Research, Society and Development, v. 11, n. 3, e29511326615, 2022
(CC BY 4.0) | ISSN 2525-3409 | DOI: http://dx.doi.org/10.33448/rsd-v11i3.26615

Estudo de caso de controle supervisório tolerante a perdas intermitentes de observação: Planta MECATRIME

Supervisory control case study tolerant to intermittent observation losses: MECATRIME Plant

Estudio de caso de control de supervisión tolerant a pérdidas de observación intermitente: Planta MECATRIME

Recebido: 06/02/2022 | Revisado: 12/02/2022 | Aceito: 15/02/2022 | Publicado: 22/02/2022

Rárisson Queiroz Hilário
ORCID: https://orcid.org/0000-0001-6001-0233
Instituto Militar de Engenharia, Brasil
E-mail: rarimhilario@gmail.com
Antonio Eduardo Carrilho da Cunha
ORCID: https://orcid.org/0000-0001-5772-2790
Instituto Militar de Engenharia, Brasil
E-mail: carrilho@ime.eb.br

Resumo
Neste artigo apresenta-se um estudo de caso de controle supervisório tolerante a perdas intermitentes de observação considerando a existência da perda de comunicação entre dispositivos interconectados à planta via LAN (rede local ETHERNET), em um sistema de manufatura flexível existente no Laboratório de Mecatrônica do Instituto Militar de Engenharia (planta MECATRIME). Este estudo teve como objetivo aplicar conceitos de controle supervisório quando há ocorrência de perdas intermitentes de observação quando há falha de comunicação e assim permitir que o sistema continue atuando. Inicialmente, foi feito um mapeamento da planta MECATRIME, em que foram construídos diagramas baseados na Engenharia de Sistemas com o intuito de identificar os pontos ou situações em que há a possibilidade de perda de observação em função da perda de comunicação e, após esse mapeamento, foi realizada a modelagem do sistema utilizando diagramas de máquinas de estado, na forma de autômatos, com a perda intermitente de observação. Foi feita também a síntese do supervisor, mostrando como uma ferramenta formal pode ser utilizada para guiar a realização do projeto de um sistema de controle tolerante a perdas de observação por comunicação.
Palavras-chave: Sistemas a eventos discretos; Controle supervisório; Perdas intermitentes de observação; Sistema de manufatura; Modelagem de sistemas.

Abstract
This paper presents a case study of supervisory control tolerant to intermittent loss of observation considering the existence of loss of communication between devices interconnected to the plant via LAN (local ETHERNET network), in a flexible manufacturing system existing at the Mechatronics Laboratory of the Military Engineering Institute (MECATRIME plant). This study aimed to apply supervisory control concepts when there are intermittent losses of observation when there is a communication failure and thus allow the system to continue acting. Initially, a mapping of the MECATRIME plant was made, in which diagrams based on Systems Engineering were constructed in order to identify the points or situations where there is the possibility of loss of observation due to the loss of communication and, after this mapping, system modeling was performed using state machine diagrams, in the form of automata, with intermittent loss of observation. The supervisor's synthesis was also made, showing how a formal tool can be used to guide the realization of the project of a control system tolerant to losses of observation by communication.
Keywords: Discrete event systems; supervisory Control; Intermittent loss of observation; Manufacturing system; Systems modeling.

Resumen
Este documento presenta un estudio de caso de control de supervisión tolerante a pérdidas intermitentes de observación considerando la existencia de pérdida de comunicación entre dispositivos interconectados a la planta a través de LAN (red local ETHERNET), en un sistema de fabricación flexible existente en el Laboratorio de Mecatrónica del Instituto de Ingeniería Militar (planta MECATRIME). Este estudio tuvo como objetivo aplicar conceptos de control supervisor cuando existen pérdidas de observación intermitentes cuando hay una falla de comunicación y así permitir que el sistema continúe actuando. Inicialmente, se realizó un mapeo de la planta MECATRIME, en el que se construyeron diagramas basados en Ingeniería de Sistemas para identificar los puntos o

situaciones en los que existe la posibilidad de pérdida de observación debido a la pérdida de comunicación y, después de este mapeo, El modelado del sistema se realizó utilizando diagramas de máquina de estado, en forma de autómatas, con pérdida intermitente de observación. También se realizó la síntesis del supervisor, que muestra cómo se puede utilizar una herramienta formal para guiar la realización del proyecto de un sistema de control tolerante a las pérdidas de observación por comunicación.
Palabras clave: Sistemas de eventos discretos; Control de supervision; Pérdidas de observación intermitente; Sistema de manufactura; Modelado de sistemas.

1. Introdução

Nos últimos anos, o desenvolvimento e a utilização de novas tecnologias cada vez mais avançadas e ágeis vem conduzindo a transformação da indústria a uma velocidade nunca observada. O efeito dessas transformações e a rapidez em que ela ocorre fez surgir um novo conceito, a Indústria 4.0, considerada a quarta Revolução Industrial. A Indústria 4.0 reflete a união de vários conceitos, e um deles é dos sistemas Ciber-Físicos (*Cyber-Phisycal Systems - CPS*) (Da Costa, 2017). Os CPS são sistemas físicos cujas operações são monitoradas e coordenadas via rede, controladas e integradas por um núcleo de computação e comunicação. Normalmente, os CPS consistem em um processo físico e um sistema cibernético e são compostos pelos sistema de monitoramento (sensores), pelas redes de comunicação, pela unidade de processamento, pelos atuadores e dispositivos de comunicação (Rajkumar et al., 2010; Wang et al., 2010; Cárdenas et al., 2008).

Sistemas industriais baseados no paradigma da Indústria 4.0 geralmente podem ser representados por Sistemas a Eventos Discretos (SED). Um SED é um sistema dinâmico dirigido a eventos com espaço de estados discreto (Cassandras et al., 2008). Com essa definição pode-se identificar que a evolução dos estados depende inteiramente da ocorrência de eventos assíncronos e discretos ao logo do tempo.

A Teoria de Controle Supervisório (TCS) (Ramadge et al., 1989), define formalmente a síntese de controladores para SED, permitindo controladores seguros e ótimos. Essa teoria utiliza a definição de controle em malha fechada (MF) no domínio do SED.

A malha fechada possui dois componentes, a planta (sistema a ser controlado), em geral corresponde a um conjunto de equipamentos ou máquinas, e o supervisor (agente de controle do sistema). A planta gera espontaneamente eventos que são observados pelo supervisor que, por sua vez, indica quais os eventos estão permitidos de ocorrer dada uma configuração da planta.

No controle supervisório de SED, o conceito de observação parcial aparece quando o supervisor não enxerga alguns dos eventos gerados pela planta. Na abordagem original de Ramadge et al., (1989), supõe-se que todos os eventos são observados, no sentido que há um sensor no sistema que gera a informação para cada evento modelado. Nas abordagens de controle com observação parcial (Wonham et al., 2017), a hipótese é que não existem sensores para alguns eventos e, portanto, supõe-se que os mesmos sejam não observáveis. Diante disso, o supervisor precisa tomar decisões de controle sem ter observado a ocorrência de alguns eventos, gerando então um problema de controle supervisório com observação parcial (Cassandras et al., 2008; Wonham et al., 2017).

Um conceito diferente de observação parcial surge com perdas intermitentes de observação (Carvalho et al., 2017; Alves et al., 2019). A diferença de uma falha intermitente para a observação parcial se dá pelo fato de que, na falha intermitente, existe o sensor que monitora a ocorrência de um evento, mas pode ser que a transmissão da ocorrência do evento não ocorra ou que uma falha no sensor cause a perda intermitente de observação.

O sistema de controle dos CPS está em rede, então, se não houver garantia de segurança ponta a ponta, pode ser que mesmo que um evento tenha ocorrido, a informação dessa ocorrência não tenha sido transmitida, perdida ou mesmo ser transmitida fora do tempo. Esse problema de controle é tratado pelo controle supervisório robusto sujeito a perdas

intermitentes de observação (Alves, 2014; Alves, 2017; Alves et al., 2014)

Nos trabalhos de Rohloff (2005); Sánches et al., (2006) e Rohloff (2012), foi considerado o problema de controle supervisório seguro mediante a falha em um sistema, onde foi suposto que poderia haver uma falha a qualquer instante e que após sua ocorrência, essa falha se tornaria permanente. Alves (2014) propôs a implementação de um supervisor que continue atuando no sistema mesmo que um evento perca a observação de maneira intermitente.

Em contrapartida, Lin (2014) investiga o controle de SED em rede, considerando o atraso na comunicação e as perdas de observação, sendo a comunicação entre o supervisor e a planta feita por meio de uma rede compartilhada e o sentido de comunicação se dá em duas direções: Planta → Supervisor e Supervisor → Planta. É aceitável considerar que a comunicação entre supervisor e planta é confiável e instantânea, se a distância de comunicação entre eles for relativamente curta e se a comunicação for realizada através de protocolos de comunicação dedicados.

Para SED em rede, essa suposição não é válida. Em sistemas de controle em rede, o controle é fechado através de uma rede em tempo real (comunicação) que é compartilhada com outros componentes dentro ou fora do sistema de controle. Há várias vantagens na utilização de sistemas de controle em rede, como por exemplo a facilidade de adicionar ou remover componentes em um sistema em rede, tornando-o mais flexível e ágil. Outra característica positiva é a economia de fiação utilizada entre os componentes, implicando na redução da complexidade e do custo do sistema.

A abordagem do Controle Tolerante a Falhas (CTF) consiste em garantir que a planta continue operando de maneira segura, evitando que ela seja interrompida por exemplo. O CTF pode ser abordado como ativo ou passivo (Paoli et al., 2011). Na abordagem ativa, com o diagnóstico ou prognóstico de uma falha, a lei de controle é adaptada ao comportamento do sistema quando este é cometido por algum tipo de falha, com o intuito de garantir que o objetivo do controle seja alcançado. Já na abordagem passiva, tem-se por objetivo encontrar um único controlador que atenda às especificações de controle tanto na operação nominal quanto após a ocorrência de falhas. O controlador encontrado na abordagem passiva, como em Alves et al. (2019), é obtido por meio das técnicas de controle robusto e ele deve assegurar que o sistema em MF continue insensível a algumas falhas sem que haja a necessidade de mudanças no controlador ou então na estrutura de controle.

Este artigo é estruturado da seguinte forma: a seção 2 apresenta uma breve revisão da teoria de SED e revê o conceito para síntese de um supervisor sujeito a perdas intermitentes de observação; na seção 3 é apresentada a planta *MECATRIME*, com sua composição, bem como o seu funcionamento e seus componentes. É apresentada também a construção dos diagramas utilizados na modelagem de uma estação da planta *MECATRIME*; na seção 4 são feitas considerações com o objetivo de realizar a síntese de um supervisor robusto quando houver perda de comunicação entre os módulos de uma estação; e na seção 5 são feitos os comentários finais e sugestões para prosseguimento deste trabalho.

2. Metodologia

Os métodos científicos são a base para a construção das teorias e eles consistem em um conjunto de procedimentos e regras aprovados pelo meio acadêmico e é de responsabilidade dos pesquisadores garantirem que a pesquisa científica tenha sido desenvolvida com seriedade e que ela seja passível de debate e verificação, para que a sociedade e o meio acadêmico a reconheça como algo sólido e de potencial relevância (Lacerda et al., 2013).

Lacerda et al., (2013) afirma que dispor de métodos de pesquisa diversos e bem definidos pode favorecer o avanço das teorias e, consequentemente, do conhecimento em determinadas áreas. Nesse sentido, a necessidade das organizações aprimorarem seus processos é reconhecida em Platts et al., (1998), pois os resultados obtidos por meio de métodos tradicionais nem sempre geram um contribuição adequada. Em Platts, (1993), Platts et al., (1998) e Da Costa et al., (2011) distinguem o conhecimento em engenharia e o conhecimento científico, sendo que a engenharia se atenta para a utilização do conhecimento

científico, tendo como objetivo projetar e construir artefatos para a solução dos problemas. Diante disso, Da Costa & De Lima, (2011) propõem a *Process Approach* – Abordagem por Processos, e em Lacerda et al., (2013), tentou-se colocar os termos dessa abordagem sob outra perspectiva, utilizando o conceito de *Design Science* (DS) – Ciência do Projeto, e o método que o constrói, apresentado em Simon, (1996), e procurou-se também apresentar como se conduziria a pesquisa em engenharia.

Para aplicar os conceitos de DS, primeiro se deve aprofundar na definição de Classes de Problemas e na tipologia geral de artefatos; após isso, se deve formalizar a operacionalização da DS na construção de pesquisas; e por último, se deve expor alguns cuidados para a validade da pesquisa em DS, com o intuito de tornar mais robusto a avaliação dos artefatos e, consequentemente, do conhecimento gerado.

2.1 Contexto da *Design Science*

O principal objetivo da DS é desenvolver conhecimento para a concepção e desenvolvimento de artefatos, reconhecendo, em um primeiro momento, que os problemas existentes nas organizações costumam ser específicos. *A Design Science* seria responsável por conceber e validar sistemas que ainda não existem, seja criando, recombinando, alterando produtos, processos, *softwares* e/ou métodos, para melhorar as situações existentes. A DS se preocupa com o conhecimento que pode ser utilizado para projetar as soluções (Van Aken, 2004)

Em Simon, (1996) é feita a diferenciação entre os ambientes natural e artificial, em que o ambiente natural se refere a um conjunto de conhecimentos sobre uma classe de objetos e/ou fenômenos do mundo. Já o ambiente artificial se ocupa da "concepção de artefatos que realizam objetivos", ou seja, dizem respeito a como as coisas devem ser para funcionar e atingir determinados objetivos. Ainda em Simon, (1996), é definido o ambiente artificial como aquilo produzido, inventado ou sofreu algum tipo de intervenção por parte do homem. Neste trabalho, SED são sistemas considerados feitos pelo homem.

(Romme et al., 2003) afirma que os estudos relacionados às organizações devem incluir a DS como um dos principais modos de conceber o conhecimento e de realizar pesquisas científicas e que o conhecimento gerado a partir desse método, contribui para o avanço no desenvolvimento da pesquisa com base em conhecimento do Tipo 2 (Mode 2), que é multidisciplinar, diferente do conhecimento Tipo 1, essencialmente acadêmica e aplicada à uma disciplina. As pesquisas orientadas ao Tipo 2 estão preocupadas em desenvolver problemas relevantes e complexos, que consideram o contexto em que seus resultados serão aplicados, e assim, o conhecimento desenvolvido não é descritivo-explicativo e sim prescritivo (Burgoyne & James, 2005).

No Quadro 1 pode-se observar a comparação entre a DS e algumas perspectivas filosóficas e epistemológicas.

2.2 Classes de Problemas e Artefatos

Simon (1996) afirma que as classes de problemas podem consistir em uma organização para o desenvolvimento e trajetória do conhecimento em uma DS, além de permitir que os artefatos e, consequentemente, suas soluções, não sejam apenas uma resposta pontual a certo problema em determinado contexto.

