estados de H^y.

Observe que a única restrição que o modelo proposto impõe é a precedência entre os eventos do conjunto de refinamentos associado a máscara, porém isso não altera a equivalência do comportamento do sistema refinado em relação ao definido em Σ. Essa equivalência é mantida pelo fato de que, em cada estado do modelo ao menos uma instância dos eventos que se deseja distinguir está habilitada. Mais que isso, nesse exemplo, o modelo H^y habilita exatamente uma única instância dos eventos que se deseja distinguir em cada estado.

Após construir os modelos que implementam a semântica dos conjuntos Δ^b e Δ^c, resta obter o modelo do distinguidor D, denotado por $H_{\mathrm{d}} = H^y \parallel H^{\Delta}$, onde $L^{\omega}(H^{\Delta}) = \Delta^*$ e $L(H_{\mathrm{d}}) = L_{\mathrm{d}}$. O modelo H^{Δ} é um autômato composto por um único estado, sendo esse um estado marcado, contendo uma transição em autolaço, rotulada com os eventos de Δ, complementando o alfabeto em H_{d}. Para este exemplo em particular, o modelo de H_{d} possui $n + 1$ estados. Como esse modelo preserva, em cada estado, um único refinamento habilitado para cada máscara, então esse modelo representa um distinguidor predítivel.

Fazendo uso dos eventos refinados em Δ, a especificação E (Figura 8 da pág.31) pode ser representada por dois autômatos E_{d}^{u} e E_{d}^{O} (Figura 12), cujos comportamentos preservam o mesmo requisito de controle, ou seja, *underflow* e *overflow*, tal que $E_{\mathrm{d}} = E_{\mathrm{d}}^{U} \parallel E_{\mathrm{d}}^{O}$ corresponde a

especificação E em Δ. O fato de E_{d} preservar a mesma semântica de requisito de controle que E, resulta que, caso não se conheça o modelo E, uma forma de obtê-lo é através de $E = \Pi(E_{\mathrm{d}} \parallel H_{\mathrm{d}})$.

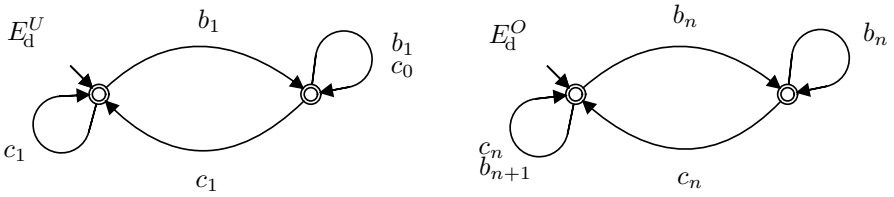

Figura 12 – Modelos de E_{d_U} e E_{d_O}

O modelo E_{d}^{U} desabilita no estado inicial um evento c_0, já que sua semântica está associada a remover uma peça de B quando seu status é vazio, portanto a ocorrência desse evento resultaria no *underflow*. Já o modelo E_{d}^{O} após um evento b_n (buffer cheio), desabilita o evento b_{n+1} cujo o significado está associada a inserção de uma peça quando B está cheio.

Note que, para estabelecer um modelo de especificação equivalente com uso de distinguidor, o esforço da modelagem diminui significativamente, uma vez que cada modelo que compõem E_{d} é composto de dois estados (valor constante, independente do número de posições de B), enquanto que no domínio de Σ o número de estados do modelo da especificação E é dado por $n + 1$, onde n é número de posição de B.

Para obter os modelos equivalentes de G_1 e G_2 em Δ, basta aplicar o mapa Π^{-1}, tal que $G_a^1 = \Pi^{-1}(G_1)$ e $G_a^2 = \Pi^{-1}(G_2)$. Esses modelos são apresentados na Figura 13.

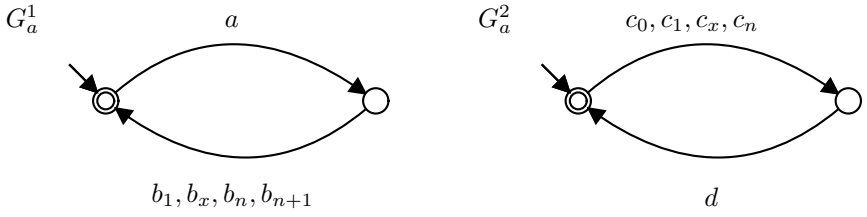

Figura 13 – Modelos para as plantas G_a^1 e G_a^2

O resultado do mapa Π^{-1} sobre autômatos pode ser obtido pelo

simples fato de substituir cada máscara, em cada transição, pelo seu respectivo conjunto de refinamentos.

O modelo global refinado é dado pela composição $G_a = G_a^1 \parallel G_a^2$, ou ainda, $G_a = \Pi^{-1}(G)$. Porém, G_a é dito ser um modelo ambíguo, ou não-determinístico, em relação a escolha de qual evento refinado deve ser habilitado dada a ocorrência de sua máscara. Tal inconveniente pode ser solucionado associando-se o modelo G_a a H_{d} a fim de eliminar o não-determinismo de G_a. Assim, resultando no modelo da planta distinguida, denotada por $G_d = \Pi^{-1}(G) \parallel H_{\mathrm{d}}$.