É importante ressaltar que cabe organizar uma lógica de definição de partida de classe de problemas, sendo que:

- Primeiro realiza-se o levantamento inicial do problema, prático ou teórico, identificando os objetivos/metas necessários para o problema;
- Em um segundo momentos, realiza-se uma revisão sistemática da literatura com o intuito de estabelecer o quadro de soluções empíricas conhecidas; e
- Faz-se a localização dos artefatos, ou seja, dos componentes do ambiente interno para atingir os objetivos em um determinado ambiente externo.

Apesar de não haver uma definição conceitual da classe de problemas, Lacerda et al. (2013) define classe de problemas como sendo a organização de um conjunto de problemas, práticos ou teóricos, que contenha artefatos avaliados, ou não, úteis para a ação nas organizações, sendo aberta a possibilidade de tratamento de problemas teóricos, as formas de testar um teoria e a formalização dos artefatos já existentes.

Dessa forma, é permitido a utilização dos métodos tradicionais de pesquisa, como pesquisa-ação, estudo de caso e modelagem por exemplo, para a formalização de artefatos existentes.

3. Referencial Conceitual para a Revisão da Literatura e Controle Supervisório com Perdas Intermitentes de Observação

Neste capítulo, é apresentada uma revisão do Controle Supervisório Sujeito a Perdas Intermitentes de Observação. As referências principais utilizadas são de Cassandras et al., (2008) e Alves et al. (2019).

3.1 Sistemas a Eventos Discretos

Uma maneira formal de se representar o comportamento lógico de um SED é por meio de linguagens. O ponto de partida é o fato de que qualquer SED possui um conjunto de eventos Σ associado a ele. O conjunto Σ é considerado como alfabeto de uma linguagem e sequências de eventos são consideradas como palavras daquela linguagem. Será denotado por Σ^* o conjunto de todas as palavras de comprimento finito formado por concatenação de elementos em Σ, incluindo a palavra vazia ε. A operação * é chamada de Fecho Kleene (Cassandras et al., 2008). O prefixo de uma palavra s no prefixo Σ é enumerável e infinito devido às infinitas sequências que o compõe. O fecho-prefixo de uma linguagem $L \in \Sigma^*$ é definido como sendo o fecho-prefixo de uma linguagem L, com $L \subseteq \Sigma^*$, e consiste em todos os prefixos das palavras em L, ou seja, $\overline{L} = s \in \Sigma^* : (\exists t \in L)\, s \leq t$. Observa-se que $L \subseteq \overline{L}$ e L é dita ser prefixo-fechada se $L = \overline{L}$.

Quadro 1 - Comparação entre *Data Science* e outras pesquisas.

Formas de Pesquisa Organizacional			
Categorias	Ciência Natural/Social	Humanidades	*Design Science*
Propósito	Entender fenômenos organizacionais, com base em uma objetividade consensual, desvendando padrões gerais e as forças que explicam estes fenômenos.	Descrever, entender, e refletir criticamente sobre a experiência humana de atores no âmbito de práticas organizadas	Produzir sistemas que ainda não existem – isto é, mudar sistemas organizacionais e situações já existentes para alcançar melhores resultados.
Modelo	Ciências naturais (física, por exemplo) e outras disciplinas que adotaram a abordagem científica (economia, por exemplo)	Humanas (como a estética, ética, hermenêutica, história, estudos culturais, literatura, filosofia).	Design e engenharia (por exemplo, arquitetura, engenharia aeronáutica, ciências da computação)
Visão do Conhecimento	Representacional: nosso conhecimento representa o mundo como ele é; a natureza do pensamento é descritiva e analítica. Mais especificamente, a ciência é caracterizada por: Uma busca por conhecimentos gerais e válidos. Ajustes nas formulações de hipóteses e testes.	Construtivista e narrativa: todo o conhecimento surge a partir do que os atores pensam e dizem a respeito do mundo; natureza do pensamento é crítica e reflexiva.	Pragmática: conhecimento a serviço da ação; a natureza do pensamento é normativa e sintética. Mais especificamente, o design assume que cada situação é única e se inspira em propostas e soluções ideais, pensamento sistêmico, e informações limitadas. Além disso, enfatiza a participação, o discurso como um meio de intervenção, e a experimentação pragmática.
Natureza dos Objetos	Fenômeno organizacional enquanto objetos empíricos, com propriedades descritivas e bem definidas, que pode ser efetivamente estudado de uma posição externa.	Discurso em que atores e pesquisadores se envolvem; apreciação da complexidade de um discurso particular tem precedência sobre a meta de alcançar o conhecimento geral.	Questões organizacionais e sistemas como objetos artificiais com propriedades mal definidas, tanto descritivas como imperativas, exigindo intervenções não rotineiras por parte de agentes com posições internas na organização. Propriedades imperativas também se desdobram de fins e de sistemas idealizados de maneira mais ampla.
Foco do Desenvolvimento da Teoria	Descoberta da relação causal geral entre variáveis (expressadas em afirmações hipotéticas): A hipótese é válida? As conclusões permanecem dentro dos limites de análise.	A questão-chave é se certa (categoria de) experiência(s) humana (s) em um ambiente organizacional é "boa", "justa" etc.	Será que um dado conjunto integrado de proposições de projeto funciona em uma certa situação (problema) mal definida? O projeto e desenvolvimento de novos artefatos tendem a se mover para fora das fronteiras da definição inicial da situação.

Fonte: Autores.

O comportamento de um SED também é representado por um autômato $G = (X, \Sigma, f, x_0, X_m)$, sendo X o conjunto e estados, Σ o conjunto finito de eventos, $f: X \times \Sigma \rightarrow X$ a função de transição, x_0 o estado inicial e X_m o conjunto de estados marcados. As linguagens geradas e marcadas por um autômato G são representadas respectivamente por $L(G)$, que representa todas as sequências de eventos que podem ocorrer a partir do estado inicial; e $L_m(G)$, que representa todas as sequências de eventos que terminam em um estado marcado, a representar uma tarefa completa.

O conjunto de eventos Σ pode ser particionado em $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$, em que $\dot{\cup}$ significa uma união disjunta, sendo que Σ_c representa o conjunto de eventos controláveis, eventos cuja ocorrência pode ser inibida por um agente de controle, dita desabilitada, e Σ_{uc} representa o conjunto de eventos não controláveis. Outra forma de particionar o conjunto de eventos é $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, sendo que Σ_o representa o conjunto de eventos observáveis e representa o conjunto de eventos não observáveis. Os eventos não observáveis são classificados dessa forma quando no sistema não há sensores que monitoram suas ocorrências.

Uma operação importante envolvendo linguagens é a projeção natural ou simplesmenΣ_{uo}te projeção. Em Ramadge & Wonham, (1989), a projeção é definida no domínio $P_o: \Sigma^* \rightarrow \Sigma_o^*$. A projeção toma uma palavra formada a partir do maior

conjunto de eventos Σ_l e apaga os eventos que não pertencem ao menor conjunto de eventos Σ_s. Em autômatos, a operação de projeção é representada pela construção do autômato observador, detalhada em Cassandras & Lafortune, (2008).

Seja um autômato determinístico $G = (X, \Sigma, f, x_0, X_m)$ em que existam eventos não observáveis $\Sigma_{uo} \subseteq \Sigma$ e observáveis $\Sigma_o \subseteq \Sigma$ com $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$.

Inicialmente, o alcance não observável de cada estado $x \in X$, denotado por $x \in X$, é definido como

$$UR(x) = \{y \in X: (\exists t \in \Sigma_{uo}^*)[(f(x,t) = y)]\}$$

Esta definição é estendida para o conjunto de estados $B \subseteq X$ por

$$UR(B) = \bigcup_{x \in B} UR(x)$$

Defina-se então o observador para G como $Obs(G) = (X_{obs}, \Sigma_o, f_{obs}, x_{0,obs}, X_{m,obs})$ e sua construção é feita seguindo os passos abaixo.

- **Passo 1:** Defina $x_{0,obs} \triangleq UR(x_0)$ e $X_{obs} = \{x_{0,obs}\}$
- **Passo 2:** Para cada $B \in X_{obs}$ e $\sigma \in \Sigma_o$, defina

$$f_{obs}(B, \sigma) = UR(\{x \in X: (\exists x_e \in B)[x \in f(x_e, \sigma)]\})$$

se $f(x_e, \sigma)$ é definido por algum $x_e \in B$, adicionar o estado $f_{obs}(B, \sigma)$ a X_{obs}. Se $f(x_e, \sigma)$ não está definido para nenhum $x_e \in B$, então $f_{obs}(B, \sigma)$ é não definido.

- **Passo 3:** Repita o **Passo 2** até que todos os estados de X_{obs} sejam analisados.
- **Passo 4:** $X_{m,obs} \triangleq \{B \in X_{obs}: B \cap X_m /= \emptyset\}$

Algumas considerações devem ser feitas após a construção do observador.

- $Obs(G)$ é um autômato determinístico;
- $L(Obs(G)) = P[L(G)]$; e
- $L_m(Obs(G)) = P[L_m(G)]$.

3.2 Teoria de Controle Supervisório Sob Perda Intermitente de Observação

Na abordagem de controle supervisório por perdas intermitentes de observação feita por Alves, (2014) e Alves et al., (2019), o conjunto de eventos Σ sofre uma nova partição, $\Sigma = \Sigma_{ilo} \dot{\cup} \Sigma_{nilo}$, em que Σ_{ilo} representa o conjuntos de eventos passíveis de perdas intermitentes de observação, enquanto que Σ_{nilo} representa o conjunto de eventos sempre observáveis. Diferentemente de Alves, (2014) e Alves et al., (2019), nesse artigo, será considerado que o conjunto $\Sigma_{uo} = \emptyset$. A razão é para simplificação da abordagem por isolamento dos eventos sujeitos a perdas intermitentes de observação dos eventos não observáveis ordinários. Consideramos a extensão para o caso geral trivial.

Ao considerar a perda intermitente de observação de um evento, define-se um novo conjunto de eventos não

observáveis $\Sigma'_{ilo} = \{\sigma' : \sigma \in \Sigma_{ilo}, sendo \Sigma_{ilo}$ um conjunto de eventos fictício, construído a partir de Σ_{ilo}, em que cada evento $\sigma' \in \Sigma'_{ilo}$ possui correspondência biunívoca com o evento $\sigma \in \Sigma_{ilo}$.

Em Carvalho, (2012) foi definida a operação de dilatação D, cujo mapeamento pode ser visto a seguir, definindo também um conjunto dilatado de eventos $\Sigma_{dil} = \Sigma \dot{\cup} \Sigma'_{ilo}$:

$$D: \Sigma^* \to 2^{\Sigma^*_{dil}}$$
$$s \mapsto D(s)$$

sendo que,

$$D(\varepsilon) = \{\varepsilon\}$$
$$D(\sigma) = \begin{cases} \{\sigma\}, & se\ \sigma \in \Sigma \setminus \Sigma_{ilo} \\ \{\sigma, \sigma'\}, & se \sigma \in \Sigma_{ilo} \end{cases}$$
$$D(s\sigma)\, x' = D(s)D(\sigma) \text{ se } s \in \Sigma^*, \sigma \in$$

A extensão do mapeamento D para uma linguagem $L \subseteq \Sigma^*$ é dada por $D(L) = \bigcup_{s \in L} D(s)$. Outra operação importante é a compressão, que recupera uma palavra s de qualquer palavra em $D(s)$ (Alves et al., 2019). O mapeamento da operação de compressão pode ser visto a seguir:

$$C: \Sigma^*_{dil} \to \Sigma^*$$
$$s_{dil} \mapsto C(s_{dil})$$

sendo que,

$$C(\varepsilon) = \Sigma$$
$$C(\sigma) = \sigma, se\ \sigma \in$$
$$C(\sigma') = \sigma, se\ \sigma' \in \Sigma'_{ilo}$$
$$C(s_{dil}\sigma) = C(s_{dil})C(\sigma), \forall s_{dil} \in \Sigma^*_{dil} e \sigma \in \Sigma_{dil}$$

Considerando a operação de projeção $P_{dil}: \Sigma_{dil} \to$, observe que há uma diferença entre a compressão e essa projeção. Enquanto P_{dil} suprime as ocorrências dos eventos em Σ'_{ilo} nas palavras, a compressão substitui as ocorrências desses eventos pelos correspondentes eventos em Σ_{ilo}, recuperando assim a palavra original da palavra dilatada. Para exemplificar, considere $s_{dil} = abc'bc'$, então $P_{dil}(s_{dil}) = abb$ e $C(s_{dil}) = abcbc$.

Na Teoria de Controle Supervisório considerando que $\Sigma = \Sigma_o$ ($\Sigma_{uo} = \emptyset$), tem-se que a interação entre supervisor (S) e planta (G) é feita de forma direta, ou seja, S controla G baseado na própria planta. Ao considerar que $\Sigma = \Sigma_{ilo} \dot{\cup} \Sigma_{nilo}$, a forma como o supervisor (denominado agora como supervisor robusto - S_R) interage com G é modificada pois, utilizando a abordagem de perda intermitente de observação, o supervisor agora atua sob a planta dilatada G_{dil} e não mais em G, seguida da projeção $\varepsilon \in L(S_R/G_{dil})$, em que $P_{dil}: \Sigma^*_{dil} \to \Sigma^*$ (Alves, 2014). Na abordagem de perda intermitente de observação, passamos a considerar então dois tipos de comportamento para a planta: o comportamento real e o

dilatado. A Figura 1 mostra a representação de S_R atuando sob G_{dil}.

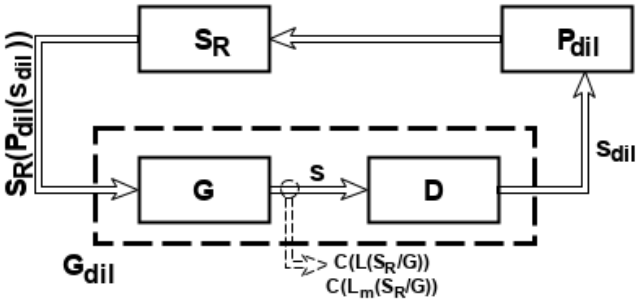
O supervisor S_R é definidor por

$$S_R{:}P_{dil}\left(D\left(L(G_{dil})\right)\right) \rightarrow 2^{\Sigma dil}$$

A linguagem gerada por S_R controlando G_{dil}, $L(S_R/G_{dil})$, é dada por:

1. $\varepsilon \in L(S_R/G_{dil})$; e
2. $$(\forall\, s_{dil} \in \Sigma_{dil}^{*})\,(\forall\, \sigma_{dil} \in \Sigma_{dil}), s_{dil}\sigma_{dil} \in L(S_R/G_{dil}) \Leftrightarrow \left(s_{dil}\sigma_{dil} \in L(G_{dil})\right) \wedge \left(s_{dil} \in L(S_R/G_{dil})\right) \wedge \left(\sigma_{dil} \in S_R\left(P_{dil}(s_{dil})\right)\right)$$

Figura 1: Realimentação em MF do Controle Supervisório para o Caso de Perdas Intermitentes de Observação.



Fonte: Autores (2020).

É possível observar na Figura 1 que o supervisor toma a decisão de desabilitar algum evento baseado na projeção da palavra dilatada $P_{dil}(s_{dil})$, pois é o momento que ele consegue enxergar a ocorrência de um evento.

A linguagem gerada por S_R controlando $G, L(S_R/G)$, é dada como $L(S_R/G) = C\left(L(S_R/G_{dil})\right)$, em que C é a operação de compressão. As linguagens marcadas são definidas por $L_m(S_R/G_{dil}) = L_m(G) \cap L(S_R/G_{dil})$ e $L_m(S_R/G) = L_m(G) \cap L(S_R/G)$.

A seguir apresenta-se o problema de controle nas formas do comportamento real (**Problema 1**) e dilatado (**Problema 2**).

Problema 1: Dados G com $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$, $\Sigma = \Sigma_{nilo} \dot{\cup} \Sigma_{ilo}$, com $\Sigma_{uo} = \emptyset$ e $K \subseteq L_m(G)$, encontrar S_R para G não bloqueante, tal que

$$\emptyset \neq L_m(S_R/G) \subseteq K.$$

Problema 2: Dados G_{dil} com $\Sigma_{dil} = \Sigma_{dil,c} \dot{\cup} \Sigma_{dil,uc}$, $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, sendo o conjunto de eventos observáveis é dado como sendo $\Sigma_o = \Sigma_{nilo} \dot{\cup} \Sigma_{ilo}$ e o conjunto de eventos não observáveis é dado como sendo $\Sigma_{uo} = \Sigma'_{ilo}$. Dado também $D(K) \subseteq L_m(G_{dil})$, encontrar S_R para G_{dil} não bloqueante, tal que

$$\emptyset \neq L_m(S_R/G_{dil}) \subseteq D(K).$$

Observa-se que **o Problema 2** é um problema de observação parcial ordinário (Cassandras e Lafortune, 2008).

A seguir, são apresentadas duas importantes definições: controlabilidade e observabilidade.

Definição 1 (Controlabilidade): Sejam $L, K \subseteq \Sigma^*$, com $L = \overline{L}$, e $\Sigma_{uc} \subseteq$. A linguagem $K \subseteq \Sigma^*$ é dita ser controlável e.r.a (em relação a) L e Σ_{uc} se

$$\overline{K}\Sigma_{uc} \cap L \subseteq \overline{K}$$

Definição 2 (Observabilidade): Sejam K e $L = \overline{L}$ linguagens sobre o conjunto de eventos $\Sigma = \Sigma_o \cup \Sigma_{uo}$ e P_o a projeção natural correspondente de Σ^* para Σ_o^*. A linguagem $K \subseteq \Sigma^*$ é dita ser observável e.r.a L , Σ_{uo} e P_o se, e somente, para todo $s, t \in \overline{K}$ e para todo $\sigma \in \Sigma$,

$$P_o(s) = P_o(t), s\sigma \in \overline{K}, t\sigma \in L(G) \Rightarrow t\sigma \in \overline{K}.$$

Outra definição importante é para uma linguagem ser dita $L_m - fechada$.

Definição 3 ($L_m(G) - fechada$): Dadas as linguagens (K e L_m) $\in \Sigma$, com $K \in L_m$, diz-se que K é fechada em relação a L_m ou $L_m - fechada$, quando:

$$K = \overline{K} \cap L_m.$$

Observa-se que os problemas de controle supervisório para o comportamento dilatado, correspondem aos problemas de observação parcial. O Teorema 1 é o resultado que dá solução para o **Problema 1**.

Teorema 1. Sejam G com $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$, $\Sigma = \Sigma_{nilo} \dot{\cup} \Sigma_{ilo}$, com $\Sigma = \Sigma_{nilo} \dot{\cup} \Sigma_{ilo}$, isto é, $\Sigma = \Sigma_o$ e $K \subseteq L_m(G)$, então existe um supervisor S_R para G não bloqueante, tal que $L_m(S_R/G) = K$ se, e somente se, existir $K_{dil} \subseteq D(K)$, tal que K_{dil} seja $L_m(G_{dil}) - fechada$, controlável (em relação a $L(G_{dil})$ e $\Sigma_{dil,uc}$), observável (em relação a $L(G_{dil}), \Sigma_{dil,o}$ e P_{dil}) e $C(K_{dil}) = K$.

O Teorema 1 é consequência direta dos resultados de Alves et al. (2014) e Alves et al. (2019), e decorre da relação entre os problemas de controle supervisório 1 e 2, e pelo fato de que o **Problema 2** é um problema ordinário de observação parcial.

Na busca de uma solução para o **Problema 1**, é necessário o cálculo de uma linguagem fechada relativa, controlável e observável. Uma das peculiaridades que precisa ser tratada é que a observabilidade de linguagens não é fechada para a união, não permitindo assim uma solução para o **Problema 1** por linguagem suprema (Cassandras & Lafortune, 2008).

É possível verificar a definição e o emprego de linguagens observáveis relativas em Cai et al. (2015), Cai et al. (2016), Alves et al. (2017) e Alves et al. (2019).

Nesse trabalho, entretanto buscaremos uma solução empregando o conceito de normalidade (Cassandras e Lafortune, 2008), conforme abordado em Alves et al. (2014).

Definição 3 (Normalidade): Considere $L = \overline{L} \subseteq \Sigma^*$ e $P: \Sigma^* \to \Sigma_o^*$. Uma linguagem $K \subseteq L$ é dita ser normal em relação a L e P se

$$\overline{K} = P^{-1}\left[P(\overline{K})\right] \cap L$$

Tem-se que a normalidade é fechada para união, permitindo encontrar um máxima sublinguagem normal ($SupN$) (Cassandras & Lafortune, 2008), e é tal que, implica em observabilidade. Se os eventos controláveis forem observáveis ($\Sigma_c \subseteq \Sigma_o$) e K for controlável e observável, então tem-se que que K é normal, ou seja, existe uma máxima sublinguagem

controlável e normal ($SupCN$), que também é uma máxima sublinguagem controlável observável ($SupCO$), sendo $SupCN = SupCO$ (Cassandras & Lafortune, 2008).

Isso será explorado neste trabalho, em que os eventos controláveis serão comandos dados à planta. Será considerado que os comandos possuem garantia de chegada da informação, enquanto as respostas podem ser perdidas.

Utilizando o conceito de normalidade, desenvolveu-se a solução para o **Problema 1**, já que a solução para **Problema 2** segue o problema de observação parcial.

Solução dos PCSs - Comportamento Real

Linguagem Gerada: O PCS possui solução se $C\left(SupCN\left(D(K)\right)\right)$ for prefixo-fechada e não vazia, e a solução é tal que $L(S_R/G) = C\left(SupCN\left(D(K)\right)\right)$.

Linguagem Marcada: O PCS possui solução se $C\left(SupCN\left(D(K)\right)\right)$ for $L_m - fechada$ e não vazia, e a solução é tal que, $L_m(S_R/G) = C\left(SupCN\left(D(K)\right)\right)$.

Assim, chega-se um passo a passo para a síntese de supervisores, que será utilizada nesse trabalho:

1. Modelar a planta G identificando $\Sigma_c, \Sigma_{uc}, \Sigma_{nilo}$ e Σ_{ilo}, com $\Sigma_{uo} = \emptyset$;
2. Modelar as especificações como linguagens prefixo-fechadas, $E = \overline{E}$;
3. Calcular a linguagem alvo $K = E \cap L_m(G)$, reconhecida por um autômato H;
4. Obter G_{dil}, tal que $L_m(G_{dil}) = D\left(L_m(G)\right)$ e H_{dil}, tal que, $L_m(H_{dil}) = D(K)$;
5. Calcular *SupCN(D(K));*
6. Dada $SupCN\left(D(K)\right)$ não vazia, o real comportamento observado é $C\left(SupCN\left(D(K)\right)\right)$. Assim, basta que $C\left(SupCN\left(D(K)\right)\right) \subseteq K$; e
7. O supervisor que atende à especificação é o reconhecedor de $P_{dil}\left(SupCN\left(D(K)\right)\right)$.

4. Planta MECATRIME: Modelagem Visual

Nesta seção é apresentada a planta MECATRIME por diversas perspectivas que serão utilizadas para realizar a sua modelagem. As perspectivas adotadas são: perspectiva de componentes, perspectiva de fluxo, perspectiva de comunicação lógica e perspectiva de comunicação de sinais. Em seguida será apresentada a modelagem por autômatos para o comportamento de uma estação da planta.

4.1 Planta MECATRIME: Visão Geral

A planta MECATRIME é um sistema de Manufatura Integrado por Computador (*Computer Integrated Manufacturing - CIM*) com seis estações de produção que realizam os principais processos encontrados na indústria, um armazém automatizado (ASRS - Automated Storage and Retrieval *System*) para peças acabadas, uma esteira transportadora, que interliga fisicamente todas as estações e uma estação de gerenciamento. As estações de produção são robotizadas e possuem um computador próprio, sendo possível realizar a fabricação de produtos manufaturados, sendo classificadas então como Sistemas

Flexíveis de Manufatura (*Flexible Manufacturing System - FMS*). Na Figura 2(a) pode-se observar a planta baixa com as estações da planta MECATRIME e na Figura 2(b) é possível observar uma visão geral da planta MECATRIME, em que é possível ver algumas estações que a compõem, bem como alguns componentes presentes em cada uma delas.

Na Figura 2(b), em um primeiro plano, temos a estação de metrologia da planta MECATRIME, que é representada pela estação E5 da Figura 2(a).

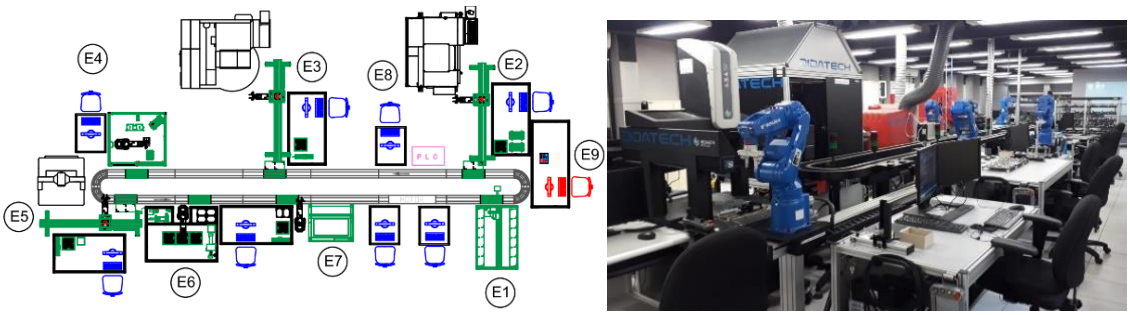
A seguir apresenta-se uma breve descrição de cada estação:

1. $E1$ – ASRS: Estação de armazenamento, composta por uma estante com 72 lugares;
2. $E2$ – Torneamento: Estação onde as peças são usinadas por um centro de torneamento *CNC Emco Concept Turn* 250;
3. $E3$ – Centro de usinagem: Estação onde as peças são usinadas por um centro de usinagem *CNC Emco Concept Mill* 250;
4. $E4$ – Soldagem: Estação onde se realiza solda MIG (*Metal Inert Gas*) robotizada de peças metálicas;
5. $E5$ – Metrologia: Estação onde é realizada a inspeção das peças por meio de uma máquina de medida de coordenadas (CMM - *Coordinate Measuring Machine*);
6. $E6$ – Montagem e inspeção visual: Estação onde ocorre a montagem de peças compostas, além de verificar a qualidade da montagem por meio de um sistema de visão computacional;
7. $E7$ – Gravação a laser: Estação onde se realiza uma estampagem nas peças utilizando uma gravadora a laser;
8. $E8$ – Gravação a laser: Estação onde se realiza uma estampagem nas peças utilizando uma gravadora a laser;
9. $E9$ – Gerenciamento ou *MANAGER*: Estação onde se realiza o controle da produção, gerando ordens de serviços (OS). Também possui um Gêmeo Digital da Planta MECATRIME para fins de simulação de produção.

A estação de armazenamento possui um robô cartesiano que acomoda as peças tanto na estante, quanto na esteira. Já as estações de produção possuem um robô manipulador de 6 graus de liberdade, *Motoman* $MH5F$ da *YASKAWA*. Essencialmente o robô manipulador transporta as peças no interior das estações. Nas estações de soldagem e montagem o robô é quem realiza as operações sobre as peças. Nas estações de torneamento, centro de usinagem e metrologia, há um trilho horizontal (LSB - *Linear Slidebase Belt-Drive*) que aumenta o espaço de trabalho do robô manipulador, permitindo assim o acesso de locais da estação que são fisicamente distantes.

Figura 2: Planta MECATRIME.

(a) Planta Baixa (b) Visão Geral



Fonte: Autores (2020).

As peças, sejam elas acabadas ou não, são acomodadas e transportadas sobre os *templates* (suportes). Ao entrar ou sair do ASRS, um *template* é identificado, acomodado sobre um pallet e então transportado pela esteira até uma estação de trabalho. Os pallets movimentam-se pela esteira até que, quando se fizer necessário, os *templates* possam ser pegos pelos robôs presentes em cada estação. Nas paradas de estação, que é o local que identifica o pallet que está passando por aquela estação e que permite o avanço ou não do pallet para a próxima estação, utilizando um pistão presente em um circuito pneumático para forçar a parada do pallet.

Em cada uma das estações de trabalho e na estação de armazenamento há uma parada de estação que realiza a identificação dos pallets, presentes na esteira, por meio de um leitor magnético. A função dos pallets é acomodar os *templates* com a peças e levá-los até a estação de destino. Da mesma forma que os pallets possuem identificação própria, os *templates* possuem um identificador RFID (*Radio-Frequency Identification*), com o intuito de identificar o *template*, e, indiretamente, qual peça está sobre ele. Vale ressaltar que, por meio de mensagens que são enviadas pelas estações durante todo o processo, o MANAGER consegue saber se *template* está vazio ou não, se está com uma peça bruta ou acabada. A leitura da identificação RFID dos *templates* é feita toda vez que eles entram ou saem da estação de armazenamento.

As peças brutas são armazenadas nos alimentadores gravitacionais e as peças acabadas são acomodadas no racks. O produto padrão para uma produção local realizada na estação de torneamento, também é chamado *Sup_Brass*, mas diferentemente da produção global, a peça é feita de resina. A Figura 3 mostra a visão geral da estação de torneamento.

A Figura 3 mostra a estação de torneamento da planta MECATRIME, bem como sua composição: braço robótico, computador e o centro de torneamento. Como já foi dito, as estações de trabalho são consideradas FMS e elas possuem um estoque próprio de peças, permitindo assim realizar produções locais sem a necessidade do uso da esteira.