Agora, resta sintetizar uma solução ótima para o PCS-D para o exemplo proposto. Essa solução pode ser obtida usando o mesmo o algoritmo apresentado na seção 2.3, considerando os seguintes modelos:

(i) a planta distinguida $G_d = G_a \parallel H_{\mathrm{d}}$, tal que $L(G_d) = \Pi^{-1}(L(G)) \cap L_{\mathrm{d}}$;

(ii) a especificação $E_{\mathrm{d}} = E_{\mathrm{d}}^U \parallel E_{\mathrm{d}}^O$;

(iii) o comportamento desejado $K_d = E_{\mathrm{d}} \cap L^{\omega}(G_d)$.

Como resultado, a solução ótima é o supervisor $S_d = \mathrm{sup}\mathcal{C}(K_d, G_d)$, modelado por um autômato V_d tal que $L^{\omega}(V_d) = S_d$. A Tabela 1 apresenta uma comparação entre os modelos usados para síntese do PCS e PCS-D, e suas respectivas soluções. Essa comparação é contextualizada em número de estados e transições que compõem cada modelo.

Tabela 1 – Resultado da síntese para o PCS e PCS-D - estados (transições)

G	E	K	V
$4\ (8)$	$n+1\ (2n)$	$4n+4\ (8n+4)$	$4n+2\ (8n)$
G_{d}	E_{d}	K_{d}	V_{d}
$4n+4\ (8n+8)$	$4\ (20)$	$4n+4\ (8n+4)$	$4n+2\ (8n)$

Como pode ser analisado, as especificações são modeladas em Δ por um número de estados constante, independente do tamanho do *buffer*,

enquanto que, em Σ, essa complexidade cresce em função da capacidade do *buffer*. Por outro lado, observe que tal simplificação não se propaga para a etapa de síntese. De fato, os modelos K e K_d, bem como os supervisores (V e V_d), possuem o mesmo número de estados. Isso se deve ao fato de que, na abordagem com distinguidores, os ganhos obtidos na simplificação da especificação são compensados pela adição do modelo do distinguidor na planta. Em geral, é esperado que a complexidade de síntese seja equivalente.

Sumarizando, pode-se dizer que as vantagens do uso de distinguidores, como apresentados até aqui, se resumem a simplificar a modelagem de especificações complexas sem alterar o resultado de síntese. Tal simplificação pode ser decisiva para viabilizar a solução de problemas complexos de controle. No entanto, também é verdade que o uso direto de distinguidores não provê economias computacionais no processo de síntese, o que seria esperado de uma abordagem focada em simplificação de modelos. Uma alternativa para explorar vantagens computacionais de modelos distinguidos é apresentada a seguir e sustenta a principal proposta deste trabalho.

2.4.3 APROXIMAÇÕES NO PCS-D

No sentido de propagar os benefícios providos pelo uso de um distinguidor para o processo de síntese, o conceito de *aproximação* é proposto em Cury *et al.* (2015). Em determinados casos, uma aproximação pode levar a uma solução de controle computada com menos ônus computacional preservando, ao mesmo tempo, os benefícios de um distinguidor.

A ideia básica por trás de uma aproximação é fazer o uso de especificações modeladas em Δ, mas substituir a planta distinguida G_d por uma aproximação G_a, com menor número de estados. A construção de um G_a mais simples do que G_d implica, inevitavelmente, em usar um modelo de planta que distingue menos do que G_d, o que pode impactar no processo de síntese

e requer uma sistemática própria.

Em Cury *et al.* (2015) os autores sistematizam a construção de G_a como se segue. Sejam D_{a} e D dois distinguidores tais que $L_{\mathrm{d}} \subseteq L_{\mathrm{da}} \subseteq \Delta^*$ e tal que D é preditível. Por inclusão, todas as cadeias que sobrevivem em L_{d} também sobrevivem em L_{da}. A diferença é que L_{da} pode talvez habilitar mais cadeias do que aquelas em L_{d}. É imaginável que, tendo que distinguir menos, um autômato H_{da} modelando L_{da} seja mais simples do que H_{d} modelando L_{d}, em termos de número de estados.

Assim, quando aplicado a uma planta G, D_{a} produz uma aproximação $G_a = D_{\mathrm{a}}(G)$ para $G_d = D(G)$, com $L(G_a) = D_{\mathrm{a}}(L(G))$ e $L^\omega(G_a) = D_{\mathrm{a}}(L^\omega(G))$.

Resta definir, então, como modelar um distinguidor H_{da} que modele L_{da} de maneira tal que possa ser associado à planta G para produzir uma aproximação que leve a um processo de síntese computacionalmente vantajoso, preservando-se a otimidade do controlador. Nesse sentido, observe que H_{d} é modular, por construção, como ilustrado no exemplo anterior. Então, qualquer composição de sub-modelos de H_{d} leva a um modelo de distinguidor H_{da} que modela D_{a}, uma vez que $L_{\mathrm{d}} \subseteq L_{\mathrm{da}}$. Ou seja, a construção de H_{da} pode se dar simplesmente desconsiderando partes do modelo H_{d}.