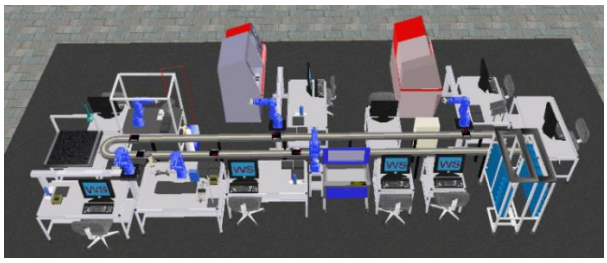
A supervisão e os comandos de produção global são feitos pelo *MANAGER* por meio do software *OpenCIM*, que também é utilizado para concepção do sistema em si, no sentido de que existe um *Digital Twin* da planta MECATRIME implementado no *OpenCIM*. A Figura 4 mostra a tela onde é possível acompanhar todo o processo de produção do MECATRIME.

É possível ter por meio da Figura 4, uma visão 3D de toda a planta, em que o *software* permite simular toda a produção antes de comandar a produção propriamente dita.

Figura 3: Estação de Torneamento.

Fonte: Autores (2020).

Figura 4: *Digital Twin* do MECATRIME.



Fonte: Autores (2020).

4.2 Modelagem Visual

Nesta seção será mostrada a importância de se realizar a modelagem de um sistema bem estruturada e padronizada.

Modelar é o primeiro passo para se extrair as informações de um sistema, pois é nessa fase que é feita uma documentação com o intuito de conseguir todas as características do sistema, como detalhes de funcionamento, estrutura física e de comunicação e outras mais informações que forem necessárias (Trivelato, 2003; Gonçalves et al., 2015). A representação da modelagem de sistemas geralmente é por meio de notação gráfica, sendo mais comum o uso de notações em UML (Sommerville, 2011). O que deve ficar claro, é que a modelagem baseada em UML não é um linguagem de programação, não é focada na codificação do software ou do hardware, ela é uma linguagem de modelagem, focada na análise do problema e na sua solução, auxiliando no entendimento geral do sistema (Pinheiro, 2016; Guedes, 2011).

Por não estar ligada à um processo exclusivo, pensou-se em utilizar os conceitos da modelagem em UML para obter todas as informações possíveis da planta MECATRIME no desenvolvimento deste trabalho.

A UML representa o sistema a partir de perspectivas diferentes, sendo essas perspectivas chamadas de visões. Em (Kruchten, 1995) são recomendadas quatro visões fundamentais de arquitetura, são elas:

- Visão Lógica: mostra as abstrações fundamentais do sistema como objetos ou classes de objetos;
- Visão de Processo: mostra como o sistema é composto de processos interativos;
- Visão de Desenvolvimento: mostra como o software é decomposto para o desenvolvimento; e
- Visão Física: mostra o hardware do sistema e como os componentes do software são distribuídos entre os processadores.

O intuito de se utilizar essas visões é reduzir a complexidade do sistema como um todo, pois cada uma delas

especifica um aspecto diferente. Para a realização deste trabalho, a modelagem feita foi pensando nas visões de processo, de desenvolvimento e física.

A UML possui alguns diagramas divididos em duas categorias: estruturais e comportamentais (Sommerville, 2011; Pimentel, 2015).

Na descrição da planta MECATRIME neste trabalho foram utilizados três diagramas estruturais e dois diagramas comportamentais. Dos diagramas estruturais empregaram-se os diagramas de componentes, que mostra o sistema por um lado funcional, de estrutura composta, que detalha a estrutura de componentes do sistema, e de implantação, que consiste na organização do conjunto de elementos para a sua execução. Dos diagramas comportamentais, empregaram-se os diagramas de comunicação, que mostra como os componentes trabalham em conjunto para executar uma função, e de máquinas de estado, para representar os comportamentos dinâmicos dos sistemas.

Os autômatos serão empregados como formas de diagramas de máquinas de estados neste trabalho.

4.3 Diagrama de Estrutura Composta

Ao analisar a planta MECATRIME a partir de uma visão física, pôde-se construir um diagrama que representa a hierarquia em que se encontra a decomposição física da planta. O desenvolvimento dessa hierarquia foi baseado no trabalho de (Brandenbourger e Durand, 2018), que divide os níveis hierárquicos da seguinte maneira:

- Linha: Uma linha de produção é um sistema de produção que contém várias unidades de fabricação de valor agregado;
- Célula: É um sistema de processamento que executa processamento completo ou parcial na peça de trabalho;
- Estação: É uma unidade autossuficiente, projetada para uma etapa específica do processo, espacialmente separada de outras estações e interligada à outra estação por meio de um sistema de transporte dentro da célula;
- Subestação: É uma estrutura parcial de uma estação usada com o intuito de decompor logicamente e fisicamente as estações complexas em unidades gerenciáveis;
- Módulo: É um agrupamento lógico de componentes que podem existir dentro de uma estação ou subestação;
- Componente: É uma unidade de automação que executa uma tarefa simples e específica de automação; e
- Elemento: É uma unidade de automação básica que necessita ser conectada à uma unidade de controle e que não podem ser divididos em unidades menores.

Pensando exclusivamente na construção de um diagrama hierárquico para a estação de torneamento e mantendo a abordagem feita para o sistema completo, chegou-se à uma hierarquia vista na Figura 5. De maneira geral, pode-se observar na Figura 5 as entradas e saídas que cada componente possui, a interação existente entre cada uma das entidades e as partes que compõem a estação. Mais especificamente, o nível Módulo é composto por:

1. *Buffer*, cuja função é acomodar temporariamente os *templates* enquanto a peça é trabalhada;
2. *Alimentador*, que possui a função de fornecer matéria-prima para a produção de um específico produto;
3. *Rack* para armazenar as peças acabadas;
4. *Manipulador* para posicionar e orientar a peça de maneira correta durante o processo de produção; e
5. *Torno*, cuja função é usinar peças de acordo com a especificação desejada.

No nível *Componentes*, é possível ver a divisão de cada entidade do nível anterior, podendo então verificar que a

entidade *Buffer* divide-se em 2 *buffers*, o *Alimentador* divide-se em 2 alimentadores, a entidade *Rack* é dividida em 4 *racks*, o *Manipulador* de maneira geral é composto pela garra, pelo braço e pelo LSB, e a entidade Torno é composta pela porta, permitindo o acesso do robô ou operador para colocar ou tirar a peça, pela morsa, que segura a peça, e pelo CNC, que possui o revólver de ferramentas.

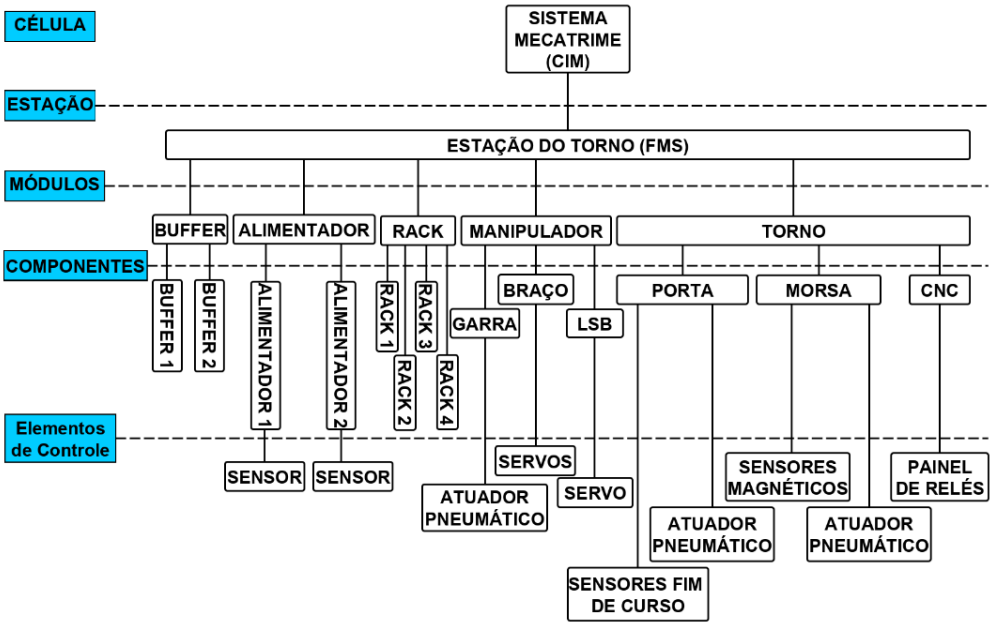
No nível mais abaixo, estão os *Elementos de Controle*, que representam sensores (presença, fim de curso e magnético), atuadores (servos e pneumático) e painel de relés, que são associados a cada equipamento.

4.4 Diagrama de Implantação

Foi elaborado então o diagrama que representa a visão de processo da estação de torneamento da planta MECATRIME e chegou-se a um modelo da Figura 6.

Na Figura 6 é possível observar que o robô é o elemento principal do fluxo de peças dentro da estação, fazendo com que as peças alimentem cada um dos módulos e assim, seja possível realizar uma produção nesta estação.

Figura 5: Diagrama de Estrutura Composta da Estação de Torneamento.

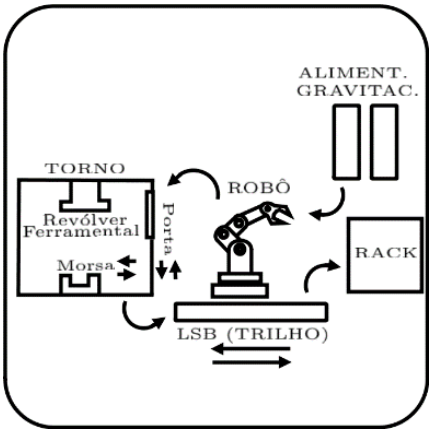


Fonte: Autores (2020).

4.5 Diagrama de Comunicação

Com a construção do diagrama de estrutura composta da estação de torneamento, pensou-se então em outro diagrama que representasse uma visão física, mas com o intuito de identificar os meios de comunicação existentes entre as partes que integram a planta. O diagrama de comunicação, conforme a Figura 7, foi desenvolvido apenas para a estação de torneamento. Ao construir um diagrama utilizando essa perspectiva, é possível identificar em um primeiro momento, onde e como podem ocorrer as perdas de comunicação e é a partir desse diagrama que se pode visualizar um encapsulamento presente em algumas partes. Tanto a perda de comunicação, quanto a expansão das partes encapsuladas, serão tratadas melhor a seguir, nos próximos diagramas.

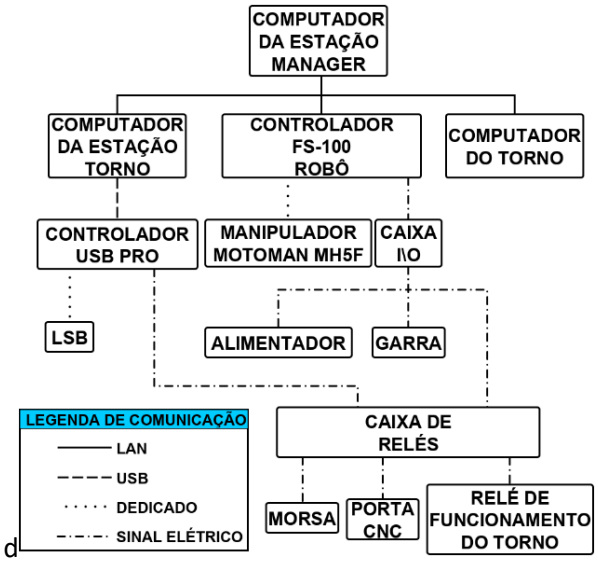
Figura 6: Diagrama de Implantação da Estação de Torneamento.



Fonte: Autores (2020).

Observa-se que a comunicação entre a estação *MANAGER* com a estação de torneamento é feita via *LAN* (rede local *Ethernet*), sendo que o computador da estação de torneamento, o controlador *FS100*, e o computador do torno estão pendurados nessa rede de comunicação. O computador da estação do torno se comunica via USB com um controlador responsável por alguns controles lógicos, sendo que esse controlador possui dois tipos de comunicação: uma dedicada com o trilho LSB, para realizar sua movimentação, e outra por meio de sinal elétrico, com uma caixa de relés, com o intuito de realizar a abertura ou fechamento da morsa, o fechamento da porta, além da própria partida e informação do fim da operação do torno. Já o controlador do robô realiza dois tipos de comunicação: uma dedicada com o manipulador e outra via sinal elétrico com uma caixa de entradas e saídas, que por sua vez é ligada ao alimentador, à garra do robô e à caixa de relés, com o intuito de realizar a abertura da porta do torno.

Figura 7: Diagrama de Comunicação da Estação de Torneamento.



Fonte: Autores (2020).

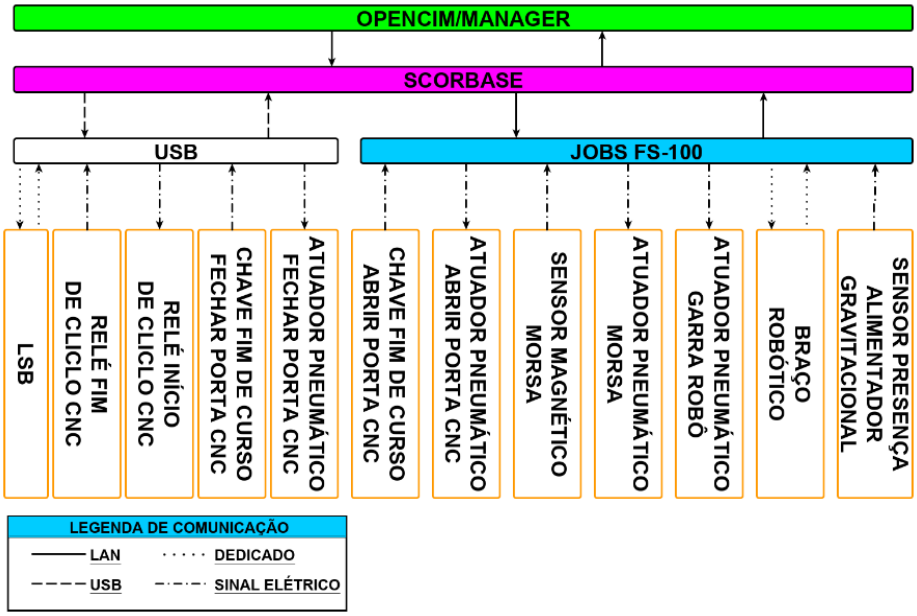
Os programas de usinagem são carregados diretamente da estação do *MANAGER* no computador do torno CNC. O *MANAGER* tem uma nuvem em que todos os programas responsáveis pelo funcionamento da planta MECATRIME estão presentes nela. Especificamente para a estação de torneamento, o programa do *SCORBASE* (será dado mais detalhes na Seção 3.6) e o programa de usinagem.

4.6 Diagrama de Componentes

Baseado na visão de desenvolvimento, pensou-se em um diagrama que mostrasse a relação entre os componentes de software e os dispositivos físicos por meio de sinais, e chegou-se a um primeiro modelo, visto na Figura 8, que mostra o tipo de comunicação que há entre hardware e software. Observa-se que esse diagrama segue a premissa do tipo de comunicação existente no diagrama da Figura 7. É possível verificar a partir da Figura 8, a interação e o fluxo de comunicação existente entre o mundo físico e o mundo cibernético. O mundo físico corresponde ao controlador USB e aos diversos dispositivos pendurados no sistema pelas diversas interfaces de rede. Já o mundo cibernético é composto pelo *OpenCIM/MANAGER*, que é o software que roda no gerente, pelo *SCORBASE*, que é o software de controle da estação e os programas (Jobs ou rotinas) do controlador *FS100*.

Analisando o mundo cibernético, é possível fazer algumas considerações. Em torno do *MANAGER* estão todas as estações da planta, as de trabalho, armazenagem e a esteira transportadora. Vale a pena ressaltar que o diagrama de comunicação de sinais foi construído considerando apenas a estação de torneamento e não a planta MECATRIME como um todo. Em cada uma das estações há o software *SCORBASE*, que é responsável por coordenar o funcionamento da produção na estação, por meio do controlador *USB CONTROLLER*, além de ser responsável pela comunicação com o *MANAGER*. Mais abaixo, existem as rotinas do controlador *FS100*.

Figura 8: Diagrama de Componentes Relacionando o Acionamento dos Dispositivos Físicos.



Fonte: Autores (2020).

Pode-se notar também, na Figura 8, que o controlador USB foi separado do *SCORBASE*. Isso foi feito de maneira proposital porque foi percebido que entre o *SCORBASE* e os dispositivos pendurados no controlador, existem dois caminhos de rede que queríamos chamar a atenção: primeiro caminho é entre o *SCORBASE* e o controlador *USB*, que está ligado via *USB*, podendo então haver uma falha física; o segundo caminho é entre o controlador USB e os dispositivos, que estão ligados por meio de fios ou por uma comunicação dedicada. Por essa razão optou-se por separar o controlador *USB* do *SCORBASE*.

Uma consideração a ser feita é o fato do *MANAGER* se comunicar apenas com *SCORBASE*, apesar do *FS100* também estar conectado à mesma rede.

Realizou-se uma expansão da Figura 8 com o intuito de construir um outro diagrama de componentes, mas desta vez

representando as rotinas executadas tanto pelo *SCORBASE*, quanto pelo *FS100* para realização das tarefas de produção local e as *TAGs*, indicando o sinal físico enviado ou recebido pelos controladores. Esse novo diagrama pode ser visto na Figura 9. É possível observar nessa figura a indicação das entradas e saídas que são utilizadas para execução das tarefas comandadas pelo *SCORBASE* e pelo *FS100*. A indicação é feita por meio de uma caixa de texto tracejada.

Também na Figura 9 foram indicados os módulos para os quais serão obtidos os modelos de máquinas de estados, na forma de autômato, na seção 3. Os eventos criados, indicados pela caixa de texto destacada de amarelo logo abaixo dos componentes físicos, tiveram relação com as TAGs, que representam eventos de comandos e respostas. Para os modelos dos autômatos, teve-se uma premissa parecida, pois foram obtidos a partir das linhas de códigos do SCORBASE e do FS100 . As rotinas que são executadas tanto pelo SCORBASE quanto pelo FS100, podem ser vistas nas respectivas caixas de texto desses controladores.

Nas duas figuras que representam a comunicação, Figura 10 e Figura 11, pode-se observar o cuidado que se teve em relação a padronização de cores das partes integrantes dos diagramas, sendo que a cor verde indica o MANAGER, a cor lilás indica o SCORBASE, a cor azul indica o FS100 e a cor laranja indica os componentes físicos.

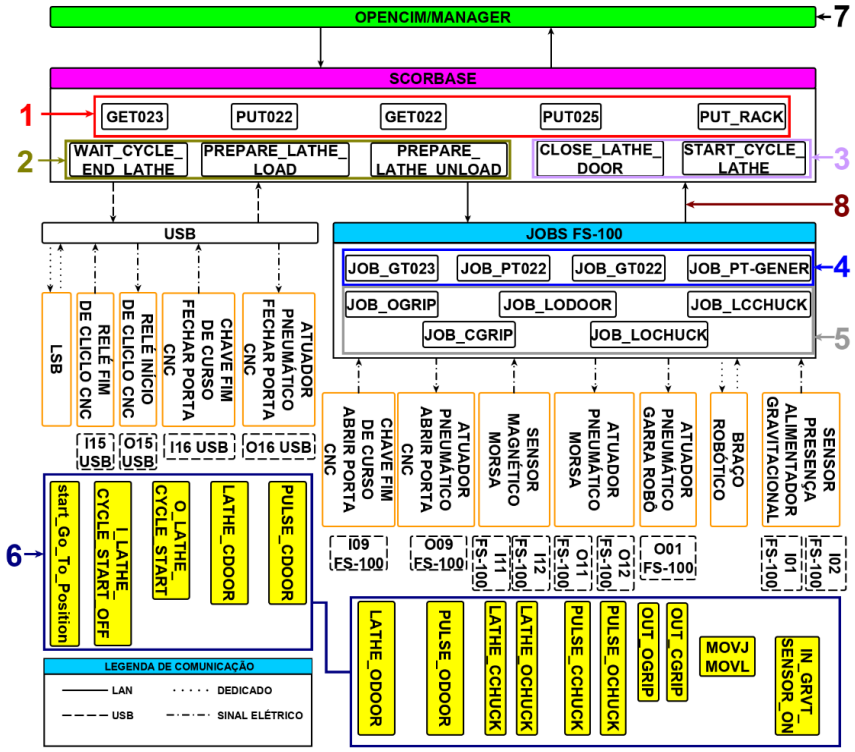
E por último, a Figura 9 também nos mostra a sequência que se seguiu para construir os modelos dos autômatos, sendo:

1. Modelos baseados nas rotinas principais do SCORBASE ;
2. Modelos baseados nos acionamentos de dispositivos pelo SCORBASE;
3. Modelos baseados em variáveis de memória do SCORBASE;
4. Modelos baseados nas rotinas principais do FS100;
5. Modelos baseados nos acionamentos de dispositivos pelo FS100;
6. Modelos baseados no funcionamento dos dispositivos físicos;
7. Modelos baseados na comunicação realizada pelo SCORBASE e MANAGER; e
8. Escolha dos eventos passíveis de perda de observação por perda de comunicação.

4.7 Diagrama de Máquinas de Estados (Autômatos)

Nesta seção será mostrada a modelagem que foi feita baseada em autômatos para as rotinas do SCORBASE e do FS100, para o funcionamento dos dispositivos físicos do sistema e da comunicação entre MANAGER e SCORBASE.

Figura 9: Diagrama de Componentes.



Fonte: Autores (2020).

4.7.1 Rotinas de PRODUÇÃO LOCAL

Para realizar a produção local da peça *Sup_Brass*, o *MANAGER* e o *SCORBASE* trocam mensagens para que a execução das tarefas seja feita de maneira segura e completa. Há algumas tarefas que são executadas pelo *FS100* e por isso, também há troca de mensagens entre *SCORBASE* e *FS100*. A comunicação entre *MANAGER* e *SCORBASE* é da forma mestre e escrevo e é feita da seguinte maneira: o *MANAGER* envia para o SCORBASE as tarefas que devem ser realizadas e o *SCORBASE* envia mensagens ao *MANAGER* quando uma tarefa é concluída. Para o *SCORBASE* e *FS100*, a comunicação é feita de maneira análoga, em que o *SCORBASE* envia o comando para o início de uma tarefa e o *FS100* responde quando ela é finalizada. Tanto o *SCORBASE* quanto o *FS100* possuem cinco rotinas principais, como visto nos blocos internos da Figura 9, e essas rotinas chamam outras sub-rotinas. As principais do *SCORBASE* e suas atribuições são:

1. *GET023* - Rotina responsável por pegar a peça no alimentador gravitacional, por movimentar o robô pelo trilho LSB, além de chamar a rotina *JOB_GT023* do *FS100* para comandar os movimentos do robô;
2. *PUT022* - Rotina responsável por colocar a peça no torno, por chamar a rotina *JOB_ PUT022* do *FS100* para comandar os movimentos do robô, além de fazer a movimentação de volta do robô pelo trilho LSB;
3. *START_CYCLE_LATHE* - Rotina responsável por iniciar o funcionamento do torno e por chamar a rotina *WAIT_CYCLE_END_LATHE* do próprio *SCORBASE*, para aguardar o encerramento da usinagem;
4. *GET022* - Rotina responsável por pegar a peça no torno, por movimentar o robô pelo trilho LSB, além de chamar a rotina *JOB_GT022* do *FS100* para comandar os movimentos do robô; e
5. PUT025 - Rotina responsável por colocar a peça no rack, por chamar a *rotina PUT_RACK* do próprio *SCORBASE*, que chama a rotina *JOB_PT-GENER* do *FS100* para comandar os movimentos do robô.

4.7.2 Rotinas do *SCORBASE*

As rotinas executadas pelo *SCORBASE* foram usadas para modelar um autômato, em que a premissa foi identificar cada linha de comando do programa e associar a execução desse comando como um evento. A Figura 10 ilustra a rotina *GET023* do *SCORBASE*, em que é possível observar as linhas de código que são executadas para que o robô se movimente e realize uma ação. Os procedimentos de abstração empregados nessa rotina são generalizáveis para as outras rotinas, sendo o *GET023* apresentado como ilustração. exceto os comandos das linhas 9 e 10.

Ao observar o funcionamento das rotinas em execução, pôde-se notar que um comando é enviado e que uma resposta é recebida ao fim de cada linha e, por isso, dividiu-se em eventos de início e fim de tarefa. Como exemplo, a linha 3 do código é dividida nos eventos *sCP1* e eCP*1*, sendo os eventos de start definidos como controláveis e os eventos de end definidos como não controláveis. A linha de comando Wait não será associada a um evento pois se trata de um atraso na programação.

Foram identificados também os eventos que representam comunicação entre os módulos e os que são eventos internos de cada um (linhas 2 e 12 por exemplo). Para exemplificar, os eventos citados representam a comunicação entre *SCORBASE* e *FS100* via rede Ethernet, sendo *sCP1* enviado do *SCORBASE* para o *FS100* e *eCP1* enviado do *FS100* para o *SCORBASE*.

Outro caso particular está ilustrado nas linhas 8, 9 e 10, da rotina *GET023*. Esse conjunto de rotinas trata da comunicação entre uma rotina do *SCORBASE* com um *Job* do *FS100*. Em particular, ilustra-se a comunicação com o *Job_GT023*. Na linha 8 trata-se do comando de início do *Job_GT023*, na linha 9, aguarda-se por um tempo de 10 décimos de segundo por conta da comunicação em rede e, por fim, na linha 10, aguarda-se a resposta do *Job* por até 60 segundos. Essas três linhas serão modeladas inicialmente por dois eventos, um de início e outro de fim de execução do *Job*. mais à frente, na Seção 4.3, vamos modelar a perda de comunicação na rede, para isso vamos inserir um evento de *time_out.*

Figura 10: Rotina GET023.

```
1 Set Subroutine GET023
2 Call Subroutine SCRIPT.GET_FROM_GFRD1
3 Copy FS100 Position SCRIPT.P1 to Position 1001
4 Copy FS100 Position SCRIPT.P2 to Position 1002
5 Copy FS100 Position SCRIPT.P3 to Position 1003
6 Copy FS100 Position SCRIPT.P4 to Position 1004
7 Go to Position SCRIPT.PB1 Speed 50 (%)
8 FS100 Start Job GT023
9 Wait FS100_DELAY_TIME (10ths of seconds)
10 FS100 Job Wait 60 (seconds)
11 Send Message $Start to MANAGER ID = TASK_ID
12 Return from Subroutine
```

Fonte: Autores (2020).

Com esse princípio, gerou-se o autômato da Figura 11(a), que representa a execução da rotina *GET023*. Na Figura 11(a), as cores observadas estão ligadas às cores da Figura 9 e evidenciam o sentido de comunicação, que é feito da esquerda para a direita. Note que do estado 1 para o 2 ocorre o evento *sG*, que é um comando enviado pelo *MANAGER* ao *SCORBASE*. Há também, eventos que não evidenciam comunicação, como por exemplo o evento *start_Go_To_Position_PB1_GET023*, que é um evento executado internamente pelo *SCORBASE*.

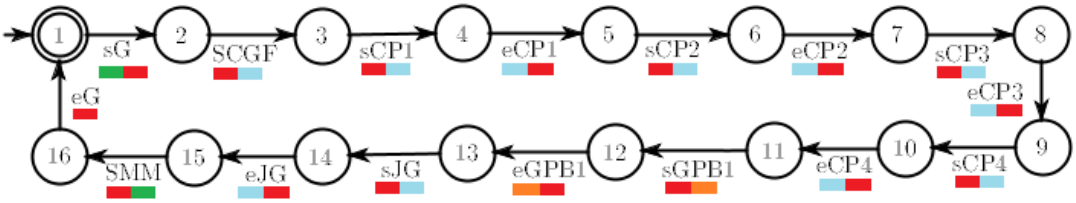
No intuito de facilitar uma melhor visualização dos eventos representados nas figuras, a Tabela 1 apresenta as siglas que foram utilizadas na simplificação desses eventos.

Para fim de tratabilidade e abstração do modelo, foram identificadas algumas condições que poderiam ser simplificadas, como por exemplo na Figura 11(a), onde há quatro eventos de copiar posição. Por se tratar de uma mesma operação, esses eventos foram simplificados em apenas um, como pode ser visto na Figura 11(b). Outro exemplo de

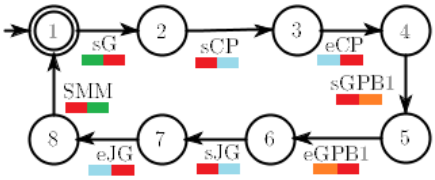
simplificação refere-se ao evento que representa o fim de uma rotina, que para a rotina GET023, trata-se do evento eG. Para iniciar uma tarefa, o MANAGER envia um comando ao SCORBASE para iniciar a execução de uma rotina. No caso do fim dessa rotina, o evento eG não indica uma comunicação entre SCORBASE e MANAGER, a comunicação é identificada no passo anterior, quando o SCORBASE envia uma mensagem de fim tarefa ao MANAGER. Por essa razão, o evento que indica comunicação entre SCORBASE e MANAGER passa a ser o evento de fim de rotina, como visto no modelo simplificado da Figura 11(b).

Figura 11: Autômato Completo Referente a Rotina *GET023* (*SCORBASE*).

(a) Completo



(b) Simplificado



Fonte: Autores (2020).

Tabela 1: Tabela de Simplificação dos Eventos.