Um caso particular que respeita a inclusão $L_{\mathrm{d}} \subseteq L_{\mathrm{da}}$ e que define um limite da aproximação é quando $H_{\mathrm{da}} = H_{\mathrm{d}}$, o que implica na igualdade L_{d} e L_{da}. Naturalmente, esse caso remete a uma aproximação que não provê simplificação na síntese, ainda que preserve os benefícios de modelagem. Nesse caso, G_a é um modelo com o mesmo número de estados de G_d. No limite oposto, $H_{\mathrm{da}} = H^{\Delta}$ implica que $L_{\mathrm{da}} = \Delta^*$, o que remete a uma aproximação possivelmente vantajosa em síntese. Nesse caso, $G_a = \Pi^{-1}(G)$ é um modelo com o mesmo número de estados de G.

Na prática, o processo de síntese envolvendo aproximações parte do caso em que $G_a = \Pi^{-1}(G)$ uma vez que esse tende a ser mais vantajoso, i.e.,

a aproximação é a versão mais ¨relaxada¨ possível de G_d. No entanto, pode ser mostrado (CURY *et al.*, 2015) que, em certos casos, tal aproximação pode não levar a uma solução ótima de controle. Para esses casos, aproximações intermediárias podem ser obtidas através da composição incremental de qualquer módulo de H_{d}, até que resulte na solução ótima. Em (AGUIAR *et al.*, 2013) é apresentada uma heurística que sugere quais partes do distinguidor são necessárias à síntese.

Mostra-se agora como uma solução para PCS-D pode ser encontrada usando uma aproximação G_a para G_d.

Dados G_d e E_{d}, como no PCS-D, seja G_a uma D-aproximação para G_d. Para $K_a = E_{\mathrm{d}} \cap L^{\omega}(G_a)$, se $\sup\mathcal{C}(K_a, G_a)$ e L_{d} forem não-conflitantes, então um supervisor S_a, implementando $\sup\mathcal{C}(K_a, G_a) \cap L_{\mathrm{d}}$, é uma solução para o PCS-D, tal que

$$\sup\mathcal{C}(K_a, G_a) \cap L_{\mathrm{d}} \subseteq \sup\mathcal{C}(K_d, G_d).$$

Embora S_a seja uma solução controlável e não-bloqueante para o PCS-D, essa solução pode ser mais restritiva do que a solução $\sup\mathcal{C}(K_d, G_d)$ (caso da inigualdade). Essa restrição é derivada do fato de que, com a aproximação, o processo de síntese dispõe de menos informações sobre a planta, i.e., as informações contidas nos eventos podem em geral ser ambíguas, o que leva o supervisor a se precaver de sequências que possam interferir na controlabilidade e/ou no não-bloqueio.

Para estimar quando uma solução é mais restritiva, basta comparar o número de estados dos modelos que implementam $\sup\mathcal{C}(K_a, G_a) \cap L_{\mathrm{d}}$ e $\sup\mathcal{C}(K_d, G_d)$. No entanto, para isso, é necessário que se calcule efetivamente $\sup\mathcal{C}(K_d, G_d)$, o que pode não ser viável computacionalmente. Aliás, uma das finalidades das aproximações é justamente evitar tal cálculo. Assim, uma forma de verificar se a solução implementada por S_a é ótima, sem a necessidade de se calcular $\sup\mathcal{C}(K_d, G_d)$, é definindo-se um limite superior para $\sup\mathcal{C}(K_d, G_d)$,

calculado com o mesmo esforço computacional da síntese com aproximações. Assim, basta testar a igualdade entre $\sup\mathcal{C}(K_a, G_a) \cap L_{\mathrm{d}}$ e esse limite superior (CURY *et al.*, 2015). Quando ela procede, então é verdade que

$$\sup\mathcal{C}(K_a, G_a) \cap L_{\mathrm{d}} = \sup\mathcal{C}(K_d, G_d).$$

Nesse trabalho, como o foco principal é sobre a implementação e não sobre os aspectos de síntese, a otimidade de S_a é testada em relação a igualdade direta com $\sup\mathcal{C}(K_d, G_d)$. Caso S_a venha a ser uma solução ótima para PCS-D, então ela pode ser usada para fins de implementação, i.e., os modelos V_a e H_{d} são compostos para representar $\sup\mathcal{C}(K_a, G_a) \cap L_{\mathrm{d}}$.

2.4.4 EXEMPLO DE APROXIMAÇÕES EM PCS-D

Considere o exemplo do *buffer* que vem sendo proposto ao longo deste trabalho. No sentido de obter uma solução para o PCS-D que reflita no maior ganho computacional na síntese, o modelo do distinguidor D_{a} é considerado inicialmente como $H_{\mathrm{da}} = H\Delta$, tal que $L_{\mathrm{da}} = \Delta^*$.