Evento Original	Sigla	Evento Original	Sigla
start_GET023	sG	LATHE_CDOOR1	LCD1
script.GET_FROM_GFDR1	SCGF	end_CLOSE_LATHE_DOOR	eCD
start_copy_FS100_Position_Script.P1_GET023	sCP1	Start_Prepare_LATHE_LOAD	sPLL
end_copy_FS100_Position_Script.P1_GET023	eCP1	end_LATHE_LOAD	SLL
start_copy_FS100_Position_Script.P2_GET023	sCP2	Start_JOB_LODOOR_PT022	sJODP2
end_copy_FS100_Position_Script.P2_GET023	eCP2	Start_JOB_LODOOR_GT022	sJODG2
start_copy_FS100_Position_Script.P3_GET023	sCP3	LATHE_CDOOR2	LCD2
end_copy_FS100_Position_Script.P3_GET023	eCP3	PULSE_ODOOR	POD
start_copy_FS100_Position_Script.P4_GET023	sCP4	LATHE_ODOOR2	LOD2
end_copy_FS100_Position_Script.P4_GET023	eCP4	end_JOB_LODOOR_PT022	eJODP2
start_Go_To_Position.PB1_GET023	sGPB1	end_JOB_LODOOR_GT022	eJODG2
end_Go_To_Position.PB1_GET023	eGPB1	start_PUT022	sP
start_JOB_GT023	sJG	End_PUT022	eP
start_MOVJ_P1004_GT023	sMJ4	send_FINISH_FROM_PUT022_to_MANAGER	SFM
end_MOVJ_P1004_GT023	eMJ4	send_END_FROM_PUT022_to_MANAGER	SEM
start_MOVJ_P1003_GT023	sMJ3	START_CYCLE_LATHE	sCC
Start_JOB_OGRIP	sJOG	send_ENDTURN_to_CNC	SEC
OUT_OGRIP	OOG	start_GET022	sG2
end_JOB_OGRIP	eJOG	end_GET022	eG2
IN_GVRT_SENSOR_ON	iGVS	send_START_FROM_GET022_to_MANAGER	SMM2
start_MOVL_P1001_GT023	sML1	Start_PUT025	sP2
end_MOVL_P1001_GT023	eML1	end_PUT025	eP2
Start_JOB_CGRIP	sJCG	send_FINISH_FROM_ PUT_RACK _to_MANAGER	SFM5
OUT_CGRIP	OCG	send_END_FROM_PUT_RACK_to_MANAGER	SEM5
end_JOB_CGRIP	eJCG	start_copy_FS100_Position_Script_GET023	sCP
start_MOVL_P1002_GT023	sML2	end_copy_FS100_Position_Script_GET023	eCP
end_MOVL_P1002_GT023	eML2	start_MOV_GT023	sM
start_MOVJ_P0999_GT023	sMJ9	end_MOV_GT023	eM
end_MOVJ_P0999_GT023	eMJ9	*PUT_RACK_NUMBER*	PRN
end_JOB_GT023	eJG	PULSE_OCHUCK	POC
send_START_FROM_GET023_to_MANAGER	SMM	PULSE_CCHUCK	PCC
end_GET023	eG	LATHE_OCHUCK	LOC
SET_LATHE_LOAD	SLL	LATHE_CCHUCK	LCC
SET_LATHE_UNLOAD	SLUL	O_LATHE_CYCLE_START	oLCS
LATHE_LOAD	LL	I_LATHE_CYCLE_START	iLCS
LATHE_UNLOAD	LU	end_JOB_PT022	eJP
Start_CLOSE_LATHE_DOOR	sCD	end_JOB_GT022	eJG2
LATHE_ODOOR1	LO1	end_JOB_PT-GENER	eJP2
PULSE_CDOOR	PCD	TO_GET023	TOG

Fonte: Autores (2020).

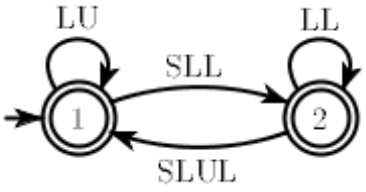
Para fim de tratabilidade e abstração do modelo, foram identificadas algumas condições que poderiam ser simplificadas, como por exemplo na Figura 11(a), onde há quatro eventos de copiar posição. Por se tratar de uma mesma operação, esses eventos foram simplificados em apenas um, como pode ser visto na Figura 11(b). Outro exemplo de simplificação refere-se ao evento que representa o fim de uma rotina, que para a rotina *GET023*, trata-se do evento *eG*. Para iniciar uma tarefa, o *MANAGER* envia um comando ao *SCORBASE* para iniciar a execução de uma rotina. No caso do fim dessa rotina, o evento *eG* não indica uma comunicação entre *SCORBASE* e *MANAGER*, a comunicação é identificada no passo anterior, quando o SCORBASE envia uma mensagem de fim tarefa ao *MANAGER*. Por essa razão, o evento que indica comunicação entre *SCORBASE* e *MANAGER* passa a ser o evento de fim de rotina, como visto no modelo simplificado da Figura 11(b).

Para as demais rotinas principais do *SCORBASE*, que são, *PUT022*, *START_CYCLE_LATHE*, *GET022* e *PUT025*, foram feitas as mesmas considerações para simplificação dos modelos.

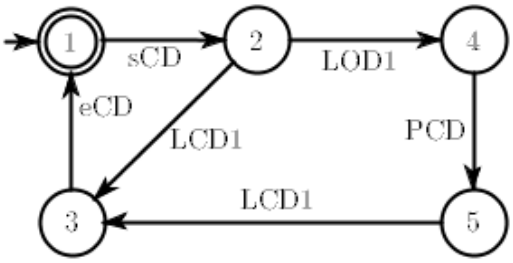
Existem variáveis que são usadas para controle na memória do SCORBASE que também foram modeladas, por exemplo, as variáveis *SLL*, *SLUL* e *PRN*, que representam respectivamente, torno sendo carregado, torno sendo descarregado e o rack que se deve acomodar a peça torneada. O modelo feito para a rotina *SET_LATHE_LOAD* pode ser vista na Figura 12(a). De maneira análoga, os modelos paras as outras variáveis de memória seguiram a mesma premissa.

Figura 12: Autômatos Referentes as Rotinas do SCORBASE.

(a) SET_LATHE_LOAD



(b) CLOSE_LATHE_DOOR



PREAPARE_LATHE_LOAD

Fonte: Autores (2020).

Ainda dentro da modelagem do SCORBASE, existem outras rotinas que são executadas, são elas: *CLOSE_LATHE_DOOR*, *PREPARE_LATHE_LOAD*, *PREPARE_LATHE_UNLOAD* e *WAIT_CYCLE_END_LATHE*, responsáveis por fechar a porta do torno, por indicar quando o torno foi carregado ou descarregado e por finalizar o funcionamento da CNC, respectivamente. A Figura 12(b) mostra o modelo feito para a rotina *CLOSE_LATHE_DOOR* e a Figura 12(c) mostra o modelo feito para a rotina *PREPARE_LATHE_UNLOAD*. Os demais modelos foram feitos de maneira análoga.

A modelagem completa feita para a estação de torneamento é apresentada em (Hilario, 2019).

Para a realização da síntese do supervisor, serão considerados os autômatos simplificados, pois assim a síntese se torna mais tratável, mas se o objetivo fosse automatizar o processo, essas considerações teriam que ser revistas, de forma que a simplificação não poderia ser aplicada ou então a automatização deveria lidar com as simplificações feitas.

4.7.3 Rotinas do *FS100*

O *FS100*é o controlador que lida com as ações do robô manipulador e com algumas entradas e saídas de baixo nível, como por exemplo abrir e fechar a garra e parte do funcionamento da porta do torno. As atribuições do *FS100* podem ser vistas na Figura 9. Na Figura 13 é possível observar as linhas de código que são executadas pela rotina *JOB_GT023*, que é responsável por realizar as movimentações do robô (*OOG*, *OCG*, *sML* e *sMJ*) para que ele possa pegar a peça no alimentador gravitacional. O comando *sML* representa o movimento do efetuador final do robô para percorrer uma linha reta mesmo que haja mudança do ângulo e é reservado para movimentos simples e que exigem precisão do efetuador final, como por exemplo, realizar uma aproximação à peça com o intuito de pegá-la. Para movimentos complexos do braço que não necessitam de precisão no efetuador final, usa-se o comando *sMJ*, em que é considerado apenas o ponto inicial e o ponto final do efetuador final, sendo que o robô irá encontrar a melhor configuração de movimento das juntas com o intuito de evitar ao máximo as singularidades (Corporation Yaskawa Electric, 2019).

A Figura 14(a) mostra o autômato criado a partir da rotina mostrada na Figura 13. De maneira análoga ao modelo do SCORBASE, também foram feitas abstrações ou simplificações no modelo do *FS100*. Os comandos de movimentação do robô, seja de movimento em linha reta ou rotacional, *sML* e *sMJ*, respectivamente, se tornaram apenas um comando de movimentação *sM*, como pode ser vista na Figura 14(b).

Assim como foi feito para o *JOB_GT023*, também foram modeladas as rotinas principais do *FS100*, são elas: *JOB_PT022, JOB_GT022 e JOB_PT025*.

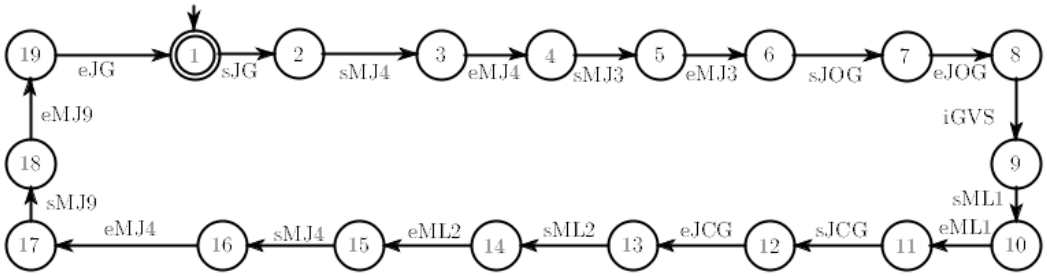
Figura 13: Rotina JOB_GT023 (FS100).

```
1 MOVJ P1004
2 MOVJ P1003
3 CALL JOB: OGRIP
4 WAIT IN#(1) = ON
5 MOVL P1001
6 CALL JOB: CGRIP
7 MOVL P1002
8 MOVJ P1004
9 MOVJ P0999
10 END
```

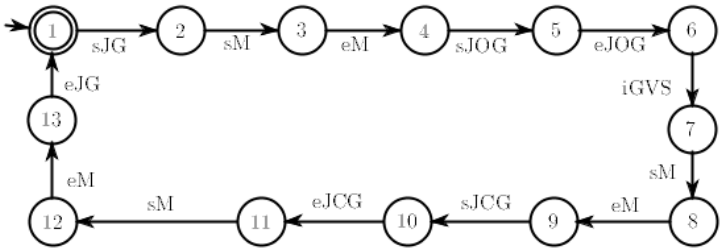
Fonte: Autores (2020).

Figura 14: Autômatos Referentes a Rotina JOB_GT023.

(a) Completo



(b) Simplificado

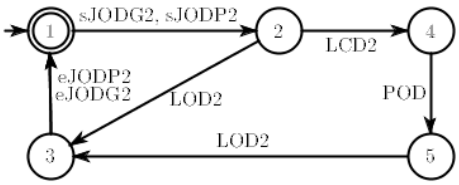


Fonte: Autores (2020).

O *FS100* também executa rotinas que acionam os pontos de entrada e saída lógico do controlador. Essas rotinas referem-se ao acionamento da garra do robô, ao acionamento da morsa do robô e ao acionamento da porta do torno. A Figura 15(a) mostra o autômato modelado para acionamento da porta do torno. Para modelagem pela composição síncrona, foi necessário identificar qual rotina principal era responsável por chamar a rotina cada rotina que, por sua vez, executava o comando em questão. Isso foi feito com o intuito de evitar a ocorrência de bloqueios na composição síncrona, pois todas as mesmas instâncias de um evento devem estar sincronizadas para acontecer. Esse procedimento também foi usado em situações em que poderia haver um bloqueio na composição síncrona quando um mesmo evento é chamado por componentes diferentes.

Figura 15: Autômatos Referentes ao Funcionamento da Porta.

(a) JOB_LODOOR



(b) LATHE_DOOR

Fonte: Autores (2020).

4.7.4 Modelos da Parte Física

É possível observar na Figura 9, que se trata de um sistema ciberfísico e, da mesma forma que foram criados modelos de autômatos com base em rotinas de programas, também foram criados modelos que representam o funcionamento físico do sistema.

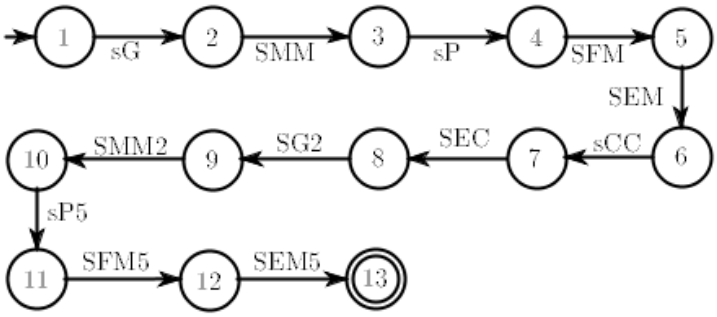
Um exemplo de modelagem considerando o funcionamento físico do sistema pode ser visto na Figura 15(b), em que é possível observar os comandos executado para abrir e fechar a porta, respectivamente *POD* e *PCD*, além de dois eventos que representam o estado da porta, aberta (*LO1* e *LOD2*) e fechada (*LCD1* e *LCD2*). Apesar de existirem dois eventos que indiquem porta aberta e dois eventos que indiquem porta fechada, não significa que existem dois sinais para indicar porta aberta ou fechada, isso foi feito apenas para evitar bloqueio quando for calculado a composição síncrona.

4.7.5 Modelo do *MANAGER*

A comunicação para comandos de tarefas é originária do *MANAGER* para o *SCORBASE* e as respostas são originárias do *SCORBASE* para o *MANAGER*. Então, apresenta-se na Figura 16 a construção do modelo do *MANAGER*.

A premissa de construção da Figura 16 foi de criar uma especificação que representasse a troca de mensagens entre *MANAGER* e *SCORBASE*. É possível observar que esse autômato corresponde à tarefa completa de fabricação do *Sup_Brass*, desde pegar a peça bruta no alimentador gravitacional, até colocá-la no rack após ser torneada.

Figura 16: Especificação de Funcionamento do Sistema da Estação de Torneamento.



Fonte: Autores (2020).

4.8 Resumo da Seção

Nesta seção foi apresentada uma proposta de modelagem visual da planta MECATRIME, por intermédio do diagrama de estrutura, Figura 5, de implantação, Figura 6, e de componentes, Figura 9 foi possível desenvolver um modelo do comportamento dos componentes cibernéticos e físicos da planta na forma de autômatos, que serão empregados na síntese dos supervisores na Seção 4. Para identificar de forma sistemática as possíveis perdas de observação de eventos por perda de comunicação, o diagrama de comunicação, Figura 7, e de componentes, Figura 9, são usados na Seção 4. No próximo capítulo será tratado o problema de controle supervisório tolerante a perdas intermitentes de observação em função da perda de comunicação entre *SCORBASE* e *FS100*.

5. Estudo de Caso de Controle com Perdas Intermitentes de Observação

Será mostrada nesta seção a síntese realizada para encontrar um supervisor que atenda o problema supervisório considerando a observação completa dos eventos e sob perdas intermitentes de observação quando há falha de comunicação.

Emprega-se os diagramas de comunicação, Figura 7, e de componentes, Figura 9, para abordar a perda de observação

de certos eventos por conta da falha de comunicação. Especificamente para este trabalho, a falha de comunicação poderá ocorrer na resposta que o *FS100* envia ao *SCORBASE* quando uma tarefa de produção local é concluída, via *LAN*.

5.1 Identificação das Possíveis Perdas Intermitentes

Nosso objetivo é mostrar o problema das perdas intermitentes de observação e a Tabela 2 mostra os tipos de comunicação que existem na planta MECATRIME, os motivos que podem causar a perda de comunicação e a hipótese que será considerada para a perda de comunicação. A coluna dois da Tabela 2 relaciona o tipo de comunicação com o evento da coluna um, sendo essa relação feita a partir do digrama de componentes, da Figura 9.

Tabela 2: Levantamento das Possíveis Causas para Perda de Observação de Eventos.

Levantamento das Possíveis Causas para Perda de Observação de Eventos.			
Evento	**Tipo de Comunicação**	**Possibilidade de Perda**	**Passíveis de Perdas**
sG	LAN	Comunicação	Perda
eG	LAN	Comunicação	Perda
sP	LAN	Comunicação	Perda
eP	LAN	Comunicação	Perda
sCC	USB/Sinal Elétrico	Comunicação e Problema no Cabeamento	Perda
SMM/SEM/SFM	LAN	Comunicação	Perda
sJG/sJP/ sJG2/sJP	LAN	Comunicação	Perda
eJG/eJP/ eJG2/eJP2	LAN	Comunicação	Perda
sGPB	USB/Dedicado	Comunicação ou Impacto Físico	-
eGPB	USB/Dedicado	Comunicação ou Impacto Físico	-
oLCS	USB/Sinal Elétrico	Comunicação ou Problema no Cabeamento	-
iLCS	USB/Sinal Elétrico	Comunicação ou Problema no Sensor	-
PCD	USB/Sinal Elétrico	Comunicação ou Travamento Físico	-
LCD	USB/Sinal Elétrico	Comunicação ou Problema no Sensor	-
POD	Sinal Elétrico	Mau Contato	-
LOD	Sinal Elétrico	Problema no Sensor/Mau contato	-
OOG	Sinal Elétrico	Mau Contato	-
OCG	Sinal Elétrico	Mau Contato	-
POC	Sinal Elétrico	Mau Contato	-
PCC	Sinal Elétrico	Mau Contato	-
LOC	Sinal Elétrico	Problema no Sensor/Mau contato	-
LCC	Sinal Elétrico	Problema no Sensor/Mau contato	-
Start_MOV	Dedicado	Impacto Físico	-
end_MOV	Dedicado	Impacto Físico	-
iGVS	Sinal Elétrico	Problema no Sensor/Mau contato	-

Fonte: Autores (2020).

A terceira coluna da tabela faz menção das possíveis razões para que haja perda de observação. Neste trabalho, é tratado apenas o problema de perdas intermitentes de observação em função de perda de comunicação. Falhas oriundas de impacto físico que impeça o robô de chegar ao seu fim de curso, por exemplo, não são tratadas. A quarta coluna mostra que inicialmente, os eventos que representam comunicação é que serão considerados como passíveis de perda de observação quando houver perda de comunicação via *LAN*. Falhas decorrentes de problemas físicos serão desconsiderados, da mesma forma em relação à comunicação dedicada, que será considerada como um meio seguro de comunicação ponta a ponta, como com o controlador *USB* e o *LSB*, ou como o *FS100* e o braço robótico. Após esse levantamento, decidiu-se então considerar os eventos que representam comunicação no sentido entre *FS100* e *SCORBASE* (*eJG* ou *eJP*) como os que poderão de fato perder a observação por perda de comunicação. Considera-se assim que os comandos do *MANAGER* conseguem chegar ao SCORBASE.

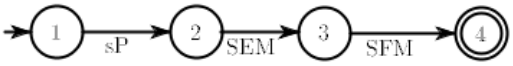
A Figura 17 mostra as especificações obtidas em cada etapa mostrada na Tabela 3. A especificação vista na Figura 16

na verdade é a evolução das especificações de cada etapa vistas na Figura 17.

Figura 7: Especificações para cada Etapa.

(a) Etapa 1

(b) Etapa 2



(c) Etapa 3

(d) Etapa 4

(e) Etapa 5

Fonte: Autores (2020).

Tabela 3: Etapas de Produção.

Etapa	Rotina/Autômato
1	*GET023*
2	*PUT022*
3	*START_CYCLE*
4	*GET022*
5	*PUT025*

Fonte: Autores (2020).

5.2 Conclusões e Sugestões para Trabalhos Futuros

Para conseguir lidar com a complexidade do problema e a limitação existente nos softwares (IDES[1] 3 beta1 (Rudie et al., 2008) e o UltraDES[2] (Pena et al., 2008)), decidiu-se dividir o funcionamento em partes, ou seja, em cada ação realizada em uma produção local, para então realizar a síntese a partir dessa divisão feita. Cada etapa representa a evolução do processo até que a produção de uma peça seja completa. A Tabela 3 mostra a divisão feita por etapa.

Para atribuição da controlabilidade dos eventos, decidiu-se que os eventos de início de tarefa são controláveis, enquanto os eventos de fim de tarefa são não controláveis. Conforme visto, na Seção 4.1, foram identificados os eventos que possuem chances de perdas de comunicação. O critério utilizado para definir se um evento é passível ou não de perda de observação é se o evento representa uma resposta a um comando por meio da rede *LAN*, como por exemplo, os eventos *SMM*, *SEM* e *SFM*, que representam a reposta para o evento *sG*, *sG2*, *sP* ou *sP2* Como primeira abordagem, foi considerado que a perda de observação será devido à falha de comunicação entre *SCORBASE* e *FS100*, ou seja, a resposta de tarefa realizada (*eJG*, *eJG2*, *eJP* ou *eJP2*) poderá não ser recebida pelo *SCORBASE*. A Tabela 4 mostra os eventos em que há possibilidade de

1 Desenvolvido no Laboratório DES da Queen's University, sob a supervisão da Profª. Karen Rudie.

2 Desenvolvido no Laboratório de Análise e Controle de Sistemas a Eventos Discretos (LACSED) da Universidade Federal de Minas Gerais, sob supervisão da Profª. Patrícia Nascimento Pena.

perda em cada etapa da produção local.

A Tabela 4 também mostra a versão observável e a não observável dos eventos passíveis de perda intermitente de observação. Pode-se observar que na etapa 3 não há evento passível de perda, isso se dá pelo fato de que nesta etapa não há comunicação entre *SCORBASE* e *FS100*.

Tabela 4: Eventos Passíveis de Perda de Observação.

Etapa	Evento	Versão Observável	Versão Não Observável
1	*eJG*	*eJG*	*eJG'*
2	*eJP*	*eJP*	*eJP'*
3	-	-	-
4	*eJG2*	*eJG2*	*eJG2'*
5	*eJP2*	*eJP2*	*eJP2'*

Fonte: Autores (2020).

A representação do evento passível de perda pode ser vista na Figura 18(a), que mostra o autômato dilatado referente à rotina *GET023* e a Figura 18(b) mostra a dilatação feita no autômato referente à rotina *JOB_GT023*. Além da dilatação, o autômato referente à rotina *JOB_GT023* teve todos os seus estados marcados, indicando que se houver perda de comunicação entre *SCORBASE* e *FS100*, qualquer estado dos autômatos se torna um estado alvo.

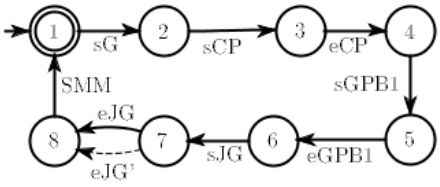
Após dilatar os eventos passíveis de perda de observação, foi realizada a síntese de um supervisor robusto, pelo cálculo da suprema sublinguagem controlável e normal (*SupCN*), empregando o passo a passo da Seção 2.2. Como dito no início, dividiu-se o problema em tarefas e por isso, foi calculado um supervisor para cada tarefa que o *MANAGER* manda executar. Vale ressaltar que para essa abordagem a especificação de cada etapa pode ser vista na Figura 17.

A Tabela 5 mostra o resultado ao se calcular o supervisor utilizando a abordagem de perda de observação e na Figura 19 é mostrado o supervisor encontrado referente a etapa um, em que é possível perceber a ocorrência de uma incoerência permitida pelo supervisor. Na Figura 19 percebe-se a permissão de que uma resposta de fim de tarefa possa ser enviada ao *MANAGER* mesmo que o *FS100* não tenha indicado ao *SCORBASE* que a conclusão dessa tarefa foi feita.

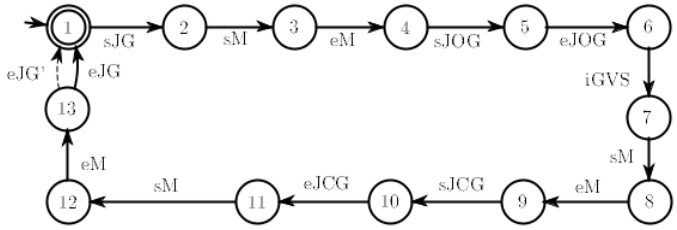
Diante disso, pensou-se em uma nova abordagem com o intuito de evitar a ocorrência desse fato e que será apresentada na seção seguinte.

Figura 8: Autômatos Dilatados.

(a) *GET023*



(b) *JOB_GT023*



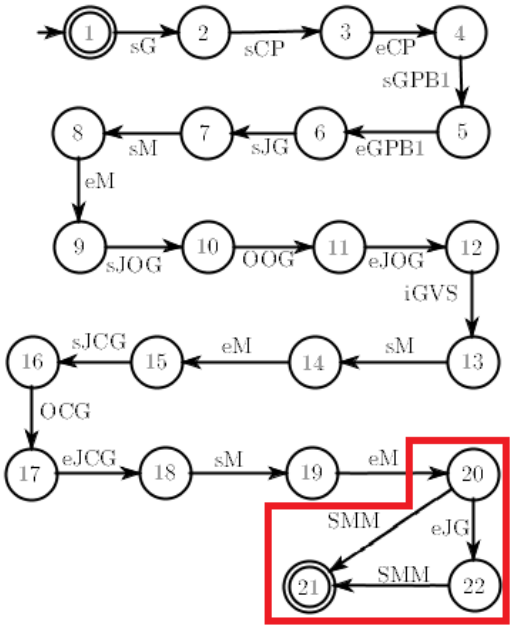
Fonte: Autores (2020).

Tabela 5: Resultado Obtido na Síntese Utilizando a Primeira Abordagem.

Etapa	Planta	Especificação	Ling. ALvo	SupCN	Obs(SupCN)
1	*21*	3	22	23	22
2	*47*	4	45	46	45
3	15	3	8	8	8
4	*37*	3	35	36	35
5	*78*	4	19	19	19

Fonte: Autores (2020).

Figura 9: Supervisor da Etapa 1 Apresentando Incoerência.



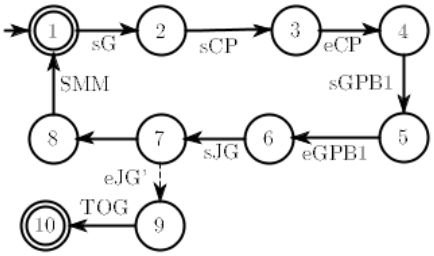
Fonte: Autores (2020).

5.3 Conclusões e Sugestões para Trabalhos Futuros

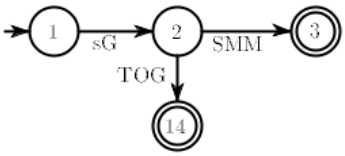
O novo autômato criado para corrigir a incoerência vista na Figura 19 foi um refinamento feito no funcionamento do sistema que permitisse ao *MANAGER* receber uma indicação de que a tarefa não foi concluída por ter ocorrido falha na comunicação. O autômato criado pode ser visto na Figura 20(a), em que é possível perceber que a nova abordagem adotada consiste em criar um caminho a partir da versão não observável do evento passível de observação, fazendo que esse caminho chegue até um estado de *time_out*. Esse raciocínio foi adotado para os demais autômatos que representam as rotinas do *SCORBASE*. Em relação aos autômatos referentes às rotinas do *FS100*, a modelagem utilizada foi a mesma da abordagem anterior, como mostrada na Figura 18(b).

Figura 10: Nova Abordagem.

(a) *GET_023*



(b) Especificação da Etapa 1



Fonte: Autores (2020).

O autômato que representa a especificação de funcionamento do sistema também teve uma modificação para que fosse possível atender a premissa da nova abordagem. A nova configuração da especificação pode ser verificada na Figura 20(b) e é possível observar que a modificação feita foi apenas acrescentar o estado de *time_out* antes da ocorrência do evento que indica o envio da mensagem do *SCORBASE* para o *MANAGER*, permitindo então, que o *MANAGER* perceba a perda de comunicação. O estado de *time_out* trata o tempo que o *SCORBASE* espera para receber uma mensagem de fim de tarefa por parte do *FS100*. Esse tempo de espera é indicado pela linha 10 do *scipt* da rotina do *SCORBASE*, como pode ser visto na Figura 10.

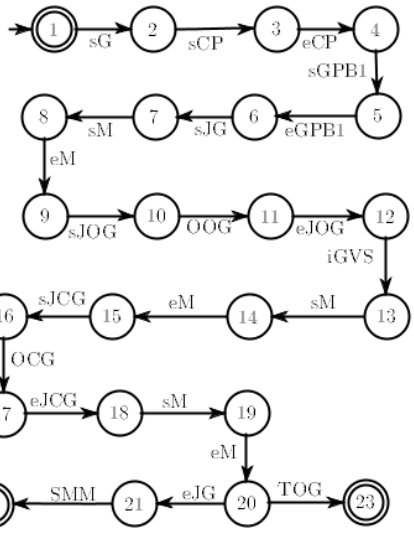
A Tabela 6 mostra os resultados obtidos em cada etapa ao utilizar a abordagem de criar um estado de *time_out*. Pode-se observar que entre a Tabela 5 e a Tabela 6 são os estados a mais que foram criados na segunda abordagem. A Figura 21 mostra o resultado obtido na etapa um. É possível perceber que a tarefa só será finalizada quando a comunicação entre *SCORBASE* e *FS100* for estabelecida, em que é possível observar que, caso contrário, o sistema irá para um estado de *time_out*, diferente do que ocorreu com a abordagem anterior, mostrada na Figura 19, em que o supervisor permitia o envio da mensagem de tarefa realizada mesmo que tivesse ocorrido uma falha de comunicação.