Nesse caso, o efeito de D_{a} sobre a planta G é obtido pela substituição de cada máscara pelo seu respectivo conjunto de refinamentos em Δ, i.e., $G_a = \Pi^{-1}(G)$. Assim, o comportamento desejado aproximado é dado por $K_a = E_{\mathrm{d}} \cap L^{\omega}(G_a)$. O restante do processo de síntese de controle com aproximação é análogo ao apresentado na Seção 2.4.2.

O resultado desse processo é o elemento supremo $\sup\mathcal{C}(K_a, G_a)$, modelado por um autômato V_a. Assim, a ação de controle S_d é implementada pelos autômatos não-conflitantes V_a e H_{d}. A Tabela 2 compara, em termos de número de estados e transições, a síntese de controle para o exemplo do buffer, envolvendo as três abordagens apresentadas anteriormente.

As duas primeiras linhas reproduzem os resultados apresentados anteriormente para o PCS e o PCS-D, enquanto a terceira linha aplica a

Tabela 2 – Resultado da síntese para o PCS, PCS-D e PCS-D com aproximações - estados (transições)

G	E	K	V
$4\ (8)$	$n+1\ (2n)$	$4n+4\ (8n+4)$	$4n+2\ (8n)$
G_{d}	E_{d}	K_{d}	V_{d}
$4n+4\ (8n+8)$	$4\ (20)$	$4n+4\ (8n+4)$	$4n+2\ (8n)$
G_{a}	E_{d}	K_{a}	V_{a}
$4\ (20)$	$4\ (20)$	$16\ (72)$	$12\ (47)$

aproximação $G_a = \Pi^{-1}(G)$ sobre G_d.

Note que, em termo do espaço de estados, G_{a} é equivalente a G, e são mais simples que G_{d}. Além disso, E_{d} é mais simples do que E, e por conseguinte a síntese de V_{a} é mais simples que V_{d} e V.

Quanto ao número de transições, a síntese de V_{a} não depende do tamanho do *buffer*. Na verdade, o tamanho do *buffer* está associado ao modelo H_{d}, que foi removido da síntese. Portanto, o supervisor V_{a} será o mesmo (e ainda será obtido com o mesmo esforço computacional) para qualquer tamanho do *buffer* (CURY *et al.*, 2015).

3 RESULTADOS

Apesar das vantagens, a síntese aproximada resulta no supervisor que é a versão parcial do controlador. De fato, para obter a versão final do controlador de G, V_{a} tem que ser associado a H_{d} a fim de se distinguir como os refinamentos ocorrem em G. Essa associação pode ser obtida pela composição síncrona. Nesse caso, $V_{\mathrm{a}} \parallel H_{\mathrm{d}}$ leva a um modelo com $4n + 2$ estados e $8n$ transições, que é tão complexo quanto V_{d}.

Contudo, isso é uma solução implementável que inevitavelmente depende de n, tal que os benefícios das aproximações ficam limitados a fase de síntese. A Tabela 3 ilustra esse raciocínio para dois *buffers* com 5 e 50 posições.

Tabela 3 – Comparação do uso de memória

Buffer com 5 posições					
G_{a}^{5}	H_{d}^{5}	E_{d}^{5}	K_{a}^{5}	V_{a}^{5}	$V_{\mathrm{a}}^{5} \parallel H_{\mathrm{d}}^{5}$
$4\ (20)$	$6\ (12)$	$4\ (20)$	$16\ (72)$	$12\ (47)$	$22\ (40)$
Buffer com 50 posições					
G_{a}^{50}	H_{d}^{50}	E_{d}^{50}	K_{a}^{50}	V_{a}^{50}	$V_{\mathrm{a}}^{50} \parallel H_{\mathrm{d}}^{50}$
$4\ (20)$	$51\ (102)$	$4\ (20)$	$16\ (72)$	$12\ (47)$	$202\ (400)$

Observe que, os *buffers* com 5 ou 50 (ou n) posições levam ao mesmo supervisor aproximado com 12 estados e 47 transições. No entanto, as soluções implementáveis (última coluna) incrementam o uso de memória substancialmente, de 22 para 202 estados, e de 40 para 400 transições.

Alternativamente, V_{a} e H_{d} poderiam ser implementados em estruturas diferentes, devidamente interligadas e se comunicando. Isso de fato, removeria H_{d} tanto da síntese quanto da implementação. Embora, nesse caso o modelo de H_{d} ainda seria dependente de n. Nesse contexto, este trabalho propõe:

- uma arquitetura que permita separar V_{a} e H_{d};

- uma estrutura de comunicação entre V_{a} e H_{d};
- uma implementação genérica para H_{d}, composta por dois estados.

Dessa forma, espera-se estender os ganhos trazidos pelas aproximações na síntese para a fase de implementação.

3.1 ARQUITETURA PROPOSTA

Seja H_{d} um modelo para um distinguidor preditível D, tal que $L(H_{\mathrm{d}}) = L_{\mathrm{d}}$ e H_{da} um modelo para um distinguidor D_{a}, tal que $L(H_{\mathrm{da}}) = L_{\mathrm{da}}$, com $L_{\mathrm{d}} \subseteq L_{\mathrm{da}}$. Seja também S_{a} um modelo para um supervisor $\mathrm{sup}\mathcal{C}(K_{\mathrm{a}}, G_{\mathrm{a}})$ obtido usando uma aproximação $G_{\mathrm{a}} = D_{\mathrm{a}}(G)$ para $G_{\mathrm{d}} = D(G)$. A fim de implementar a solução $\mathrm{sup}\mathcal{C}(K_{\mathrm{a}}, G_{\mathrm{a}}) \cap L_{\mathrm{d}}$ de forma descentralizada, se propõe a arquitetura apresentada na Figura 14.

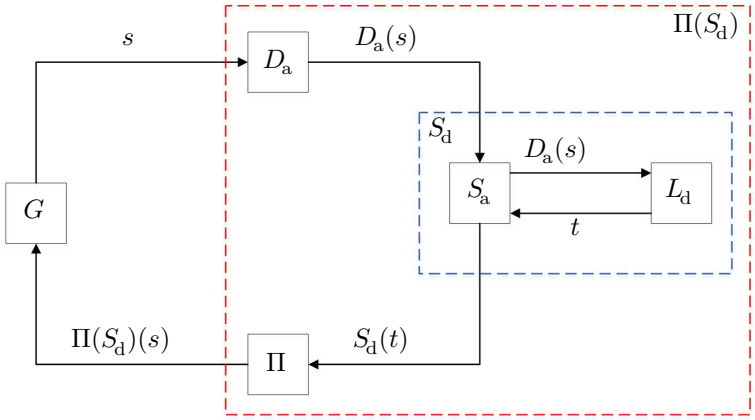

Figura 14 – Implementação descentralizada de controladores

Após uma cadeia $s \in L(G)$, a planta do sistema G espera o sistema de controle S_{d} responder com um conjunto de eventos habilitados, de modo que a planta possa se comportar de acordo com a ação de controle. Como o sistema de controle é tratado em Δ, e o modelo da planta reconhece eventos em Σ, então a planta espera receber uma resposta de $\Pi(S_{\mathrm{d}})$.

Agora a questão é como reproduzir tal resposta para G,

considerando que não se tem realmente o supervisor S_{d} e não se tem a intenção de implementar a composição equivalente a $\mathrm{sup}\mathcal{C}(K_{\mathrm{a}}, G_{\mathrm{a}}) \cap L_{\mathrm{d}}$. Essa questão pode ser respondida pela comunicação das duas estruturas concorrentes, S_{a} e L_{d}.

Note que, após uma cadeia $s \in L(G)$, o único distinguidor que se pode consultar é D_{a}, que foi utilizado para calcular S_{a}. Como D_{a} não é necessariamente preditível, i.e, $|D_{\mathrm{a}}(s)| \geq 1$, $D_{\mathrm{a}}(s)$ pode retornar um conjunto de refinamentos para a mesma fonte de evento em Σ.

Isso significa que S_{a} pode receber de $D_{\mathrm{a}}(s)$ eventos cuja elegibilidade não podem ser decidida. O supervisor S_{a} poderia, por exemplo, receber dois eventos, α_1 e α_2, com a mesma máscara α em G, e habilitar somente α_1. Nesse caso, a elegibilidade de α em G ainda seria incerta, a menos que se consulte L_{d}. Em resumo, para um evento sobreviver sob controle, ele deve ser habilitado por S_{a} e L_{d}, não necessariamente nesta ordem.

A caixa interna tracejada da Figura 14 implementa essa verificação e será detalhada na Seção 3.2. Essencialmente, ela reproduz o mesmo efeito que S_{d} devido a interação entre S_{a} e L_{d}. Isso evita que S_{a} e L_{d} sejam compostos, e o resultado dessa interação é um conjunto de eventos $D(s)$ que contem escolhas preditíveis para cada evento habilitado por $D_{\mathrm{a}}(s)$. Se tais eventos ainda sobreviverem sob o controle de S_{a}, então eles também pertencem a $S_{\mathrm{d}}(t)$, cuja projeção $\Pi(S_{\mathrm{d}}(t))$ de fato ativará as respectivas fontes de eventos em G.

3.2 INTEGRANDO S_{a} E L_{d}

Nesta seção será apresentado como S_{a} e L_{d} podem ser integrados de tal modo que suas ações disjuntas levem ao mesmo comportamento que suas composições. Dessa forma, espera-se que eles possam ser implementados separados. O Algoritmo 1 sintetiza esta ideia.