Tabela 6: Resultado Obtido na Síntese na Segunda Abordagem.

Etapa	Planta	Especificação	Ling. ALvo	SupCN	Obs(SupCN)
1	*21*	3	22	23	22
2	*47*	4	45	46	45
3	15	3	8	8	8
4	*37*	3	35	36	35
5	*78*	4	19	19	19

Fonte: Autores (2020).

Figura 11: Supervisor da Etapa1 Utilizando Nova Abordagem.



Fonte: Autores (2020).

Na Seção 3.7.2 foi apresentado um Script referente a programação da rotina *GET023*, sendo que existe uma linha de comando (*Wait FS100_DELAY_TIME* (*10ths of seconds*)) que acusa a falta de comunicação entre *SCORBASE* e *FS100*, mas essa falha de comunicação não é tratada pelo *MANAGER*, pois ela não é transmitida pelo *SCORBASE*. Para que a falha de comunicação possa ser tratada pelo *MANAGER*, basta acrescentar uma linha de comando que indique que a comunicação entre *SCORBASE* e *FS100* não foi realizada, podendo assim então, evitar que a produção local fique parada.

A seguir, na Figura 22, é dado um exemplo de como poderia ser feito o novo *Script* da rotina *GET023*, sendo que foram adicionadas algumas linhas de comando para que fosse verificado se a comunicação foi estabelecida e que enviasse uma mensagem ao *MANAGER* caso contrário.

Verificou-se também que a variável *QCR*, destacada de vermelho, é observada pelo *MANAGER* quando a comunicação entre *SCORBASE* e *FS100* não for bem-sucedida, permitindo então, que o *MANAGER* tome decisões baseadas na perda de comunicação. Para garantir que o sistema funcione no caso sem perdas de comunicação, ou seja, não permita o envio de mensagem de fim de tarefa por parte do *SCORBASE* quando a tarefa não for concluída, pensou-se então no recurso de "pular" linha logo após o apontamento de perda de comunicação.

Figura 12: Rotina *GET_023* com Indicação de Perda de Comunicação.

```
1 Set Subroutine GET023
2 Call Subroutine SCRIPT.GET_FROM_GFRD1
3 Copy FS100 Position SCRIPT.P1 to Position 1001
4 Copy FS100 Position SCRIPT.P2 to Position 1002
5 Copy FS100 Position SCRIPT.P3 to Position 1003
6 Copy FS100 Position SCRIPT.P4 to Position 1004
7 Go to Position SCRIPT.PB1 Speed 50 (%)
8 FS100 Start Job GT023
9 Wait FS100_DELAY_TIME (10ths of seconds)
10 If FS100 Job Wait < 61 Jump to COMMUNI-
CATION_OK
11 QCR = "FAIL"
12 Jump to SYS_FAIL
13 COMMUNICATION_OK:
14 Send Message $Start to MANAGER ID = TASK_ID
15 SYS_FAIL:
16 Return from Subroutine
```

Fonte: Autores (2020).

6. Conclusões e Sugestões para Trabalhos Futuros

6.1 Pontos Abordados ou Contribuições

Neste trabalho foi apresentado o estudo de caso tratando do problema de controle supervisório robusto em uma produção local da estação de torneamento da planta MECATRIME, com o objetivo de identificar e então tratar os possíveis problemas de perdas intermitentes de observação devido a falhas de comunicação. A identificação dessas perdas foi feita por meio da construção de diagramas da Engenharia de Sistemas, que auxiliaram na abordagem do sistema para que fosse possível verificar os pontos em que ocorriam as perdas e para discernir quais perdas eram oriundas de falhas de comunicação. Também foi proposta uma solução para o problema e foram definidas as condições de existência da solução.

Foi identificado que a abordagem pura, com perdas, permitiu que falhas de comunicação fossem desconsideradas na coordenação global, levando ao sistema para uma situação insegura. Uma proposta que refina o comportamento de falha foi adicionada e demonstrou-se a eficácia da abordagem e uma possível forma de implementação. Essa mesma abordagem permitiu mostrar o aprimoramento que se fez necessário no modelo de autômatos para encontrar um supervisor que atendesse aos requisitos de teoria necessários e consequentemente ao real funcionamento do sistema. Foi possível também, reconhecer a necessidade de aprimoramento nos programas que executam as rotinas de códigos, com o objetivo de dar mais dinamismo ao sistema na ocorrência de perda de observação quando há perda de comunicação.

Com este trabalho também foi possível mostrar a importância de realizar a construção de diagramas da engenharia de sistemas para se conhecer mais detalhes de um sistema, além de realizar uma modelagem padronizada, permitindo assim que trabalhos futuros abordados nesse sistema possam obter informações com mais facilidade e clareza.

6.2 Pontos não Abordados ou Limitações

Neste trabalho não foram consideradas as abordagens de observabilidade relativa e observável para a síntese de um supervisor. Foi considerado apenas a abordagem da suprema sublinguagem controlável e normal.

Apesar de fazer um apanhado geral sobre o funcionamento da planta MECATRIME, apenas a estação de torneamento foi tratada em detalhes, considerando apenas a perda de observação por falha de comunicação entre *SCORBASE* e *FS100*. Há outros pontos nessa estação em que não foram consideradas perdas.

As demais estações que compõem o sistema não foram alvo do estudo de caso proposto neste trabalho devido à

grande complexidade que ele possui.

Os softwares utilizados para a síntese do supervisor tiveram suas limitações evidenciadas no desenvolver deste trabalho. O IDES 3 beta 1, por exemplo, teve sua limitação em relação à capacidade de memória, em que não foi possível realizar o cálculo do supervisor nas etapas quatro e cinco do estudo de caso. Nessas duas últimas etapas, foi utilizada a biblioteca open-source, *UltraDES*, desenvolvida em linguagem C#. Essa biblioteca possui diversas funcionalidades para aplicação em SEDs, mas para este trabalho, sua limitação se deve ao fato de não ser possível definir eventos não observáveis.

Diante desses pontos não abordados ou pelas limitações encontradas, na próxima seção são feitas algumas sugestões para trabalhos futuros.

6.3 Sugestões para Trabalho Futuros

Como sugestão para trabalhos futuros, recomenda-se realizar a síntese utilizando a abordagem de observabilidade relativa, por meio da suprema sublinguagem relativa e observável e das ínfimas superlinguagens e assim fazer a comparação dos resultados obtidos em cada abordagem.

Outra recomendação é expandir este trabalho para as demais estações que compõem a planta MECATRIME, reproduzindo os diagramas construídos para a estação de torneamento, diminuindo assim a complexidade de cada uma delas e assim, mostrar a estrutura de componentes do sistema, a organização do conjunto de elementos, como os dispositivos conversam entre si na rede, e mostrar o comportamento dinâmico do sistema.

Uma terceira recomendação é acrescentar mais pontos de perda de comunicação nos modelos de autômatos com o intuito de realizar a síntese de um supervisor em que abranja ao máximo essas possíveis perdas de observação.

Uma última sugestão é implementar novas funcionalidades na biblioteca *UltraDES* que permitam realizar a síntese ao abordar eventos não observáveis.

Referências

Alves, M. V. (2 de 2014). Controle Supervisório Robusto de Sistemas a Eventos Discretos Sujeitos a Perdas Intermitentes de Observação. Master's thesis, Universidade Federal do Rio de Janeiro.

Alves, M. V. (11 de 2017). Supervisory Control of Networked Discrete Event Systems with Timing Structure. Ph.D. dissertation, Universidade Federal do Rio de Janeiro.

Alves, M. V., Basílio, J. C., Cunha, A. E., Carvalho, L. K., & Moreira, M. V. (5 de 2014). Robust Supervisory Control Against intermittent Loss Observations. 12th IFAC/IEEE Workshop on Discrete Event Systems, (pp. 294-299). Cachan.

Alves, M. V., Carvalho, L. K., & Basilio, J. C. (11 de 2017). New Algorithms for Verification of Relative Observability and Computation of Supremal Relatively Observable Sublanguage. IEEE Transactions on Automatic Control, 62, 5902-5908.

Alves, M. V., Cunha, A. E., Carvalho, L. K., Moreira, M. V., & Basilio, J. C. (1 de 2019). Robust Supervisory Control of Discrete Event Systems Against Intermittent Loss of Observations. International Journal of Control.

Brandenbourger, B., & Durand, F. (9 de 2018). Design Pattern for Decomposition or Aggregation of Automation Systems into Hierarchy Levels. 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), 1, 895-901. 10.1109/ETFA.2018.8502627

Burgoyne, J.; James, K. T. Towards Best or Better Practice in Corporate Leadership Development: Operational Issues in Mode 2 and Design Science research. British Journal of Management, v. 17, p. 303-316, 2006. http://dx.doi. org/10.1111/j.1467-8551.2005.00468.x

Cai, K., Zhang, R., & Wonham, W. M. (3 de 2015). Relative Observability of Discrete-Event Systems and Its Supremal Sublanguages. IEEE Transactions on Automatic Control, 60, 659-670.

Cai, K., Zhang, R., & Wonham, W. M. (1 de 2016). Correction to "Relative Observability of Discrete-Event Systems and Its Supremal Sublanguages". IEEE Transactions on Automatic Control, 62, 511.

Cárdenas, A. A., Amim, S., & Sastry, S. (2008). Secure Control: Towards Survivable Cyber-Physical Systems. 28th Internacional Conference on Distributed Computing Systmes Workshops, (pp. 495-500).

Carvalho, L. K., Moreira, M. V., & Basilio, J. C. (2012). Generalized Robust Diagnosability of Discrete Event Systems. Automatica, 48, 2068-2078.

Carvalho, L. K., Moreira, M. V., & Basilio, J. C. (2017). Diagnosability of intermittent sensor faults in discrete event systems. Automatica, 79, 315-325.

Cassandras, C. G., & Lafortune, S. (2008). Introducion to Discrete Event Systems (2a ed.). Springer.

CORPORATION YASKAWA ELECTRIC. (8 de 2014). FS100 OPERATOR'S MANUAL. Tech. rep., Yaskawa Electric Corporation.

Da Costa, C. (9 de 2017). Indústria 4.0: O Futuro da Indústria Nacional. Pós-Graduação em Revista - IFSP, 1, 5-14.

Da Costa, S. E. G., & De Lima, E. P. (2011) Processos: Uma abordagem da Engenharia para a Gestão de Operações. campus, 63-72.

Golçalves, E. J., & Cortés, M. I. (2015). Computação: Análise e Projeto de Sistemas (3a ed.). (E. Universidade Estadual do Ceará, Ed.) Fortaleza.

Guedes, G. T. (2011). UML 2 Uma Abordagem Prática (2a ed.). Novatec.

Hilario, R. Q. (10 de 2019). Estudo de Caso de Controle Supervisório Tolerante a Perdas Intermitentes de Observação: Planta MECATRIME. Masther's thesis, Instituto Militar de Engenharia.

Kruchten, P. B. (1995). The 4+1 View Model of Architecture. IEEE Software, 126, 42-50.

Lacerda, D. P., Dresch, A., Proença, A., Júnior, J. A. V. A. Design Science Research: método de pesquisa para a engenharia de produção. IEEE Software, 126, 42-50.

Lin, F. (2014). Control of Networked Discrete Eevent Systems: Dealing With Communication Delays And Losses. SIAM Journal on Control and Optimization, 52, pp. 1276-1298.

Paoli, A., Sartini, M., & Lafortune, S. (2011). Active Fault Tolerant Control of Discrete Event Systems Using Online Diagnostics. Em Automatica 47, 639-649.

Pena, P. N., Alves, L. V., Chagas Alves, M. R., & Rafael, G. C. (s.d.). UltraDES. Fonte: https://lacsed.eng.ufmg.br/ultrades

Pimentel, A. R. (9 de 2015). Projeto de Software Usando a UML. Tech. rep.

Pinheiro, Á. F. (12 de 2016). Série Fundamentos da Engenharia de Software: Linguagem de Modelagem Unificada. Recife: Revista de Publicação Independente.

Platts, K. W. (1993). A Process Approach to Researching Manufacturing Strategy. International Journal of Operations & Production Management, 13(8), 4-17 http://dx.doi. org/10.1108/01443579310039533

Platts, K. W. et al. (1998).Testing manufacturing strategy formulation processes. International Journal of Production Economics, 56-57, 517-523

Rajkumar, R., Lee, I., Sha, L., & Stankovic, J. (2010). Cyber-Physical Systems: The Next Computing Revolution. Design Automation Conference, (731-736).

Ramadge, P. J., & Wonham, W. M. (1 de 1989). The Control of Discrete Event Systems. Proceendings of the IEEE, 77, 81-98.

Rohloff, K. R. (12 de 2005). Failure Tolerant Supervisory Control. 44th IEEE Conference on Decision and Control, and the European Control Conference, (pp. 3493-3498). Seville.

Rohloff, K. R. (2012). Bounded Sensor Failure Tolerant Supervisory Control. 11th IFAC Workshop on Discrete Event Systems, 45, 272-277. Guadalajara, México.

Romme, A. G. L. Making a difference: Organization as Design. Organization Science, 14(5), 558-573, 2003. http://dx.doi.org/10.1287/ orsc.14.5.558.16769

Rudie, K., Nguyen, P., Wood, M., Grigorov, L., Edlund, K., Michelsen, A. G., Sugarman., V. (s.d.). IDES 3 beta 1. Fonte: https://qshare.queensu.ca/Users01/rudie/www/software.html

Sánchez, A. M., & Montoya, F. J. (2006). Safe Supervisory Control Under Observability Failure. Discrete Event Dynamic Systems, 16, 493-525.

Simon, H. A. (1996). The Sciences of the Artificial. (3rd ed.), MIT Press

Sommerville, I. (2011). Engenharia de Software (9a ed.). Pearson Prentice Hall.

Trivelato, G. D. (2003). Técnicas de Modelagem e Simulação de Sistemas Dinâmicos. Tech. rep., Ministério da Ciência e Tecnologia. Instituto Nacional de Pesquisas Espaciais - INPE.

Van Aken, J. E. Management Research Based on the Paradigm of the Design Sciences: The Quest for Field- Tested and Grounded Technological Rules. Journal of Management Studies, 41(2), 219-246, 2004. http://dx.doi.org/10.1111/j.1467-6486.2004.00430.x

Wang, E. K., Ye, Y., Xiaofei Xu, S.M.Yiu, L.C.K.Hui, & K.P.Chow. (2010). Security Issues and Challenges for Cyber Physical System. International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing, (pp. 733-738).

Wonham, W. M., & Cai, K. (5 de 2017). Supervisory Control of Discrete-Event Systems. http://www.control.utoronto.ca/cgi-bin/dldes.cgi