Após cada cadeia $s \in L(G)$, o supervisor S_{a} verifica se os

Algorithm 1 Implementação disjunta de S_{d}

Input: $\sigma \in \Sigma$; $s \in \Sigma^*$; $t \in \Delta^*$; distinguisher D_{a}; supervisor S_{a};
1: $t \leftarrow \epsilon$;
2: **while** $L(G)(s) \neq \emptyset$ **do**
3: **if** $|S_{\mathrm{a}}(D_{\mathrm{a}}(s)) \cap \Delta^{\sigma}| > 1$ **then**
4: $t \leftarrow REQUESTL_{\mathrm{d}}(D_{\mathrm{a}}(s))$;
5: **else**
6: $t \leftarrow D_{\mathrm{a}}(s)$;
7: **return** $S_{\mathrm{a}}(t)$;
8:
9:
10: **function** REQUEST$L_{\mathrm{d}}(D_{\mathrm{a}}(s))$
11: **return** $D_{\mathrm{a}}(s) \cap L_{\mathrm{d}}$;

refinamentos elegíveis para cada evento $\sigma \in \Sigma$, habilitados por S_{a}, são preditíveis (linha 3). Caso sejam, a versão distinguida da cadeia s é t (caso else, linha 6), i.e., $t = D_{\mathrm{a}}(s) = D(s)$.

Caso contrário, i.e, se de fato existe $\sigma \in \Sigma$ que para Δ^{σ} em S_{a} é ambígua (verifica na linha 3), então H_{d}, com $L(H_{\mathrm{d}}) = L_{\mathrm{d}}$, tem que ser requisitado (linha 4) a fim de distinguir qual a instância exata $\delta \in \Delta^{\sigma}$ deve sobreviver. Essa distinção preditível permitirá então S_{a} decidir inequívocamente se a fonte do evento σ deve ser habilitado ou não em G (linha 7).

Note que, quando S_{a} é consultado para definir a necessidade de chamar H_{d} (linha 3), a requisição em si (linha 4) é executada informando como entrada o conjunto refinado Δ^{σ} para todo os candidatos (ambíguos). Essa otimização evita, por exemplo, requisições desnecessárias a H_{d} quando, depois de s, S_{a} desabilita todos eventos em Δ^{σ}, para $|\Delta^{\sigma}| > 1$.

Observe que, embora H_{d} esteja separado de S_{a}, eles ainda executam concorrentemente, com H_{d} a ser consultado sempre que S_{a} necessitar. Isso sugere que H_{d} e S_{a} poderiam ser implementados em hardwares diferentes e ainda desempenhariam a mesma função para o sistema de controle. Mais do que isso, uma vez que S_{a} detém a decisão sobre requisitar ou não H_{d}, logo, essa é uma clássica arquitetura de comunicação mestre-escravo. A seguir, será ilustrado esta ideia pela implementação de S_{a} e H_{d} para o exemplo proposto ao

longo desse trabalho.

3.2.1 IMPLEMENTADO UM SISTEMA DE CONTROLE PARA O EXEMPLO

O primeiro passo para implementar um sistema descentralizado de controle distinguido é implementar o supervisor e distinguidor. Para o exemplo do *buffer*, foram selecionados dois microcontroladores da *Texas Instruments* (INSTRUMENTS, 2016), com memória suficiente para suportar cada estrutura. O autômato V_{a} (supervisor) foi implementado em um *Tiva C Series TM4C1294NCPDT*. Para o autômato H_{d} (distinguidor), a forma como foi implementado neste trabalho, foi suficiente um *MSP430G2553*.

A tradução dos autômatos para a linguagem C, suportada pelos microcontroladores, foi conduzida com a ajuda da ferramenta *Deslab* (TORRICO, 2016). A seguir, serão fornecido mais detalhes sobre cada implementação, bem como uma opção para comunicá-los.

3.2.1.1 SUPERVISOR

A tradução de V_{a} para a linguagem C conduz para uma estrutura que simplesmente reflete uma máquina de estados na forma de *switch-cases*. A diferença, nesse caso, é que tal máquina de estados (supervisor) depende de informações adicionais que são fornecidas por outra máquina de estados remota (distinguidor). Assim, a comunicação correta entre S_{a} e L_{d} torna-se crucial para a arquitetura de controle proposta.

A comunicação entre S_{a} e L_{d} é realizada através de uma arquitetura mestre-escravo, que foi implementada utilizando um microcontrolador *Tiva* para o mestre (supervisor) e outro microcontrolador *MSP* para o escravo (distinguidor). O código fonte da comunicação é disponibilizado em (ROSA, 2016) e foi desenvolvido utilizando o protocolo *Modbus* no modo de transmissão *RTU*, amplamente utilizado em redes industriais. A ideia por trás da

implementação é sumarizada pelo Algoritmo 2.

Algorithm 2 Comunicação Mestre-Escravo

Input: $D_{\mathrm{a}}(s)$;
1: **function** COMMUNICATION($D_{\mathrm{a}}(s)$)
2: Set package pkg as empty;
3: $pkg \leftarrow constructMsg$WRT($D_{\mathrm{a}}(s)$);
4: $pkg \leftarrow transmit(pkg)$;
5: $pkg \leftarrow constructMsg$RD();
6: $pkg \leftarrow transmit(pkg)$;
7: $t \leftarrow process(pkg)$;
8: **return** $t \in \Delta^*$;

O supervisor (mestre) constrói um pacote Modbus usando a função padrão 16 (*Write Multiple Registers*) (linha 3), e envia ao distinguidor (escravo) (linha 4). O escravo, por sua vez, deve responder tal mensagem a fim de informar que os dados foram recebidos e armazenados em seus registradores (linha 4). O nó mestre só pode continuar se esta resposta é entregue com sucesso.

Em seguida, o mestre prepara outro pacote (linha 5) usando a função padrão 3 (*Read Holding Registers*) (linha 5), visando ler os dados que o escravo escreveu em seus registradores, que são na verdade a distinção produzida para a mensagem recebida anteriormente. Esse pacote é transmitido ao escravo (linha 6) e, como resposta, o mestre aguarda receber t, i.e, a versão distinguida da cadeia s.

Resta discutir opções para implementar H_{d} no *hardware* escravo, de modo que a resposta $t \in \Delta^*$ possa ser gerada ao mestre. Na Seção **??**, foi argumentado que a simples geração da versão de *switch-cases* levaria a uma implementação que cresce em função do tamanho do *buffer*. A seguir é proposto a implementação de uma estrutura genérica composta por dois estados e quatro transições, que implementa o comportamento de H_{d} para um *buffer* de qualquer tamanho.

3.2.1.2 DISTINGUIDOR

Considere modularizar H_{d} como na Figura. 15.

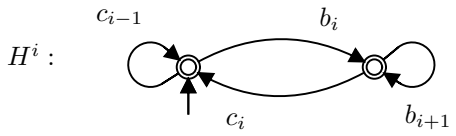

Figura 15 – Versão modular do distinguidor $H_{\mathbf{d}} = \|_{i=1}^{n} H^i$

Em comparação ao modelo do distinguidor da Figura 11, H^i expressa a mesma ideia, mas de uma maneira diferente. Para que tal modularização funcione, os refinamentos dos eventos b e c precisam mapear cada posição do *buffer* usando um evento distinto, diferente do que é feito na Figura 11, onde todas as posições intermediárias de 2 a $n-1$ são representadas pelo mesmo evento.

A abordagem da Figura 11 é útil para síntese, uma vez que ela remove do supervisor a responsabilidade de carregar as transições intermediárias do *buffer*, e a transfere para H_{d}, que não participa da síntese. No entanto, para implementar a ação de controle, H_{d} tem que ser abordado de qualquer maneira. Contudo, se as transições do *buffer* são rotuladas por um mesmo evento, isso significa que o espaço de estados terá de memorizar todas as posições do *buffer*, aumentando o consumo de memória. Em contrapartida, no caso de cada rótulo de transição ser distinto dos outros, o mesmo comportamento pode ser expresso por um modelo igual ao da Figura 15.

Considere os eventos c_x e b_x como na Figura 11, de modo que agora passem a ser representados por c_i e b_i, para $i = 2, \cdots, n-1$, tal que $\Delta^b = \{b_1, b_i, b_n, b_{n+1}\}$ e $\Delta^c = \{c_0, c_1, c_i, c_n\}$.

O modelo do distinguidor pode ser implementado por um único autômato composto por dois estados e quatro transições como na Figura 15. De fato, a estrutura de transições de estados responsável por distinguir

cada posição do *buffer* sempre será simétrica. Essa característica pode ser explorada a fim de manter na memória apenas um autômato genérico, rotacionando apenas os rótulos associados as transições em cada interação do *buffer*.

3.3 ASPECTOS DE IMPLEMENTAÇÃO

Apesar da implementação do modelo H^i ser particular para o exemplo proposto nesse trabalho, acredita-se que isso possa ser generalizado para qualquer distinguidor.

Além disso, é válido notar que, embora seja utilizado um número fixo de estados e transições para implementar o modelo H^i, ainda é necessário manter na memória um vetor para armazenar o estado atual de cada módulo i. Mesmo assim, estima-se que isso seria substancialmente mais barato do que implementar toda a sua composição (Figura 11).

Ilustrando essa ideia em termos do consumo de memória para representar cada modelo, tem-se que em uma arquitetura de 16 *bits* a versão modular (H^i) consome $28+n$ *bytes*, enquanto sua composição $14n+4$ *bytes*. No caso modular, o consumo de *bytes* associado ao tamanho do *buffer* pode ser reduzido, uma vez que n é o vetor de estados atuais. Uma alternativa para isso seria armazenar o estado atual de cada modulo em um *bit*, já que os estados possíveis são 0 ou 1, o que culminaria em uma economia na ordem de 8 vezes.

Também vale lembrar que a arquitetura de comunicação proposta nesse trabalho inevitavelmente impõe uma latência na ação de controle. A fim de estimar o *overhead* adicionado na malha de controle, e para inferir se isso deve ou não ser uma preocupação, foram realizados testes para dimensionar o tempo da comunicação utilizando um *baudrate* de 57600. Os resultados são apresentados na Tabela 4.

Os experimentos foram realizados em uma instância do exemplo

Tabela 4 – Delay da comunicação mestre-escravo

Clock (MHz)	Time(μs)
120	$206,0$
60	$206,0$
32	$208,0$
16	$210,0$
2	$299,1$

proposto e se referem à média de um conjunto representativo (fixo) de requisições. Para cada configuração, o *clock* foi fixado no escravo em $\approx 16MHz$, enquanto no mestre ele foi variado. Como pode ser visto, nem o pior caso do *overhead* ($299,1$ μs) é suficientemente representativo para comprometer a malha de controle. Na verdade, o espaço de tempo entre as ocorrências dos eventos é suficientemente grande para comunicar supervisor e distinguidor várias vezes e, portanto, isso é irrelevante para ação de controle.

Dessa forma, entende-se que a arquitetura proposta é um boa alternativa para implementar controladores para SEDs, uma vez que, além de beneficiar a síntese, a implementação pode ser conduzida com economia de hardware. Por economia de hardware refere-se a um menor consumo de memória, que geralmente, está associado a dispositivos mais baratos e de baixo consumo de energia. Em outras palavras, essa abordagem permite obter vantagens na implementação de controladores refinados, em termos de memória, recursos e consumo de energia, além tornar a manutenção do sistema de controle (modular) mais prática e confiável.

4 CONCLUSÃO

Este trabalho investigou a implementação de controladores refinados para SEDs. Em vez de compor supervisor e distinguidor para obter o sistema de controle, como de praxe, este trabalho sugere que eles podem ser implementados separados, em dois hardwares diferentes que se comuniquem entre si, de tal forma que a ação de controle resultante seja a mesma que a versão centralizada, enquanto o custo da implementação é reduzido. Essa ideia foi ilustrada por meio da arquitetura proposta na Seção 3.1, e uma opção de implementação disso é sugerida na seção seguinte.

Outro aspecto que pode ser explorado a fim de beneficiar a etapa de implementação de controladores refinados é a simetria entre os modelos que compõem um distinguidor, de tal forma que seja necessário implementar um único módulo e os demais sejam obtidos pela rotação do conjunto de eventos associados a cada transição. Neste trabalho, essa ideia foi implementada para um caso particular mas acredita-se que tal fato possa ser generalizado para qualquer distinguidor.

Um exemplo de controle para um *buffer* foi implementado a fim de ilustrar a abordagem proposta.

A principal vantagem trazida por este trabalho é propagar os ganhos provenientes do uso dos eventos refinados nos modelos de SEDs, da síntese para fase de implementação.

Além disso, considerando que tanto supervisor quanto distinguidor podem ser modulares, a abordagem proposta traz o benefício adicional de permitir estender facilmente a estrutura de comunicação para qualquer número de módulos descentralizados, o que realmente compõe algumas perspectivas de pesquisas futuras.

REFERÊNCIAS

AGUIAR, R. S. S.; CUNHA, A. E. C.; CURY, J. E. R.; QUEIROZ, M. H. Heuristic search of supervisors by approximated distinguishers. In: **IFAC Workshop on Dependable Control of Discrete Systems (DCDS'13)**. York, England: [s.n.], 2013. p. 121–126.

BOUZON, G.; QUEIROZ, M. H. de; CURY, J. E. R. Supervisory control of des with distinguishing sensors. In: **International Workshop on Discrete Event Systems, WODES'08**. Gothenburg, Sweden: [s.n.], 2008. p. 22–27.

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

CUNHA, A. E. C. da; CURY, J. E. R. Hierarchical supervisory control based on discrete event systems with flexible marking. **IEEE Transactions Automatic Control**, v. 52, p. 2242–2253, Dec. 2007.

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

CURY, J. E. R.; QUEIROZ, M. H. de; BOUZON, G.; TEIXEIRA, M. Supervisory control of discrete event systems with distinguishers. **Automatica**, v. 56, p. 93 – 104, 2015.

HAYKIN, SIMON; VEEN, Barry Van. **Sinais e sistemas**. [S.l.]: Bookman, 2001.

INSTRUMENTS, Texas. 2016. Disponível em: <http://www.ti.com/>.

OGATA, Katsuhiko. **Engenharia de controle moderno**. 5. ed. São Paulo: Pearson Prentice Hall, 2010.

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

ROSA, M. **Decentralized Implementation of Distinguished Controllers**. [S.l.], 2016. Disponível em: <https://github.com/marcellorosa/dcds2017>.

TEIXEIRA, M. **Explorando o uso de distinguidores e de autômatos finitos estendidos na teoria do controle supervisório de sistemas a eventos discretos**. 137 p. Tese (Doutorado) — Universidade Federal de Santa, 2013.

TEIXEIRA, M.; MALIK, R.; CURY, J.E.R.; QUEIROZ, M.H. de. Supervisory control of des with extended finite-state machines and variable abstraction. **Automatic Control, IEEE Transactions on**, v. 60, n. 1, p. 118–129, 2014.

TEIXEIRA, M.; RIBEIRO, R.; BARBOSA, M.; ENEMBRECK, F.; MASSA, R. A modeling architecture for the orchestration of service components in factory automation. In: **IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'15**. Luxemboug, Luxemboug: [s.n.], 2015.

TORRICO, C. **Deslab 3.6**. [S.l.], 2016. Disponível em: <https://goo.gl/l4bD2i>.

WONHAM, W.M. **Notes on Discrete Event Systems**. 2002. University of Toronto.